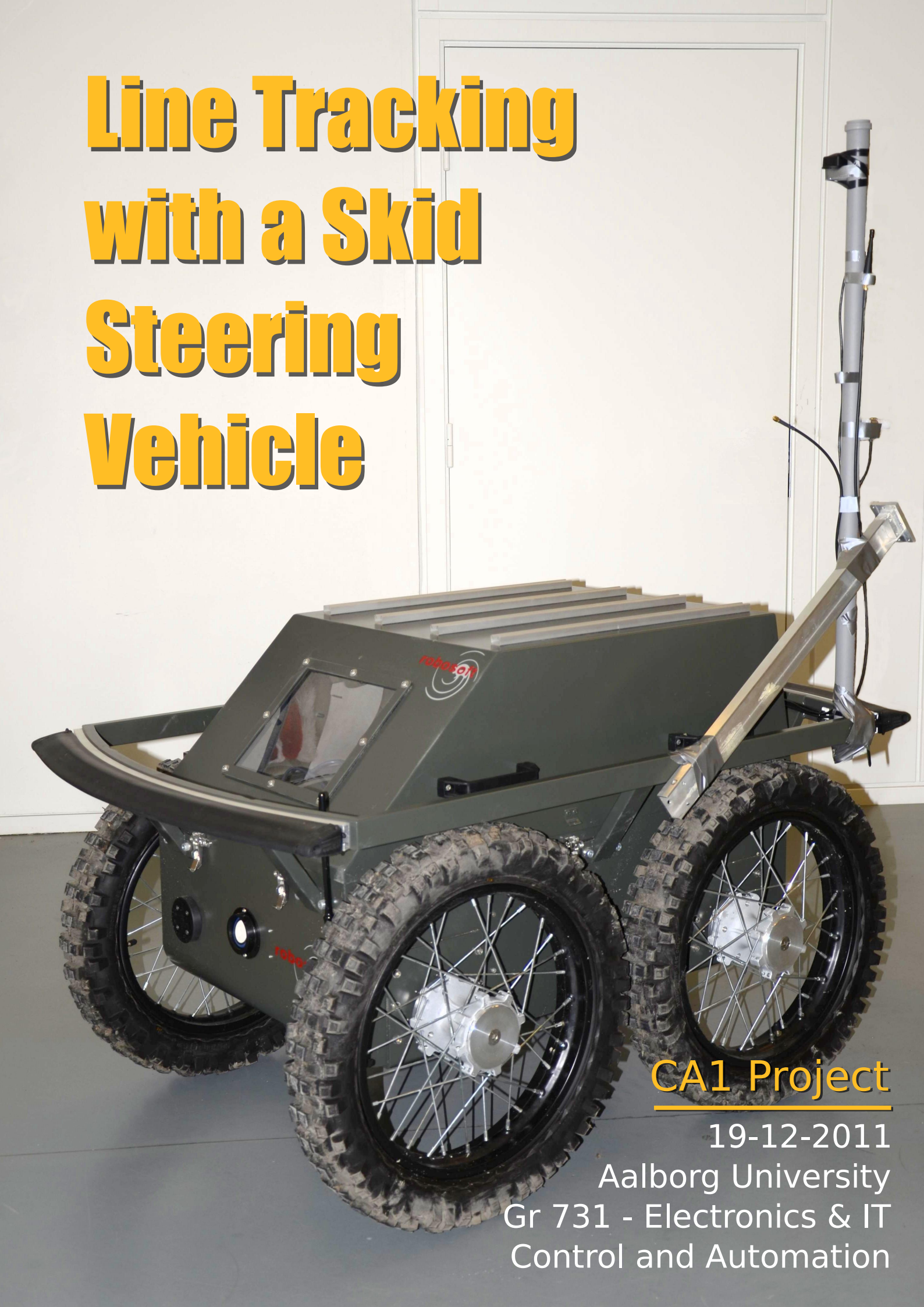
Line Tracking with a Skid Steering Vehicle



CA1 Project

19-12-2011

Aalborg University
Gr 731 - Electronics & IT
Control and Automation

AALBORG UNIVERSITY

Department of Electronic systems
Electronics & IT
Fredrik Bajers Vej 7
9220 Aalborg Ø
Tel. 99 40 86 00
www.sict.aau.dk

Title:

Line Tracking with a Skid Steering Vehicle

Subject:

Networked Control System

Project period:

September 2nd - December 19th, 2011

Group:

Group 731

Group members:

Niels H. Andersen
Mikkel U. Kajgaard
Rune Madsen
Jesper Mogensen
Anders Wittendorff

Supervisors:

Anders la Cour-Harbo
Karl Damkjær Hansen

Copies: 8

Page count: 76

Attachments: CD + Paper

Appendix: 2

Completion: 19^{th} December 2011

Synopsis:

This project aims at designing a control system for an Unmanned Ground Vehicle (UGV) that makes this capable of steering onto and follow straight lines given by waypoints sent wirelessly from a Mission Control Center (MCC). The project originates in the ASETA project, which suggests usage of UGVs for early treatment of crops in a field. For this project a four wheeled skid steering vehicle is provided. The vehicle is fitted with GPS and a magnetometer to provide information on position and heading of the vehicle respectively.
A model is from the kinematics and dynamics of the vehicle determined. A controller structure is inspired by and designed from how a "good helmsman" would steer a boat onto a straight line. A communication protocol is designed to make it possible to send and receive data from the UGV over a wireless connection between UGV and MCC.
Requirements are set for the system, and finally an acceptance test is carried out to determine the functionality of the controller structure and the protocol. The test showed that the overall functionality of the system works but due to sensor problems the desired precision could not be obtained.

Worksheets Reading Guide

This collection of worksheets, is alongside a paper, the result of the 1st semester of the master programme in control and automation, produced by group 731 in fall 2011 at Aalborg University. The main purpose for this semester project is to create a network based control system with focus on scientific communication. The project is based on the ASETA project, which considers the possibilities and benefits of using unmanned robots for agricultural work.

The worksheets are worked out as standalone worksheets each given a number. Some worksheets contain sections with more detailed descriptions of the aspect, which again can hold subsections for more specific detailed purposes. Worksheets, sections and subsections are numbered. E.g worksheet 3, section 2, subsection 1 is numbered 3.2.1.

In the worksheets several acronyms are used. These are defined and found in the back of the collection on page 63.

Throughout the worksheets, there is referred to several manuals and datasheets, for the used equipment for this project. These references are followed by a ◎ meaning that the datasheet or manual can be found on the enclosed CD with a descriptive name. Alongside with the datasheets and manuals, the obtained data used for plotting graphs for the worksheets are found on the CD. Furthermore the MATLAB files used to create plots and graphs are found in format **.m* and Simulink models as **.mdl*. A digital version of the worksheets along with the paper is to be found on the CD as well. References to specific files on the CD are formatted in this way: ◎ `/folder/filename.ext`.

Niels Hyltoft Andersen Mikkel Urban Kajgaard Rune Madsen

Jesper Mogensen Anders Wittendorff

Contents

1 Introduction

This project spawns from the Adaptive Surveying and Early treatment of crops with a Team of Autonomous vehicles (ASETA) project. The goal of the project is using Unmaned Aircrafts (UAs) and UGVs for agricultural purposes [1]. The purpose of using UAs and UGVs is to optimize the chemical treatment of weed in order to reduce the amount of chemicals used. It is beneficial to treat weed at an early stage of growth, before it begins to compete with the crops by stealing nutrients and sunlight.
The idea of the ASETA project is to have a team of autonomous robots surveying the field and detect growing weed. The autonomous robots are supposed to locate specific areas with weed, thereby the treatment can be site specific instead of the traditional methods where the whole field is treated. The usage of chemicals over the whole field is time consuming and expensive due to using man driven agricultural machinery, further more it is polluting the environment unnecessarily if the chemicals are spread where there is no weed. The use of intelligent autonomous robots could therefore in the long term be beneficial for both the farmers economy and the environment.

The ASETA project takes point of origin in four hypotheses [1]:

1. Localized detection and treatment for weeds will significantly decrease the need for pesticides and fuel and thereby reduce environmental impact.
2. Such early detection can be accomplished by multi-scale sensing of the crop fields by having UAs surveying the field and performing closer inspection of detected anomalies.
3. A team of autonomous UAs and UGVs can be guided to make close-to-crop measurements and to apply targeted treatment on infested areas.
4. A team of relatively few vehicles can be made to perform high level tasks through close cooperation and thereby achieve what no one vehicle can accomplish alone.

To achieve an effective treatment of the crops within a reasonable price tag and time frame compared to existing manual solutions, advantages of different kinds of robots must be utilised.

UAs are able to perform a fast detection of weed over a large area by flying over the field while collecting data. UGVs can be used to perform actions that requires close contact with the weed and have the advantage of being able to carry a heavy payload e.g. a tank of pesticides.

In the ASETA project sugar beets have been chosen as test crop for proving the described concept. A field of sugar beets is therefore being maintained at University of Copenhagen to allow real life tests of the system.

Throughout this project focus will be upon designing software for a single UGV, specifically the RobuROC4 [2], and implement a controller that enables it to drive to a series of waypoints within a field. These waypoints are expected to be planned and provided from an external MCC. Planning of these waypoints is not within the scope of the project.

To allow a wireless reception of waypoints and exchanging information of current status, a transfer protocol between the MCC and the RobuROC4 must also be developed.

From the above described problem, this project will take point of origin in the following problem statement:

How can a software based solution be designed and implemented, which can receive waypoints via a wireless connection and navigate the RobuROC4 to these waypoints in a way that limits damage inflicted on the crops.

2 Operational Setup

This worksheet presents the operational setup for the project and the used hardware. First the vehicles task in the field is described alongside with a description of the outline of the field. Afterwards the used hardware and the composition of it is presented.

The operational setup is a reduced version of the entire ASETA project [1] from which this project takes its point of origin. It consists of an Unmanned Ground Vehicle (UGV) driving in a beet field, navigating by utilising a Global Positioning System (GPS) and a magnetometer. Typically in beet fields, the beets are planted with a spacing of 50 cm [3] [4]. This row spacing makes the accuracy of normal GPS technology insufficient since it is only precise down to meters for moving objects [5]. For this reason a Real Time Kinematic (RTK) system is added, which is able to obtain precision down to centimeters. A Mission Control Center (MCC) is set up with wireless communication, from which waypoints and commands are sent to the UGV. The UGV will respond wirelessly with the data requested and status on the drive. The setup is illustrated in figure 2.1.

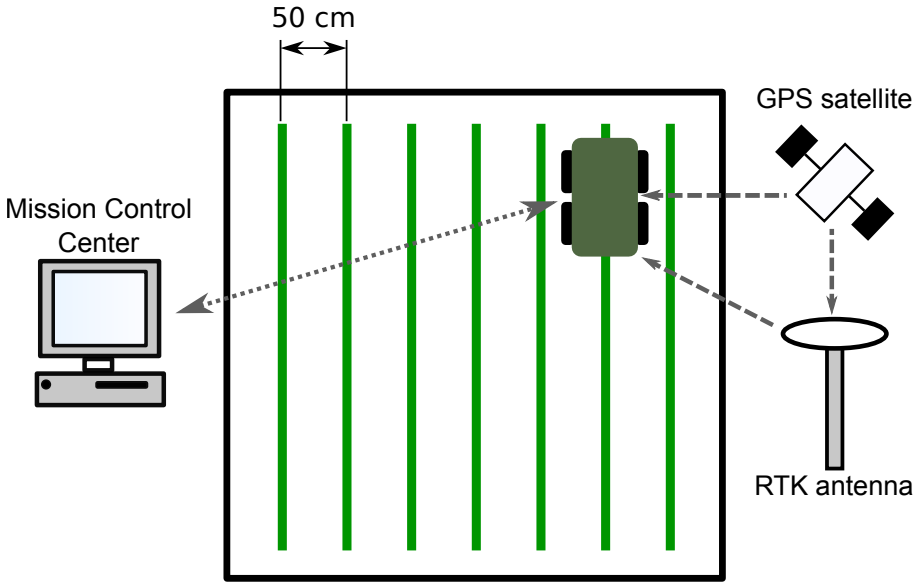


Figure 2.1. UGV driving in a beet field by receiving waypoints and commands from MCC, its location is determined by RTK GPS.

In figure 2.1, the beet field is illustrated with the beets planted in straight lines which is most common. This of course may vary as a consequence of an irregular shape of the field or obstacles such as small lakes or trees. Also the surface of the field can be alternating, meaning that sometimes the UGV will experience variation in friction, which is assumed to cause slippage.

The RTK GPS setup consists of a GPS module fitted on the UGV, and a stationary base station with GPS antenna and a radio module. The base station uses the fact that it is not it moving to determine a more precise position of itself. By knowing the precise position of the base station, it is capable of broadcasting correction signals to the moving GPS unit attached on the UGV.

The UGV is, beside of the magnetometer and GPS, equipped with a computer unit, controlling the vehicle. This is also the unit that handles the wireless communication with the MCC. This wireless link could for instance be Universal Mobile Telecommunications System (UMTS) or Wireless Fidelity (WiFi), but for this project WiFi is suggested.

2.1 Provided Hardware

For this project all hardware is provided beforehand. In the following sections the provided hardware is described component by component in terms of functionality, precision and interface.

2.1.1 Unmanned Ground Vehicle

The provided Unmanned Ground Vehicle (UGV) is a RobuROC4. This robot is build for outdoor use and has 4 independent wheels mounted. They are powered by 4 brushless DC motors working together two and two on the left and right wheel pairs, making the robot able to turn on the spot. The top speed is rated to 2 m/s and its weight is approximately 140 kg and up to 100 kg extra payload can be added. The measured dimensions of the RobuROC4 can be found in table 2.1

Dimension	Measure [mm]
Length	1380
Height	830
Width	812
Track Width	685
Tire Width	105

Table 2.1. RobuROC4 dimensions.

The RobuROC4 has an embedded controller, called RobuBOX, running Windows XPe. The robot is equipped with odometer, proxymeter, and a bumber sensor. The robot has a wireless emergency stop and can be driven in two modes, by a Xbox360 wireless gamepad or robuBOX commands. The commands can be sent by a UDP connection to the robot via Ethernet, and can be used for setting angular and linear velocity, and requesting data from sensors in the RobuROC4. The sensor data is returned from the robot at a specified rate. The sensor data which the RobuROC4 is capable of returning is specified

in the RobuROC4 communication manual . Among others it is capable of returning its current velocity (both angular and linear), but also more static information such as maximum velocities and its physical dimensions.

2.1.2 GPS

The provided GPS module is of type and model Ashtech MB-100. The MB-100 supports output of several types of the National Marine Electronics Association (NMEA)0183 standard sentences. The list of these are found in the datasheet for the MB-100 . The sentence output rate for the MB-100 is adjustable up to 20 Hz. The connection to the MB-100 is through USB or RS232 serial connection. In an RTK setup, the precision of the longitude and latitude coordinates is according to the datasheet 1 cm + 1 ppm of the distance from the moving GPS antenna module to the base station antenna in meters.

2.1.3 Magnetometer

The provided magnetometer is of type Ocean Server - OS5000-US. The OS5000-US supports output in several different types of the NMEA0183 sentence standard. The list of output sentences is found in the OS5000-US datasheet . The sentence output rate is adjustable from 0.01 Hz up to 40 Hz. The communication with the OS5000-US is through USB or RS232 connection. According to the datasheet the output heading of the magnetometer has a precision of 0.5 ° when the magnetometer is mounted properly.

2.1.4 Control Laptop

The control laptop mounted on the RobuROC4 is a Toshiba TECRA M11-134 with a Intel® Core™ i3-350M-processor. The used operative system is a 32-bit Ubuntu 10.04 LTS. The specifications for this laptop are listed in table 2.2.

Hardware	**Performance specs.**
Processor	2.26 GHz, 2 cores (hyperthreaded), 4 threads
Physical memory	8 GB DDR3 SDRAM, 1066 MHz
WiFi	IEEE 802.11(a,b,g,n)
USB ports	2

Table 2.2. Hardware specifications for the control laptop.

2.1.5 Composition of the Provided Hardware

The RobuROC4 has the GPS and magnetometer mounted on a plastic tube, which is mounted in the back left corner of the vehicle. The antennas for RTK radio communication and WiFi are mounted on the plastic tube as well. The placement of the antenna is illustrated in figure 2.2 and shown on the image in figure 2.3. The dimensions of the

RobuROC4 shown in figure 2.2 is without the safety bumper (see figure 2.3) and do therefore not comply with the dimensions in table 2.1.

The laptop is fastened under the robots' lid. The laptop is connected to the magnetometer and GPS via USB to serial converters and thereby using the serial interface of the two sensors. The laptop is connected to the RobuROC4 via ethernet. The overall system is assembled as described above and will from this point on be referred to as the Autonomous Line Tracking Agricultural Robot (ALTAR)

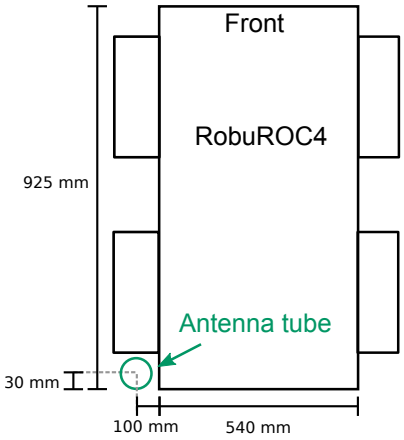


Figure 2.2. Placement of the plastic tube.

Figure 2.3. Image of composition of the provided hardware.

3 Requirement Specification

This worksheet specifies the requirements set for this project. The first part describes some overall requirements followed by a delimitation of these. Afterwards a set of specific requirements to fulfil during this project are stated.

3.1 General Requirements

The considerations presented here are based on an ideal view upon the desired functionality of the RobuROC4. The following considerations does however still provide an underlying basis for developing of the specific requirements.

The ALTAR has, as mentioned in worksheet 2, to work in a field where every single crop is of value to the farmer. Therefore it is vital that damage done to the crops should be minimised. This implies that the ALTAR has to be able to drive around in a beet field without driving over ridges with beets.

During this project the ALTAR communicate with the MCC via a WiFi channel. A protocol for this communication must therefore be designed such that it is possible to pass both information to, and receive data from the ALTAR. At a later stage the WiFi could be exchanged with a different wireless connection with longer range and less bandwidth. Hence the protocol should not depend on WiFi and should be designed in a way that limits needed bandwidth of the connection.

The locations where the ALTAR has to drive to will be determined by where the crops need fertiliser and herbicides. These locations can be any possible location within the field. The robot should therefore be able to drive to any point in the field independently of its starting point.

In order to drive as efficiently as possible, the path to every waypoint should be kept as short as possible and be driven as fast as possible. This does not only make the robot reach its target faster, it also makes sure no more harm than necessary is done to the field by choosing an unnecessarily long route. Driving as fast as possible makes sure that as much work as possible is performed within a given time frame. This however implies some considerations in terms of battery time, such that the speed and acceleration chosen optimises the battery usage.

Since the autonomous system implemented on the RobuROC4 is supposed to be a good alternative to existing solutions of early treatment of beets, the demand of influence by human hand should therefore be kept as low as possible. For instance the robot should try to avoid getting stuck and in case of low battery level or errors in the systems, it should be able to return to its working station and wait for help.

3.2 Focus of the project

During this project, the focus will be upon implementing software that allows the ALTAR to autonomously drive from one waypoint to another without damaging the crops or the field in general. The following section briefly describes which parts of the functionalities described in the previous section that is to be implemented.

The route planning itself is expected to be performed externally on the MCC. Hence the main task for the robot can be reduced to follow a straight line between two waypoints.

To simplify the system initially the ALTAR is limited to only driving forward and turn at the end of a row if it needs to go the other way.

Since the robot is to be controlled externally from a MCC, a protocol has to be designed to allow exchange of information. During this project this protocol will be designed and implemented on the ALTAR.

Although error handling is an important aspect of an autonomous systems, this will be given a low priority through the implementation.

3.3 Specific Requirements

The following specific requirements are divided in to two categories. One states the desired functionality of the ALTAR, while the other states the requirements for the communication between the ALTAR and the MCC.

3.3.1 ALTAR

The ALTAR must be able to:

r.1 **Follow a line between two waypoints with a maximum deviation of** $\pm 55\,\text{mm}$ **orthogonal from the line segment.**

Assuming that the ridges of the beets are 100 mm wide, and the row spacing as mentioned in 2 is 500 mm, it can be seen from figure 3.1 that the gap between the wheel of the

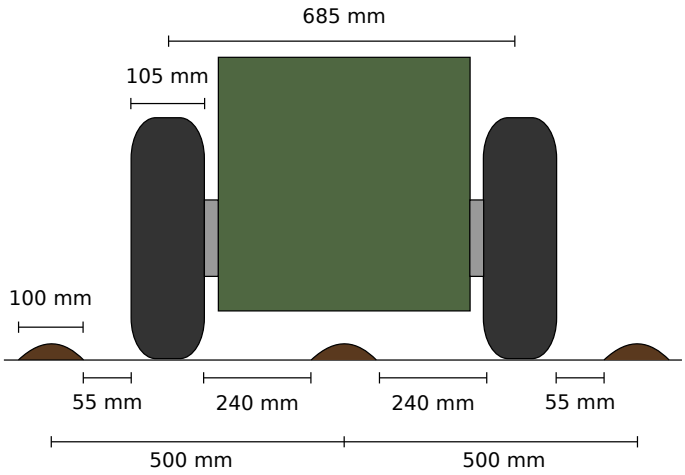


Figure 3.1. Measurement on gap between RobuROC4 wheel and ridges.

RobuROC4 and the ridge will be 55 mm, why this value is chosen.

r.2 Drive to a waypoint and stop within 55 mm of it.

Since the robot must not only be able to follow a line, but also stop when it reaches its destination, r.2 is set to specify the minimum precision.

r.3 The wheel sets must at no time turn opposite of each other.

This requirement is stated for two reasons. One is that the robot should do as little damage to the field as possible while driving. Due to the skid steering, tight cornering will make the wheel sets turn opposite of each other and cause unnecessary damage to the field. The second is that allowing the two wheel sets to drive opposite of each other with exactly the same velocity will cause the RobuROC4 to rotate around itself on the spot. In muddy conditions this is expected to cause the robot digging into the ground and get stuck.

3.3.2 Communication

These requirements specifies which kind of communication should be possible between the ALTAR and the MCC. This implies that both software, hardware and protocol should support these requirements.

The protocol must allow the following functionalities of the communication between ALTAR and MCC:

c.1 Send and remove line segments in form of GPS coordinates from the MCC to the ALTAR using WiFi.

This requirement is set, as the robot must be able to drive autonomously in the field, following a route of waypoints determined by the MCC.

c.2 Return data from sensors both internally and externally mounted on the ALTAR.

While driving, certain sensor readings from the ALTAR can be important to the MCC. These readings include battery level and current position. For debugging and error handling purposes other sensor readings could be of interest from the MCC, hence all available sensor data should be supported both by the protocol and the ALTAR.

c.3 Setup the ALTAR to periodically return sensor data.

This requirement is stated to reduce the load on the network. This requirement reduces the need of requests made from the MCC since it can permanently subscribe to data from a specific sensor with a certain frequency.

4 Modelling of RobuROC4

This worksheet describes how a model of the RobuROC4 is determined. The goal of this model is to get a representation of the vehicle's behaviour when different inputs are applied to it. A Simulink model will be used to test different controllers and planning algorithms to select the best fit.

4.1 Kinematic Model of RobuROC4

A kinematic model for the RobuROC4 is defined to identify the relation between velocities on the wheel pairs and the resulting linear and angular velocity of the vehicle. The derivation of the kinematic model is inspired by [6]. The derived model takes point of origin in figure 4.1 illustrating the vehicle.

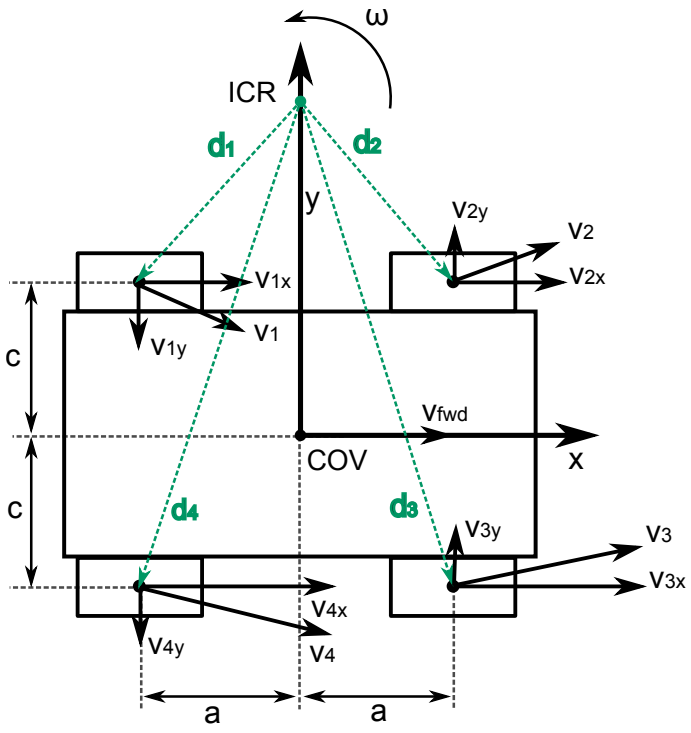


Figure 4.1. Wheel velocities of the RobuROC4.

As illustrated in the figure, the Centre Of Vehicle (COV) is defined as the point with equal distance to all wheels. In the figure the local coordinate system of the vehicle is placed with origo in the COV. The x-axis' positive direction is pointing in the forward direction of the vehicle, and the y-axis' positive direction points out to the left side of the vehicle.

In the derivation of the kinematic model, the longitudinal slippage on the wheels is neglected, and the following relation is therefore considered valid:

$$v_{\mathrm{ix}} = r_{\mathrm{i}}\omega_{\mathrm{i}} \tag{4.1}$$

Where v_{ix} is defined to be the longitudinal velocity of the i'th wheel depicted in figure 4.1, ω_{i} is the angular velocity of the i'th wheel and r_{i} is the radius of the i'th wheel.

It is assumed that the angular velocity of the left wheels are the same, and the angular velocity of the right wheels are the same. Having the same radius on all wheels, the following relation is derived:

$$v_{\mathrm{L}} = v_{1\mathrm{x}} = v_{2\mathrm{x}} \tag{4.2}$$

$$v_{\mathrm{R}} = v_{3\mathrm{x}} = v_{4\mathrm{x}} \tag{4.3}$$

Based on the assumption of equal velocity on the wheels of the two wheel pairs, the Instantaneous Centre of Rotation (ICR) must be on the y-axis of the local coordinate system, hence the coordinate of the ICR is $(0, ICR_{\mathrm{y}})$. In figure 4.1 d_{i} is the vector from the ICR to the i'th wheel. This is comprised of two components d_{ix} and d_{iy}. ω is the instantaneous angular velocity of every point in the vehicle around the ICR. v_{fwd} is the resulting linear velocity of the COV in the x-direction of the vehicle.

From figure 4.1 the following equation describing the angular velocity ω of the vehicle is derived:

$$\frac{v_{\mathrm{ix}}}{-d_{\mathrm{iy}}} = \frac{v_{\mathrm{fwd}}}{ICR_{\mathrm{y}}} = \omega \tag{4.4}$$

A relationship between the y-component of the d_{i} vectors and the y-coordinate of the ICR is also derived from the figure:

$$d_{1\mathrm{y}} = d_{2\mathrm{y}} = -ICR_{\mathrm{y}} + c \tag{4.5}$$

$$d_{3\mathrm{y}} = d_{4\mathrm{y}} = -ICR_{\mathrm{y}} - c \tag{4.6}$$

Using equation 4.2, 4.4, and 4.5 equation 4.7 for the velocity on the left wheels can be derived. Likewise by using equation 4.3, 4.4, and 4.6 equation 4.8 for the velocity on the right wheel pair can be derived.

$$\begin{aligned} v_{\mathrm{L}} = v_{1\mathrm{x}} &= -d_{1\mathrm{y}} \cdot \omega = -(-ICR_{\mathrm{y}} + c) \cdot \omega = ICR_{\mathrm{y}} \cdot \omega - c \cdot \omega = \frac{v_{\mathrm{fwd}}}{\omega} \cdot \omega - c \cdot \omega \\ &= v_{\mathrm{fwd}} - c \cdot \omega \end{aligned} \tag{4.7}$$

$$v_{\mathrm{R}} = v_{\mathrm{fwd}} + c \cdot \omega \tag{4.8}$$

Combining equation 4.7 and 4.8 yields relation 4.9 between the longitudinal linear velocity v_{fwd} and the wheel velocities, and relation 4.10 between angular velocity ω and the wheel velocities.

$$v_{\mathrm{fwd}} = \frac{v_{\mathrm{R}} + v_{\mathrm{L}}}{2} \tag{4.9}$$

$$\omega = \frac{v_{\mathrm{R}} - v_{\mathrm{L}}}{2c} \tag{4.10}$$

As described earlier these equations are only valid when no longitudinal skid occurs. This is however only the case when the angular velocity ω is close to zero. By using these equations directly to model the behaviour of the vehicle, the correctness of the model depends on the angular velocity ω. To create a more correct model, a more thorough analysis of the behaviour when skidding must be carried out.

4.1.1 Model Test

This model test is a check up on the conversion from the inputs v_{fwd} and ω to wheel velocities v_{R} and v_{L} carried out by the internal controller in the RobuROC4. More precisely for the case where either the left or right wheel pair has zero velocity.

A simple test is carried out by applying a combination of v_{fwd} and ω to the vehicle that must result in either $v_{\text{R}} = 0$ or $v_{\text{L}} = 0$, this combination is found by using equation 4.7, 4.8, 4.9 and 4.10. The relationship between v_{fwd} and ω when $v_{\text{R}} = 0$ can be calculated from 4.7 and 4.10 as:

$$
\begin{aligned}
v_{\text{L}} &= v_{\text{fwd}} - \omega c \\
\omega &= \frac{0 - v_{\text{L}}}{2c} \\
&\Downarrow \\
\omega &= \frac{-v_{\text{fwd}}}{c}
\end{aligned}
$$

In table 4.1 the applied velocities and the results are listed. In the calculations c is set to 0.34. This value is determined by polling the distance between the wheels directly from the RobuROC4 as described in section 2.1.1.

$v_{\text{fwd}}[m/s]$	$\omega[rad/s]$	**Outcome**
0.25	−0.74	$v_{\text{R}} = 0$
0.25	0.74	$v_{\text{L}} = 0$
0.50	−1.47	$v_{\text{R}} = 0$
0.50	1.47	$v_{\text{L}} = 0$
1.00	−2.94	$v_{\text{R}} = 0$
1.00	2.94	$v_{\text{L}} = 0$
1.50	−4.41	$v_{\text{R}} = 0$
1.50	4.41	$v_{\text{L}} = 0$
2.00	−5.88	$v_{\text{R}} > 0$
2.00	5.88	$v_{\text{L}} > 0$

Table 4.1. Results of the kinematic model verification.

As seen in the table, the derived model meets the expected outcome in all cases except for the last two. In the last two tests the velocity of the wheels is expected to be zero but is determined larger than zero. The most likely reason for this is that the input angular velocity is above the maximum angular velocity of 5 rad/s. This maximum linear velocity is extracted directly from the RobuROC4 by polling for this value.

Calculating the wheel velocities for the two last situations shown in table 4.1 and using the angular velocity saturated to $\pm 5\,\mathrm{rad/s}$ instead of $\pm 5.88\,\mathrm{rad/s}$, results in a wheel velocity of $0.3\,\mathrm{m/s}$ instead of $0\,\mathrm{m/s}$, which complies with the observed situation.

This model test shows that the derived kinematic model is valid when one wheel pair has zero velocity and the input to the model is within the limitations of the vehicle, however it is also considered valid for use when both wheels have velocities different from zero.

4.2 Dynamics of the RobuROC4

In order to make the model complete, the dynamics are modelled. To do this an input reference is applied to the linear velocity input and the angular velocity input separately while the other is set to zero. This input reference is a step of the maximum angular and linear velocity. These values are extracted from the vehicles communication interface as described in section 2.1.1 and are given as $5\,\mathrm{rad/s}$ and $2\,\mathrm{m/s}$. The measurement journal for this test can be found in appendix A.

In figure 4.2 the output of the linear velocity test is shown. The data is split up into two runs since the room in which the test was carried out, was not big enough for the test to be carried out in one run. In figure 4.3 the output of the angular velocity test is shown.

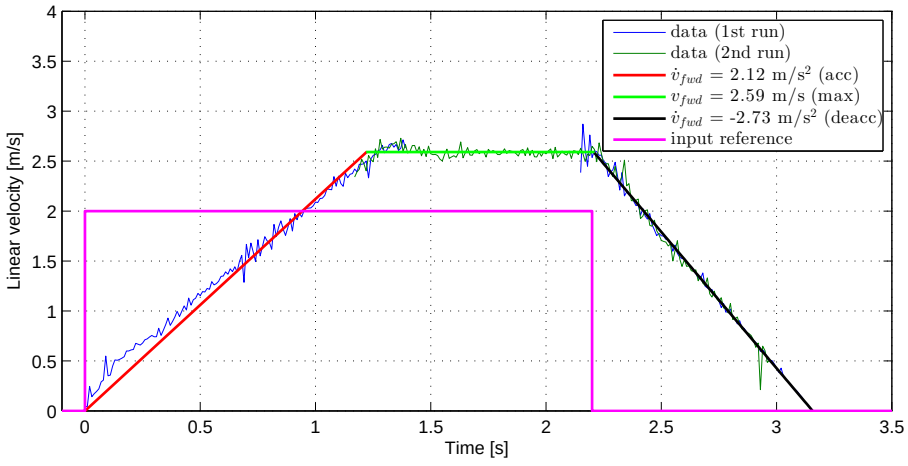


Figure 4.2. Linear velocity of RobuROC4 in m/s.

From these figures it is seen that the internal controller in the RobuROC4 secures that the desired linear and angular velocity inputs are reached with constant acceleration. Since the accelerations are constant, first order approximates have been made to extract values of the dynamic behaviour. Table 4.2 shows the determined values.
As mentioned in section 2.1.1 the RobuROC4 provides the possibility of transmitting data containing its current forward and angular velocity. These velocities are determined from

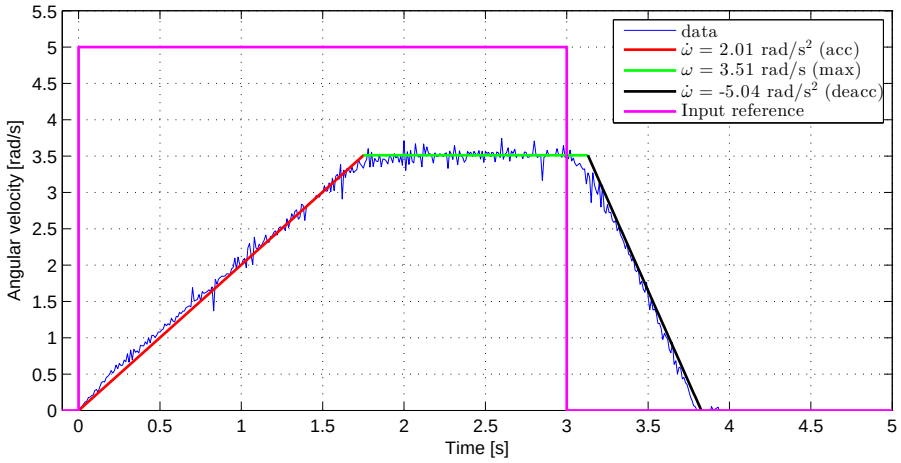


Figure 4.3. Angular velocity of RobuROC4 in rad/s.

	Acceleration	Deceleration	Maximum velocity
test v_{fwd}	2.12 m/s^2	2.73 m/s^2	2.59 m/s
test ω	2.01 rad/s^2	5.04 rad/s^2	3.51 rad/s

Table 4.2. Table of the measuring results for RobuROC4.

measurements performed directly on the wheels and shows a consistent error compared to the measurements obtained by the motion tracking. The motion tracking detected a velocity of a factor 1.30 larger than the RobuROC4 determined. This seems to be due to the fact that the vehicle has been ordered with larger wheels than standard. This explains why there is a constant error on the velocity data output from the RobuROC4. Furthermore the angular velocity applied was not reached, this is assumed to be due to skid when turning.

4.3 Wheel Velocity Limitations

The following two tests are carried out to determine how the built-in controllers respond to different combinations of forward linear velocity input and angular velocity input. Especially interesting is the case where the combination of forward and angular velocity is predicted to exceed the maximum determined wheel velocities. The measured velocities in this test are not performed by the motion tracking system, instead the internal resolver is used. To eliminate the inconsistency caused by larger mounted wheels than standard, the velocities measured in this test are multiplied with a constant factor $k = 1.3$ found in section 4.2. The reason why both angular velocity and linear velocity are multiplied by the same factor k, is seen from the kinematic model equations 4.9 and 4.10, where both right and left wheel velocities are multiplied by the factor k. In the equations below the prime indicates velocities of a vehicle with the larger wheels mounted.

$$v'_{\text{fwd}} = \frac{v_{\text{R}} \cdot k + v_{\text{L}} \cdot k}{2} = \frac{v_{\text{R}} + v_{\text{L}}}{2} \cdot k = v_{\text{fwd}} \cdot k$$
$$\omega' = \frac{v_{\text{R}} \cdot k - v_{\text{L}} \cdot k}{2c} = \frac{v_{\text{R}} - v_{\text{L}}}{2c} \cdot k = \omega \cdot k$$

During this test the RobuROC4 was lifted from the ground with its wheels spinning freely.

Figure 4.4 shows how the RobuROC4 responds to a gradually increasing angular velocity with a step of 0.65 rad/s and the forward velocity input constant at 1.3 m/s. Figure 4.5 shows the same test performed with a forward velocity constant at 2.6 m/s.

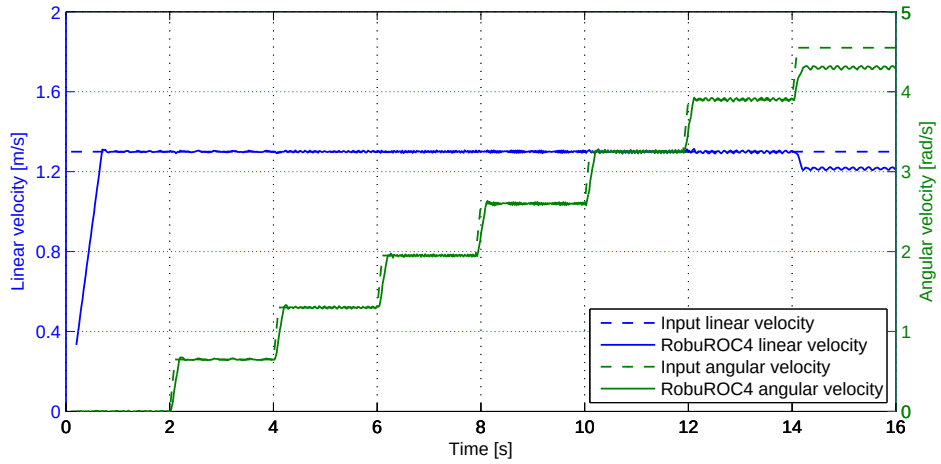


Figure 4.4. Velocity inputs and actual achieved velocities of the RobuROC4 when input is constant linear velocity at 1.3 m/s and angular velocity stepping with 0.65 rad/s.

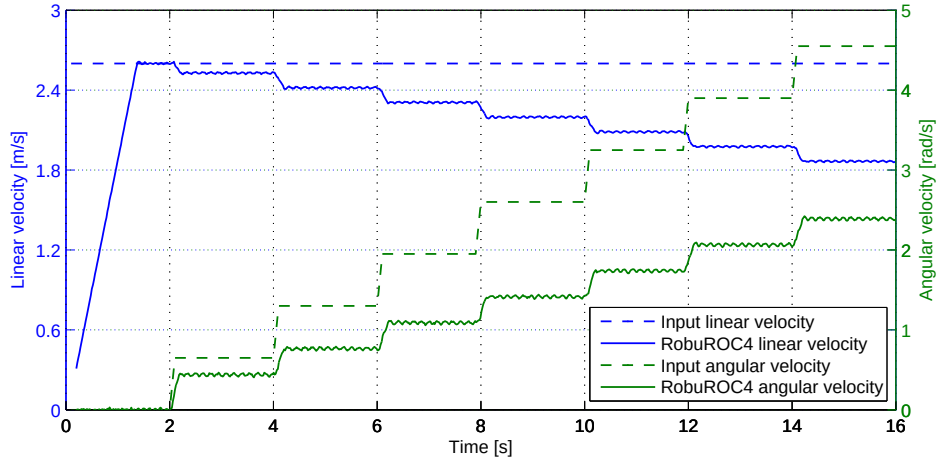


Figure 4.5. Velocity inputs and actual achieved velocities of the RobuROC4 when input is constant linear velocity at 2.6 m/s and angular velocity stepping with 0.65 rad/s.

From figure 4.4 it is seen that the vehicle is able to maintain a constant linear velocity of 1.3 m/s except from the last step. At this point both the angular velocity and the linear

velocity are saturated because the velocity of one of the wheel pairs is saturated. This is seen by calculating the theoretical wheel velocities by using the kinematic model equations 4.7 and 4.8.

$$v_{\mathrm{R}} = v_{\mathrm{fwd}} + \omega c = 1.3 + 4.55 \cdot 0.34 = 2.847\,\mathrm{m/s}$$
$$v_{\mathrm{L}} = v_{\mathrm{fwd}} - \omega c = 1.3 - 4.55 \cdot 0.34 = -0.247\,\mathrm{m/s}$$

Because of maximum wheel velocity, v_{R} is saturated to 2.6 m/s. Using the saturated $v_{\mathrm{R}} = 2.6\,\mathrm{m/s}$, and the theoretical calculated $v_{\mathrm{L}} = -0.247\,\mathrm{m/s}$, the resulting v_{fwd} and ω is calculated from the kinematic model equations 4.9 and 4.10.

$$v_{\mathrm{fwd}} = \frac{v_{\mathrm{R}} + v_{\mathrm{L}}}{2} = \frac{2.6 - 0.247}{2} = 1.2\,\mathrm{m/s}$$
$$\omega = \frac{v_{\mathrm{R}} - v_{\mathrm{L}}}{2c} = \frac{2.6 - (-0.247)}{2 \cdot 0.34} = 4.2\,\mathrm{rad/s}$$

This results in a reduction of both linear velocity and angular velocity, and as seen in figure 4.4 the theoretical calculated velocities agree with the measured velocities. Due to the relatively low linear velocity in the first test, the angular velocity must be high before saturation appears. Figure 4.5 shows how the saturation immediately become visible in the second test. This is caused by the constant high linear velocity, and therefore a small increment in the angular velocity causes one wheel set to be saturated to the maximum.

From these tests it is concluded that if a combination of linear and angular velocities are applied to the RobuROC4, causing one wheel pair to reach maximum velocity, then this wheel pair will be saturated and the other wheel pair velocity will be left unchanged.

4.4 Resulting model of the RobuROC4

The model of the RobuROC4 is developed from the maximum velocities and accelerations found in section 4.2 and the kinematic model found in section 4.1. Table 4.2 shows the results from the measurements on the RobuROC4. Figure 4.2 shows the dynamics in forward direction only and figure 4.3 shows the rotational dynamics of the vehicle.

Unlike most physical systems which can be approximated by a first or second order transfer function, the measurements show that the RobuROC4 has constant acceleration, independent of the input as long as it is within the limits of the angular and linear velocities. The reason for this behaviour is the internal controllers implemented in the RobuROC4 that makes sure a given input is reached with a constant acceleration.

To mimic the behaviour of the RobuROC4 while designing path planning and controllers, a Simulink model has been made as shown in figure 4.6. This model uses equation 4.7, 4.8, 4.9, and 4.10 to implement constant acceleration using a feedback loop. The two acceleration saturations limit the acceleration and deceleration to $2.12\,\mathrm{m/s^2}$ and $2.73\,\mathrm{m/s^2}$ respectively. The velocity limitations make sure none of the wheels exceed the measured limit of 2.59 m/s in both directions.

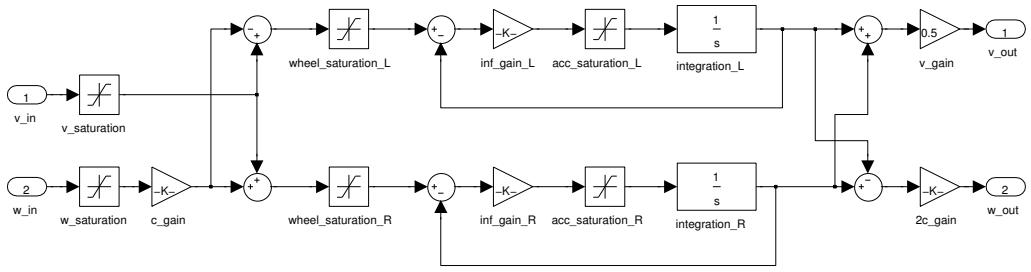


Figure 4.6. Simulink model of the RobuROC4.

Using only data from the forward velocity results in a model which will be most accurate while driving straight, but inaccurate when rotating on the spot. The reason for using these values is that, as specified in requirement r.3, the wheels must never turn opposite of each other during driving. Hence the forward dynamics are considered best fitting scenario for driving.

4.5 Model validation

To validate the model shown in figure 4.6, a test is performed to compare model behaviour with the behaviour of the real RobuROC4. This is done by applying the same v_{fwd} and ω input to both the Simulink model and the RobuROC4 and comparing the output from each of them. The inputs are generated by providing waypoints to an early model of the controller concept, using the model described in section 4.4. The inputs are saved to a file `/MATLAB_files/model_validation/feed_forward_inputs.csv`. The positions are recorded in three different ways. An estimate by the RobuROC4 which is found by polling for the RobuROC4's own estimated position, a position recorded by the GPS and a position recorded from the Simulink model used to generate the inputs. During the test, the maximum linear velocity and angular velocity was reduced to 50 % and 25 % respectively. This was done to reduce the influence of slip while rotating and give operators more time to react in case of any unwanted behaviour.

Figure 4.7 shows the positions recorded during the test. It should be noted that most of the diversion between estimate from the RobuROC4 and the Simulink model can be explained by the larger wheels. The same phenomenon is observed in section 4.2. From the GPS measurement it is seen that the vehicle starts to turn at the expected distance from the starting point. This indicates that the linear velocity behaves as predicted. The angular velocity on the other hand seems to be less than predicted in the model. This results in a path, measured by the GPS, which diverts approximately 30 ° from the path predicted by the model.

In figure 4.8 and 4.9 the effect of the larger wheels has been accounted for by multiplying the model estimate with a factor of 1.3 found in section 4.2 to ease comparison. Figure 4.8 shows the recorded linear velocities during the test. The GPS measured linear velocity is logged directly from the GPS without any modification. The linear velocity of the

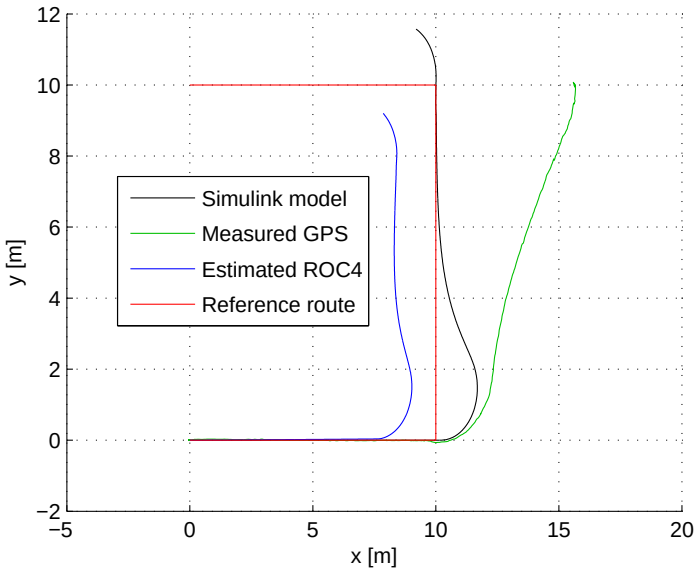


Figure 4.7. Plot of the recorded positions during the test.

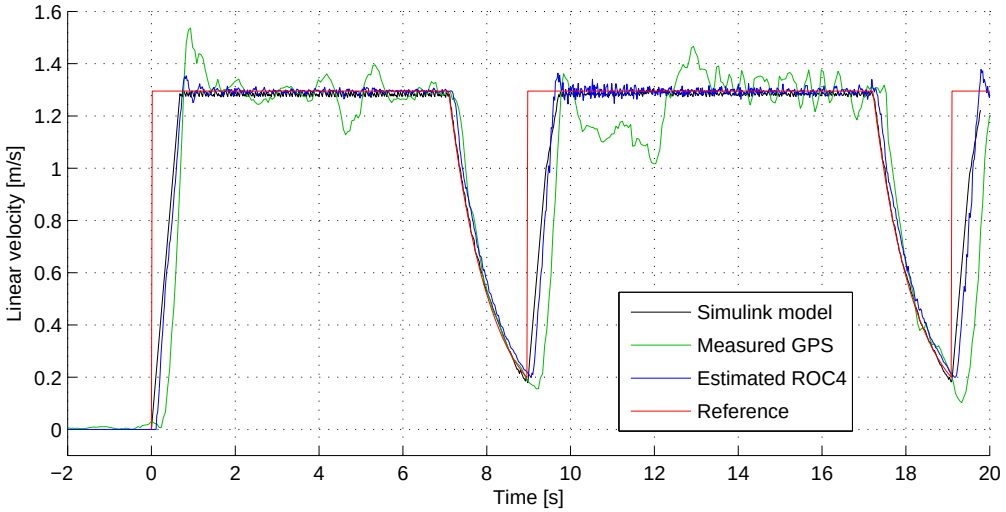


Figure 4.8. Plot of the recorded linear velocities during the test.

Simulink model and estimated by RobuROC4 follow each other closely. The GPS linear velocity diverts from the other two, especially while driving at constant velocity. The shape of the velocity data indicates the influence by some kind of disturbance. The pole at which the GPS is mounted, is as described in section 2.1.5 a plastic tube mounted loosely to the RobuROC4 frame by cable ties and tape. This is considered a more likely explanation to the fluctuation than an unstable linear velocity of the vehicle when it drives straight. It is concluded that the deduced model behaves nearly like real vehicle in terms of linear velocity.

Figure 4.9 shows the angular velocities during the test. It should be noted that due to problems with the magnetometer, it was not possible to compare angular velocities from this with the desired. First of all it can be concluded that the Simulink model behaves almost in the same way as expected by the RobuROC4. Figure 4.7 however shows that

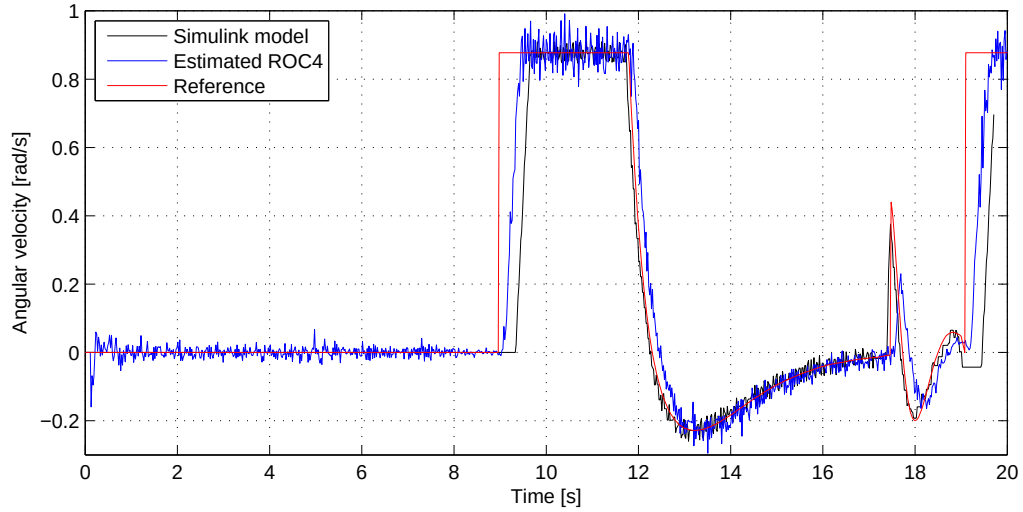


Figure 4.9. Plot of the recorded angular velocities during the test.

the real angular velocity is less than predicted, since the vehicle does not turn as much as expected.

This is expected since our model does not take slip of the wheel pairs into consideration. Equation 4.9 and 4.10 are derived from a case where all rotation of the left and right wheel pairs is translated into forward velocity and angular velocity. For a four wheeled skid steering vehicle this will never be the case while turning. To deduce a more accurate model of the rotation, this slip must be taken into consideration.

Studying the first transition of the reference angular velocity from 0 rad/s to 0.88 rad/s, the model estimate seems to be delayed by approximately 0.4 s compared to the RobuROC4 estimate. This behaviour is caused by the fact that both wheel pairs are set to accelerate with the same constant acceleration. If e.g. the vehicle is desired to start driving forward and turning from a stand still, the Simulink model will increase both wheel velocities equally fast until one of the wheels reaches its target velocity. This results in no rotation until one of the wheels achieves its desired velocity and different wheel velocities are achieved.

Since the RobuROC4 estimate of rotation reacts almost instantaneously, it can be concluded that the internal controllers also take rotation into consideration in terms of accelerating the wheels.

Although the Simulink model shows flaws, especially in terms of angular velocity, it is still considered good enough to predict behaviour of the vehicle to design controllers from.

5 Planning and Control

The following worksheet treats the topic of planning a path for the ALTAR and designing a controller which enables it to follow this path. The planning of the path has to take the physical limitations of the RobuROC4 into consideration, while the controllers must be designed from the dynamics of the RobuROC4. The planning and control algorithm is presented and simulated to inspect its behaviour.

The controller implemented on the ALTAR has the overall objective of enabling it to drive a route specified by waypoints provided by the MCC. As stated in the requirement specification in worksheet 3 the objective is to reach each waypoint as fast as possible. This however has to be accomplished while keeping within the maximum orthogonal distance of 55 mm as stated in requirement r.1 and being able to stop within 55 mm of the end point as stated in requirement r.2. To minimise the damage done to the field and the risk of getting stuck, requirement r.3 specifies that the the wheel pairs must not turn opposite of each other.

To achieve this functionality a control principle as seen in figure 5.1 is used. The blue arrows denotes reference inputs to the controller. The overall control algorithm takes in a line segment as reference to control the ALTAR.

If the the robot drives in a straight line between current position and target waypoint as seen in figure 5.2 it would cause the robot to run over unnecessarily many beets. Instead a reference to the controller is calculated using a concept that from this point on will be referred to as helmsman control. The angle reference is implemented with the concept of "good helmsman" defined in [7]. This implies that the vehicle seeks onto the straight line as seen in figure 5.3 which results in a path that reduces the amount of beets that are run over.

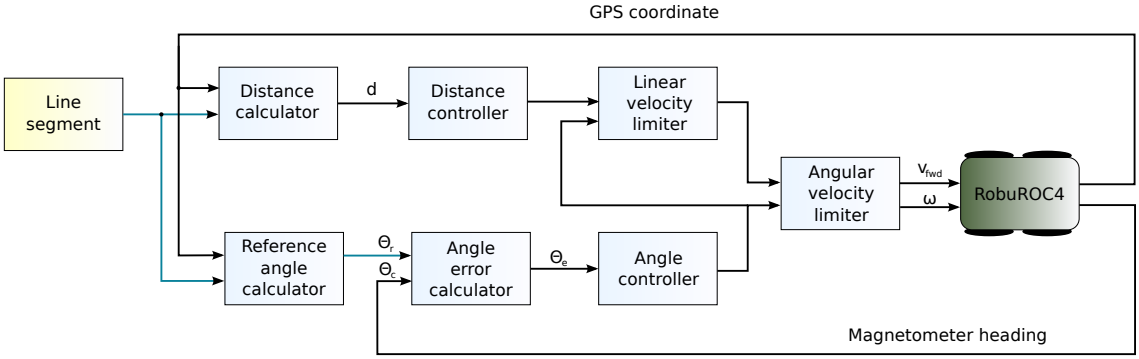


Figure 5.1. Overview of the control principle. Line segment is received and compared to the GPS position and magnetometer angle by calculating aiming point, angle error and distance to endpoint. The error is minimised, and the distance is reduced by using two controllers.

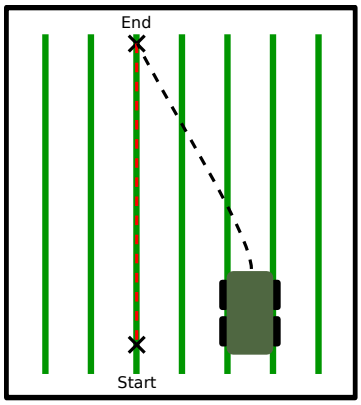


Figure 5.2. Figure of direct steering to end waypoint.

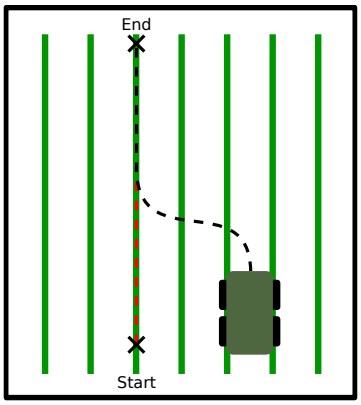


Figure 5.3. Figure of helmsman control principle.

In the following sections each block of the controller principle will be described in terms of functionality. In section 5.4 the task of choosing when to leave current line segment and follow the next one is discussed.

5.1 Reference Calculation

This section describes how the the reference angle is generated in the *Reference angle calculator* to follow a path as shown in figure 5.3. The section also introduces how the angle error and distance to end point are calculated in the *Distance calculator* and *Angle error calculator* respectively.

Reference angle calculator

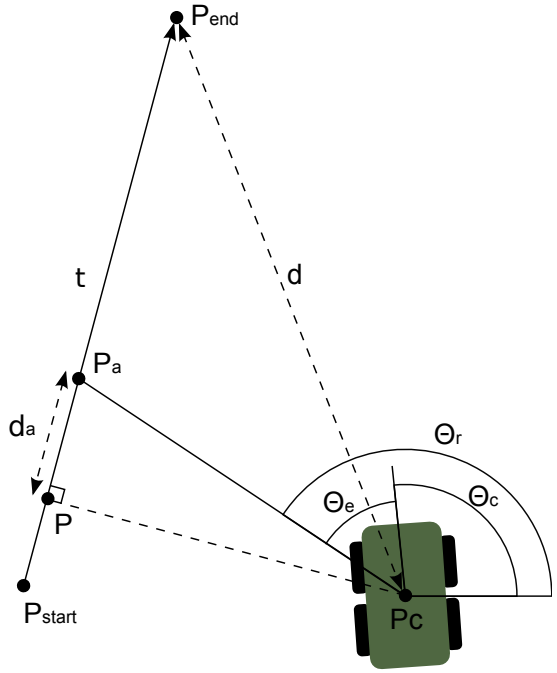
The reference angle is calculated by using helmsman control principle. The idea behind this principle is to generate an aiming point P_a on the line segment and aim at this point instead of aiming at the endpoint P_end, see figure 5.4.
The first step of calculating the aiming point is to project the current position of the ALTAR P_c orthogonally onto the line segment described by its vector t. This is done using orthogonal projection of point on vector in the two dimensional vector space.

$$t = P_\mathrm{end} - P_\mathrm{start}$$
$$P = P_\mathrm{start} + \left(\frac{(P_\mathrm{c} - P_\mathrm{start})^T t}{t^T t} \right) t$$

The aiming point P_a is then generated by aiming a distance d_a ahead of the projection point P towards the endpoint P_end:

$$P_\mathrm{a} = P + d_\mathrm{a} \cdot \frac{t}{||t||_2} \tag{5.1}$$

The distance d_a is chosen to be defined in two different ways depending on the Euclidean distance from the projection point P to the end point P_end. If this distance $||P - P_\mathrm{end}||_2$ is less than a certain threshold l, the distance is chosen to be a constant fraction $\frac{1}{c_\mathrm{h}}$ of the



θ_r Reference angle
θ_c Current angle
θ_e Error angle
P_{start} Last waypoint
P_{end} Target waypoint
P_a Aiming point on line segment
P_c Current position
t Line segment
P Projection point
d Distance from vehicle to target
d_a Distance between aiming point and projection point

Figure 5.4. Overview of helmsman control principle.

distance $||P - P_{end}||_2$. Choosing the distance d_a in this way ensures that the aiming point approaches the end point of the line segment as the distance $||P - P_{end}||_2$ decreases.

A disadvantage of choosing d_a as a fraction of the distance $||P - P_{end}||_2$ is that d_a will get large when the distance is large which creates a path like 5.2. To avoid this d_a is chosen to be a constant value k when the distance $||P - P_{end}||_2$ is greater a certain threshold l. This results in the following way of choosing d_a:

$$d_a = \begin{cases} \frac{||P_{end} - P||}{c_h} & \text{for } ||P_{end} - P||_2 \leq l \\ k & \text{for } ||P_{end} - P||_2 > l \end{cases} \tag{5.2}$$

To achieve a smooth transition between the two ways of aiming, the aiming point should be exactly the same just before and after the transition. To ensure this, the distance d_a must be calculated to the same value on both sides of the threshold distance l. To find the relationship between the factors l, k, and c_h this happening, the two ways of calculating d_a are equated in the situation where $||P_{end} - P||_2 = l$:

$$k = \frac{||P_{end} - P||_2}{c_h} = \frac{l}{c_h} \tag{5.3}$$

Choosing the constant distance k is a compromise between overshoot if it is chosen too small and running over too many crops if it is chosen too large. Through Simulink simulation a distance of 2 m has been determined to be a good compromise.

In the same way the c_h factor has been determined to be 2 in order to hit the ending waypoint within 55 mm as specified in the requirement specification. By rewriting 5.3 the threshold distance l that ensures a smooth transition could be determined to be:

$$\begin{aligned} l &= c_h \cdot k \\ &= 2 \cdot 2 = 4\,\text{m} \end{aligned}$$

Distance calculator

The distance d in figure 5.4 is calculated in the *Distance calculator* as the Euclidean distance between the current position P_c of the ALTAR to the target waypoint P_{end}.

Angle error calculator

The error angle θ_e in figure 5.4 is calculated as the difference between the current heading of the ALTAR θ_c, and the reference angle θ_r.

5.2 Controller Design

The controllers chosen for the system are separate p-controllers. The p-controllers are used to minimise error angle and to reduce the distance to target point.

Distance controller

The gain for the forward velocity controller is calculated directly from the measured deceleration, since the ALTAR is supposed to drive as fast as possible until it reaches a waypoint. The linear velocity of a system with constant acceleration can be described as equation 5.4.

$$v_{fwd} = v_0 + at \tag{5.4}$$

Where v_0 is set to the highest possible initial velocity $2.59\,\mathrm{m/s}$, the known deceleration a is set to $-2.73\,\mathrm{m/s^2}$ and v_{fwd} is set to $0\,\mathrm{m/s}$. The time it takes to decelerate the RobuROC4 from $2.59\,\mathrm{m/s}$ to $0\,\mathrm{m/s}$ can be calculated to be $0.95\,\mathrm{s}$. During this time the distance travelled d can be calculated as:

$$d = v_0 t + \frac{1}{2}at^2 = 2.59 \cdot 0.95 - \frac{1}{2} \cdot 2.73 \cdot 0.95^2 = 1.23\,\mathrm{m}$$

Which is the minimum distance at which the ALTAR has to start decelerating before it reaches a waypoint. Since the controller is calculating the desired velocity from equation 5.5, the maximum gain for the velocity controller can be calculated as follows.

$$\begin{aligned} v_{fwd} &= v_{max} = 2.59\,\mathrm{m/s} \\ v_{fwd} &= d\ K_v \\ K_v &= 2.11 \end{aligned} \tag{5.5}$$

Since deceleration could be reduced on a muddy field where the wheels can loose grip, a K_v value of 1 is chosen instead. This causes the robot to start decelerate $2.59\,\mathrm{m}$ before the end waypoint is reached.

Angle controller

The design principle behind designing the gain value for the angular velocity uses the same considerations as the distance controller. For the ALTAR to minimise the error angle θ_e as

fast as possible, the angle gain should be as high as possible, on the other hand it should also be able to minimise the error angle without overshooting. In order to achieve this, the time the ALTAR takes to stop from full angular velocity is calculated as:

$$\begin{aligned}\omega &= \omega_0 + \alpha t \\ 0 &= 3.51 - 5.04t \\ t &= 0.7\,\text{s}\end{aligned}$$

The angle travelled is then:

$$\theta = \omega_0 t + \frac{1}{2}\alpha t^2 = 3.51 \cdot 0.7 - \frac{1}{2} \cdot 5.04 \cdot 0.7^2 = 1.22\,\text{rad}$$

The maximum gain can then be calculated:

$$\begin{aligned}\omega &= \theta_e K_\omega \\ K_\omega &= 2.87\end{aligned}$$

To leave some room for reduced deceleration a K_ω of 2 has been chosen for the controller, which implies that the maximum angular velocity is obtained when θ_e is greater than 1.76 rad.

5.3 Limitations of v_{fwd} and ω

Although the RobuROC4 is a Multiple Input Multiple Output (MIMO) system with in and outputs v_{fwd} and ω. In general the input v_{fwd} does not influence the output angular velocity, and the ω input does not influence the linear velocity output. However as seen in section 4.3 when the combination of v_{fwd} and ω reaches a certain limit, they will start to influence each other due to limitations of the wheel velocities. To reduce this problem and be able to treat the two inputs as two individual Single Input Single Output (SISO) systems, a decoupling is made in the *Linear velocity limiter* seen in figure 5.1. In addition the angular velocity is limited in the *Angular velocity limiter* to make sure that the wheel pairs do not turn opposite of each other as stated in requirement r.3.

Linear velocity limiter

From the model tests in 4.3 it is clear that both v_{fwd} and ω are affected when one or both of the wheels reaches its maximum velocity. It has been chosen to reduce the linear velocity since it is considered more important to be able to archive the desired angular velocity compared to the desired linear velocity. Reducing the linear velocity increases the time it takes to reach the target while a reduction in angular velocity reduces the manoeuvrability and therefore also the precision.

It is desired to reduce the linear velocity v_{fwd} exactly enough to reach the desired angular velocity. To do this equation 4.7 and 4.8 from section 4.1 are used. When the maximum wheel velocity on one wheel is known to be 2.59 m/s, the maximum value of v_{fwd} can be found as:

$$v_{\mathrm{L/R}} = v_{\mathrm{fwdMax}} \pm \omega c$$
$$\Downarrow$$
$$v_{\mathrm{fwdMax}} = 2.59 - |\omega c|$$

The highest possible ω can be achieved when v_{fwd} is half of maximum wheel velocity since this leaves maximum headroom for wheel velocity difference when none of the wheel pairs are allowed to turn opposite of the other.

The output velocity of *Linear velocity limiter* block is the minimum of the input from the *Distance controller* and the maximum allowed linear velocity v_{fwdMax} given by the desired angular velocity ω.

The resulting output of the *Limit linear velocity* block in figure 5.1 can be seen in figure 5.5. We see that the output linear velocity is gradually decreased as the angular velocity is increased until the output velocity reaches $v_{\mathrm{max}}/2$, where the it is saturated.

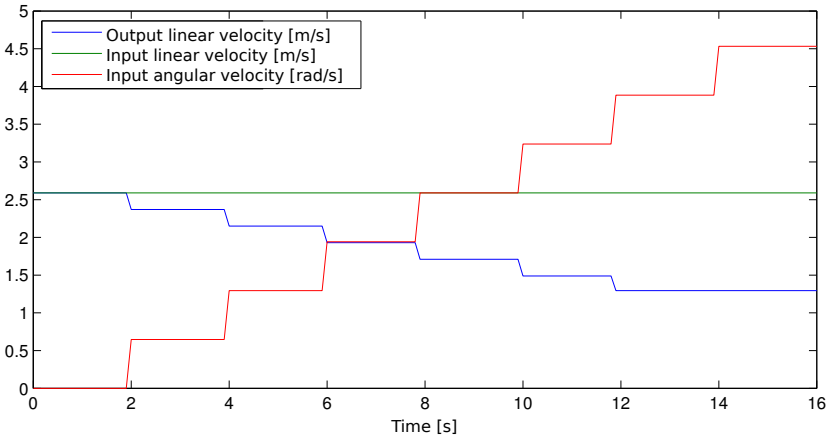


Figure 5.5. Output of the *Limit linear velocity* block where input forward velocity is kept constant and angular velocity is increased gradually.

Angular velocity limiter

Since requirement r.3 from the requirement specification specifies the wheel pairs never to spin opposite of each other, an angular velocity limiter is designed to prevent this. Equation 4.9 and 4.10 are used to deduce an expression for the case where this happens by setting $v_{\mathrm{L}} = v_{\mathrm{min}}$. Where v_{min} defines a minimum velocity of 0.2 m/s that the wheel has to keep in order to reduce the risk of getting stuck.

$$v_{\mathrm{fwd}} = \frac{v_{\mathrm{R}} + v_{\mathrm{min}}}{2}$$
$$\Downarrow$$
$$v_{\mathrm{R}} = 2v_{\mathrm{fwd}} - v_{\mathrm{min}}$$

This expression is inserted in the following equation, describing the angular velocity:

$$\omega = \frac{v_{\mathrm{R}} - v_{\mathrm{min}}}{2c}$$

Hereby the following relationship can be found:

$$c = \frac{v_{\mathrm{fwd}} - v_{\mathrm{min}}}{\omega}$$

Hence to remove any negative wheel velocities the relationship between $(v_{\mathrm{fwd}} - v_{\mathrm{min}})$ and ω has to be greater than c. If the relationship is less than c, ω is then reduced to the highest possible value which does not make any of the wheels turn backwards:

$$\omega = \frac{v_{\mathrm{fwd}} - v_{\mathrm{min}}}{c}$$

Figure 5.6 shows the output of the *Angular velocity limiter* block. During this test the input linear velocity is set to 0.5 m/s the first 7 seconds and then stepped to 1 m/s. Meanwhile the angular velocity is increased gradually and the resulting output angular velocity is observed. We see that the angular velocity is limited to approximately 0.9 rad/s when input velocity is set to 0.5 m/s and approximately 2.4 rad/s when input velocity is set to 1 m/s which is corresponds to the desired functionality.

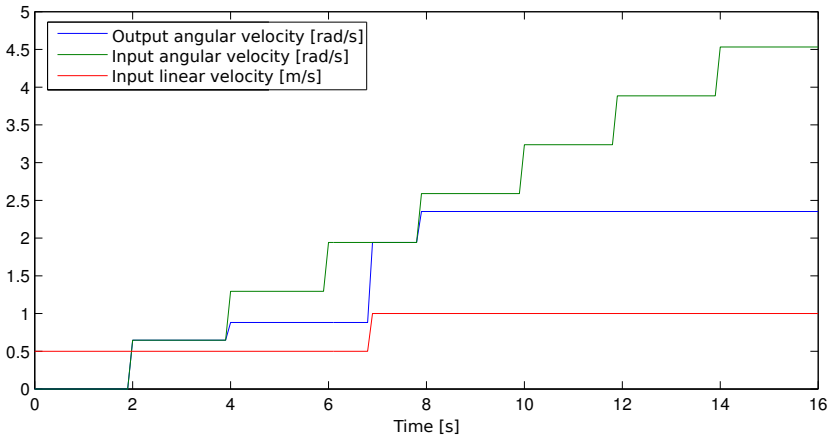


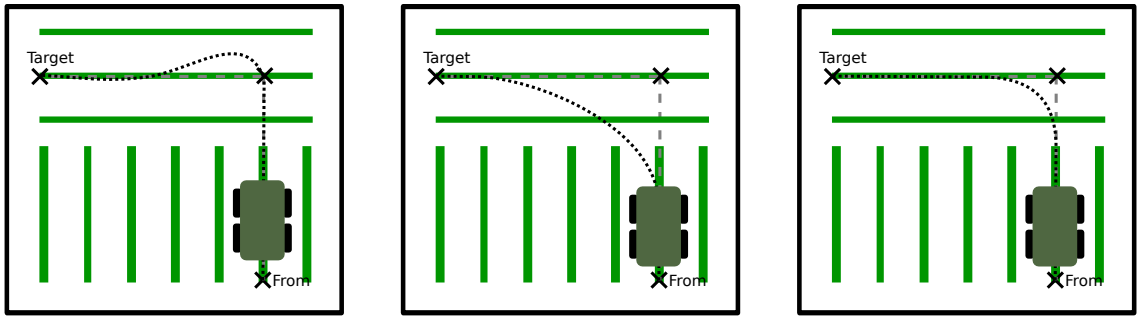
Figure 5.6. Output of the *Angular velocity limiter* block when forward velocity is stepped from 0.5 m/s to 1 m/s and angular velocity is increased gradually.

5.4 Change Line Segment Distance

In the previous sections the task of following a straight line has been discussed. When driving in a beet field changing from one line segment to another is necessary when reaching the end of a beet row. It has been chosen to leave the current line segment at a distance from the end point of the current line segment called the *change line segment distance*.

If no limitation was set on the angular velocity a solution would be to drive to the end point of the current line segment and turn on a spot to follow the new line segment.

The limitations placed on the angular velocity imply that the radius of rotation increases from zero. Therefore an overshoot will occur if the change of line segment is done when the robot reaches the end point of the current line segment, as shown in figure 5.7(a). On the other hand leaving the current line segment too early will cause the robot to drive unnecessary on the crops of the field, as shown in figure 5.7(b). It has been chosen to aim for a solution where no overshoot is experienced and crop damage is limited, as shown in figure 5.7(c).



(a) Case 1. (b) Case 2. (c) Case 3.

Figure 5.7. Three different situations of when leaving the current line segment.

To be able to use the proposed solution the radius of rotation must be determined. The radius is determined when one wheel pair have zero velocity. Having zero velocity on one wheel pair leads to the minimum radius of rotation, which is the distance 0.34 m from one wheel pair to the centre of the RobuROC4. The controller structure implies that this minimum radius of rotation is only obtained when the error angle is relatively large, since this would require high angular velocity such that one wheel pair is not turning. The controller structure implies that the angular velocity decreases as the angle error decreases, meaning the rotation radius increases as the error angle decreases. In order to compensate for this an estimate of the *effective radius* of rotation is used. This radius is chosen to be three times the minimum radius given by $c = 0.34\,\text{m}$:

$$r = 3 \cdot c = 3 \cdot 0.34 = 1.02\,\text{m}$$

In figure 5.7 the situation is shown when the angle between the current line segment and the next line segment is 90 °. This angle could take any value in the interval $[0, 180°]$ and therefore the distance x can not be set to a fixed value, instead it must depend on the angle between the current and the next line segment. The problem is illustrated in figure 5.8 and figure 5.9 for two different angles θ_{next} between the current and the next line segment.

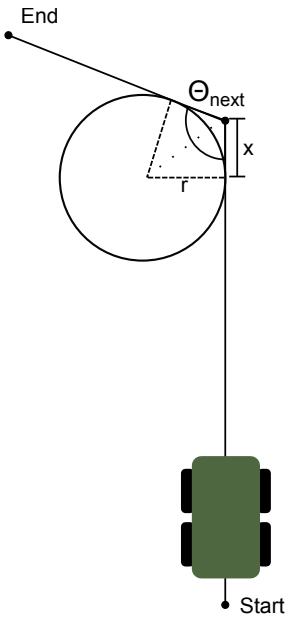


Figure 5.8. Situation with $\theta_{\text{next}} > 90°$.

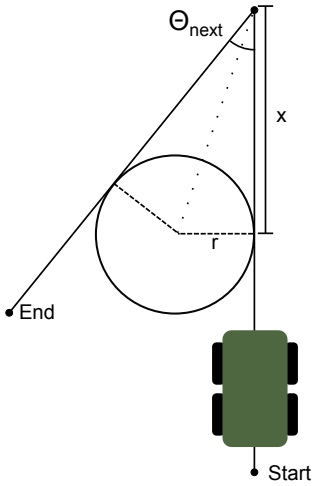


Figure 5.9. Situation with $\theta_{\text{next}} < 90°$.

As seen in the figures the distance x (*change line segment distance*) has to be larger when θ_{next} is small. From the figure it is seen that the distance x is calculated from θ_{next} and radius r as:

$$tan\left(\frac{\theta_{\text{next}}}{2}\right) = \frac{r}{x}$$

$$\Downarrow$$

$$x = \frac{r}{tan\left(\frac{\theta_{\text{next}}}{2}\right)} \tag{5.6}$$

The graph in figure 5.10 shows the relation between θ_{next} and x given in equation 5.6. As seen in the figure the distance x decreases to zero as the angle θ_{next} increases to 180°. Since a distance of zero could never be guaranteed in practice the minimum distance x is limited by the the precision requirement r.1 in the requirement specification on page 7.

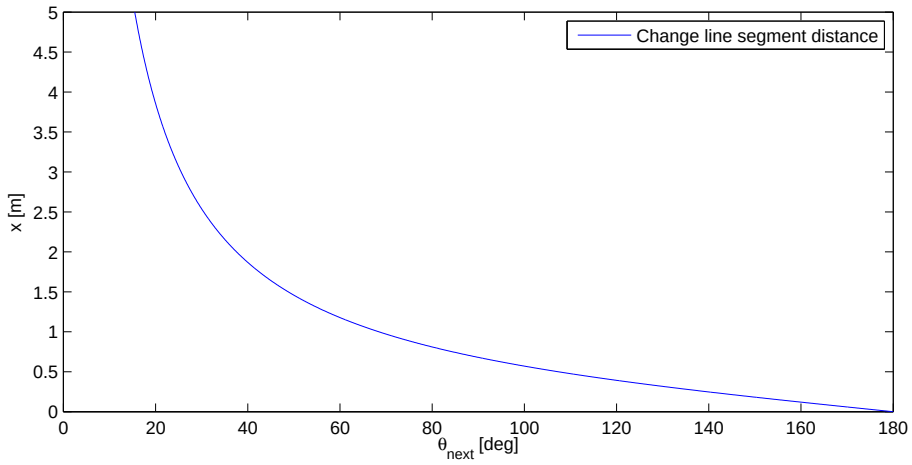


Figure 5.10. Change line segment distance x as a function of θ_{next}.

5.5 Simulation of Controller Algorithm

This section inspects how the designed controllers and limitations operate together. The controller structure depicted in figure 5.1 is fed with six waypoints to create a route that tests a series of different turns. The waypoints to define the line segments are as follows: (0;0), (20;0), (20;20), (5;5), (0;20), and (0;0).

Figure 5.11 shows the simulated path of the ALTAR. We see that the robot follows the line segment and breaks off its route before a turn to minimise the overshoot. Figure 5.12 shows the orthogonal error from ALTAR to the line during the run. The orthogonal error converges after each turn as intended, but the overshoot of approximately 0.5 m after each turn is unacceptable.

From these two plots it is clear that the controller algorithm does enable the RobuROC4 model to follow a route defined by waypoints but overshoots. But since the robot must be able to follow the line segment within ±55 mm as specified in requirement r.1, a different approach is therefore taken to obtain higher precision.

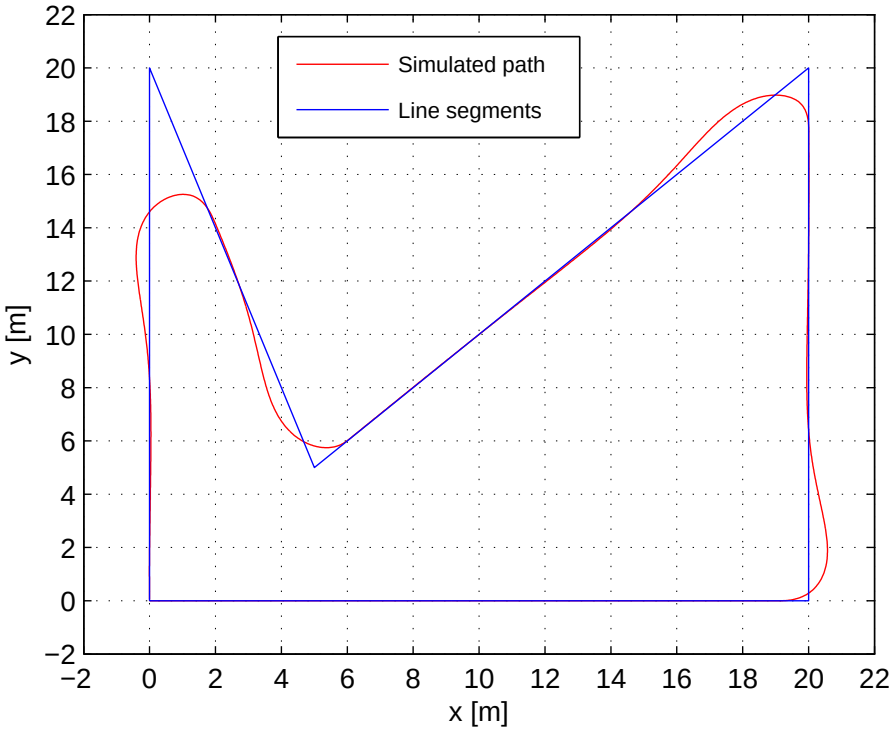


Figure 5.11. Simulated position of the ALTAR following a route of line segments with maximum speed.

In figure 5.13 and 5.14 the same run has been simulated. This time both maximum forward velocity and angular velocity is limited to 50 % of the values found in table 4.2. Compared to the other run at full speed, the maximum orthogonal error at each turn remains almost the same since the controller is asked to break off from a line segment at the same distance as before. The overshoot after the first turn has been reduced to 40 mm, which complies

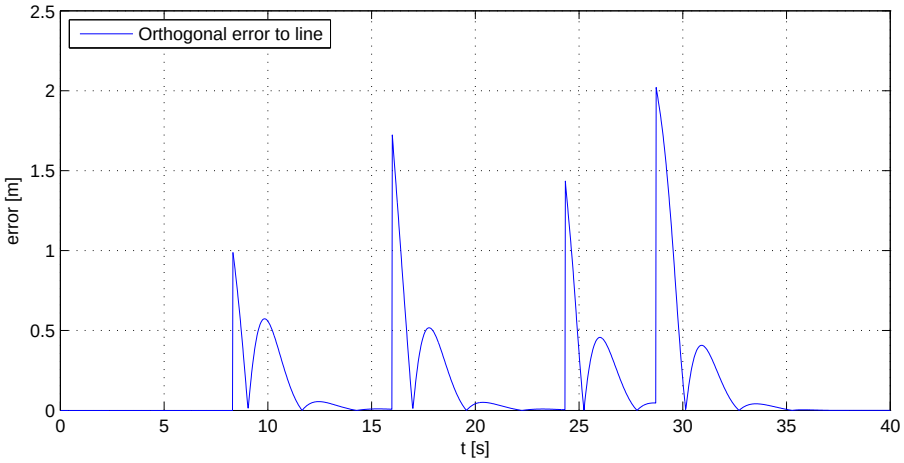


Figure 5.12. Simulated orthogonal error to route of line segments when driving with maximum speed.

with requirement r.1 of being able to follow a line within $\pm 55\,\mathrm{mm}$.

From these simulations the forward and angular velocity will be reduced to 50 % of maximum during the acceptance test. This choice is also supported by the fact that major slip has been observed when angular velocity is chosen close to its maximum of $3.51\,\mathrm{rad/s}$.

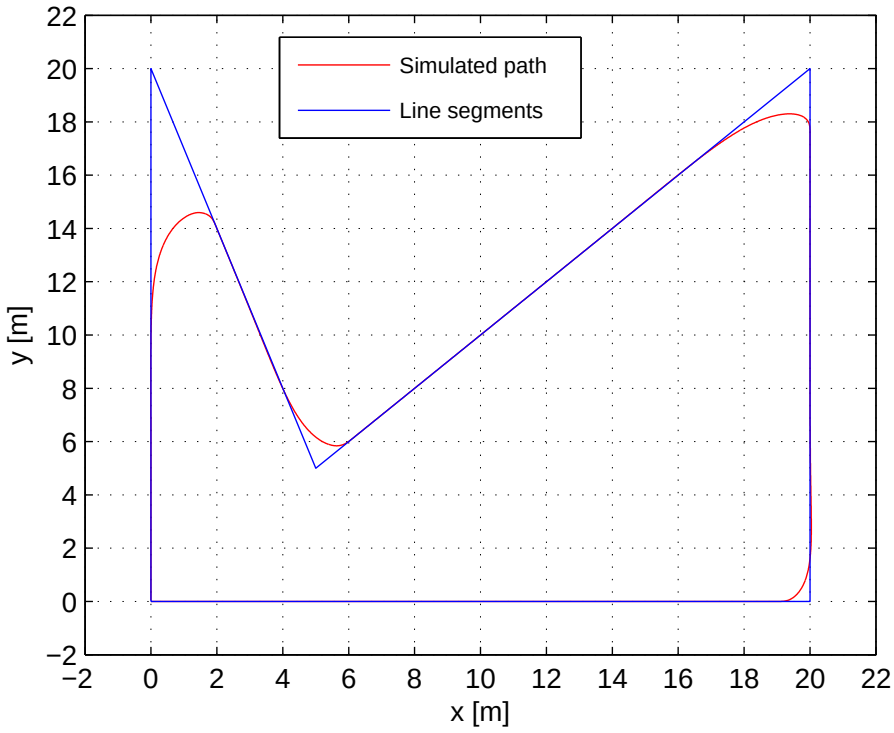


Figure 5.13. Simulated position the ALTAR following a route of line segments at reduced speed.

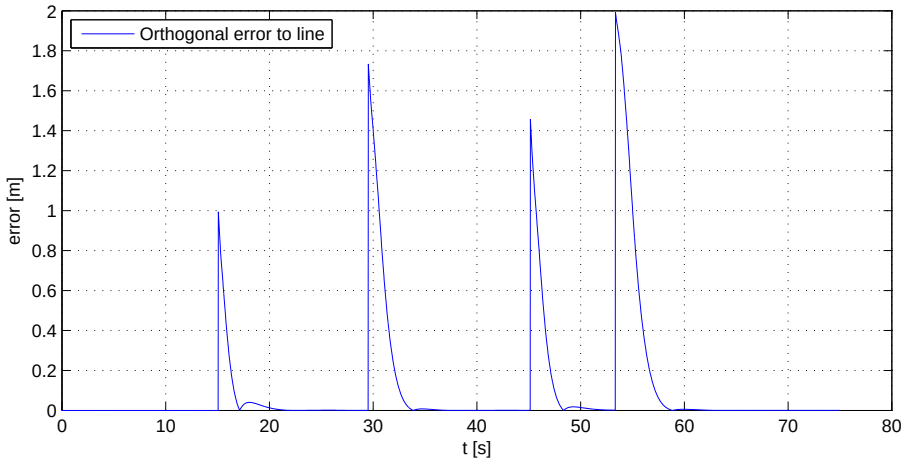


Figure 5.14. Simulated orthogonal error to route of line segments when driving at reduced speed.

6 Software

This worksheet desribes the developed software for this project. First the overall software structure is presented, afterwards the individual functionalities are described.

The software used to control the ALTAR is based on the Robot Operating System (ROS) framework [8]. The ROS framework provides a simple method to create processes and interprocess communication. In the ROS framework processes are called Nodes. Nodes are created as independent programs in either `C++` or Python, in this project all nodes are written in `C++`. The source code for the implemented nodes is placed on the cd `/software/ROSRobuROC4/src/`.

The ROS framework provides two different types of interprocess communication: topics and services. A node can publish information on a topic with a given message type, without knowing which nodes, if any, are listening to that topic. In addition a node can provide a service to other nodes, a service corresponds to a function call with specified parameters and return values. If a node needs information from a topic, it simply needs to subscribe to this topic.

Figure 6.1 shows the nodes, services and topics that are implemented as the software part of this project. All the code is executed on the onboard computer specified in table 2.2.

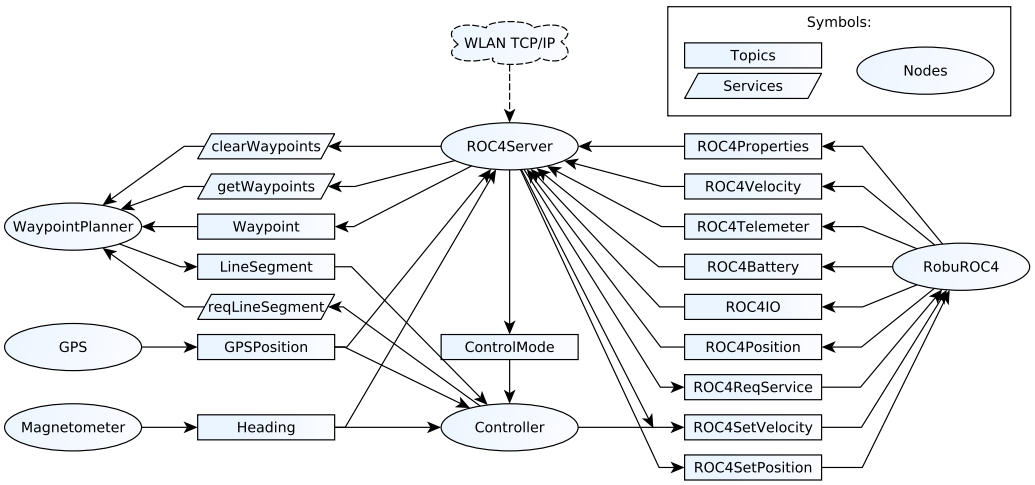


Figure 6.1. Overview of the software structure in the ALTAR system.

As shown in figure 6.1 the system consists of six nodes. Four nodes are created to handle communication to external devices (GPS, Magnetometer, RobuROC4 and ROC4Server), one node is created to handle the planning job (WaypointPlanner), and the last node is the node implementing the controller (Controller). In table 6.1 the responsibilities of the individual nodes are listed.

Node name	Responsibility
RobuROC4	Communication with the RobuROC4 and providing a simple interface to use the functionalities of the RobuROC4.
GPS	Read data from the GPS, decrypt, manipulate, and publish it on a topic to make it accessible for other nodes.
Magnetometer	Read data from the Magnetometer, decrypt, manipulate, and publish it on a topic to make it accessible for other nodes.
ROC4Server	Communication with external clients using the communication protocol defined in worksheet 7.
WaypointPlanner	Provide line segments to the controller on request, based on the waypoints given by the MCC.
Controller	Control the RobuROC4 using the control procedure described in worksheet 5.

Table 6.1. Nodes in the ALTAR system.

As shown in figure 6.1 the majority of the inter node communication is comprised by use of topics. It is chosen to use topics rather than services because the primary stream of information is published by a node periodically. In addition the use of topics to transport information makes the development process easier, since it is easy to monitor the data flow in the total system just by listening to the topics, without affecting the functionality of the system.

In table 6.2 is a description of all messages used on topics in the system, and in table 6.3 the services are described. The definition of the message types and services are found on the cd in /software/ROSRobuROC4/msg and /software/ROSRobuROC4/srv respectively.

Topic name	Message filename	Content
Waypoint	Waypoint.msg	A longitude/latitude coordinate of the waypoint.
LineSegment	LineSegment.msg	A line segment containing a start and end point given by their longitude/latitude coordinates. In addition the angle between the line segment contained in the packet and the next line segment to indicate how sharp a turn the Robot needs to turn when changing to the next line segment.
GPSPosition	GPSPosition.msg	A longitude/latitude coordinate of the current position of the GPS. Linear velocity and heading estimated by the GPS. Precision information composed by the number satellites and RTK GPS Status.
Heading	Heading.msg	Heading calculated by the magnetometer and the heading calculated from the raw magnetic field outputs from the magnotometer.
ControlMode	ControlMode.msg	The current wanted control mode (1 = auto, 2 = manual/disabled, 3 = Sensor calibration).
ROC4Properties	ROC4Properties.msg	A Package containing all stationary properties of the RobuROC4, e.g. width, max speed, min/max acceleration.
ROC4Velocity ROC4SetVelocity	ROC4Velocity.msg	Read or set a linear and an angular velocity.
ROC4Telemeter	ROC4Telemeter.msg	Distance to objects in front of the ultrasonic sensors *(Not implemented)*.
ROC4Battery	ROC4Battery.msg	Percentage of remaining power and the current state of the battery in the RobuROC4.
ROC4IO	ROC4IO.msg	Analogue/digital inputs and outputs on the internal RobuROC4 IO card *(Not implemented)*.
ROC4Position ROC4SetPosition	ROC4Position.msg	Read or set Longitude/latitude coordinate of the RobuROC4 position, and the heading. In addition an indication of the precision of the estimates is included in the message, when the message is used in the ROC4Position topic.
ROC4ReqService	ROC4ReqService.msg	ID of the service (in the RobuROC4) of which information is wanted, and the period (in 10 ms) of which the information is wanted.

Table 6.2. Topics in the ALTAR system.

Service name	Provided by	Service description
clearWaypoints	WaypointPlanner	Erases all the waypoints in the planner (no information is returned).
getWaypoints	WaypointPlanner	Returns the wanted waypoints.
reqLineSegment	WaypointPlanner	Generates a new line segment, and publishes it on the topic (no information is returned).

Table 6.3. Services in the ALTAR system

As seen in table 6.3, the reqLineSegment service does not respond with the requested line segment. Instead the line segment is published on a separate topic. This construction is made like this due to the following two scenarios:

1. The controller has no line segments e.g. just after start up and in stead of polling for a new line segment the controller could just wait for the line segment to be published on the topic, when the waypointPlanner have enough waypoints to construct it.
2. The current route (given by the waypoints in the waypoint planner) is cancelled, and all waypoints are cleared from the waypoint planner. With this construction the controller is immediately informed because an *invalid* line segment is published on the lineSegment topic.

6.1 ControllerNode

The main purpose of the controller node is to implement the controller described in worksheet 5. In addition it must handle the inputs from the sensors and manipulate them when needed.

The controller has three different modes: *Auto*, *Manual*, and *Sensor calibration*. The *Auto* mode is the primary mode and is used when the controller should run and set the velocity of the robot using the ROC4SetVelocity topic. In manual mode the controller is disabled, and velocities can be published to the ROC4SetVelocity topic from the MCC without influence from the controller. The *Sensor calibration* mode is implemented to synchronise the heading inputs from the GPS and the magnetometer, and is an extended version of the manual mode.

6.1.1 Auto mode

The controller mode implements the controller described in worksheet 5. The structure is sequential and follows the following structure:

1. Calculate the distance to target d.
2. Calculate the aiming point P_a .
3. Calculate the angle error Θ_e.
4. Calculate the velocities from the calculated error and distance.
5. Reduce angular velocity to avoid negative wheel velocities.

After running through the steps in the controller, ROS is asked to see if new data has arrived on the topics.

The control loop runs at 20 Hz since this is the maximum output rate of the GPS sensor. The magnetometer can output data up to a rate of 40 Hz, but this rate has been reduced to 20 Hz as well.

6.1.2 Sensor calibration mode

As described in the introduction of the ControllerNode the heading input from the GPS and the magnetometer must be synchronised. This synchronisation is necessary due to misalignment of the magnetometer causing the output from the magnetometer to have an offset compared to the actual heading of the vehicle. This misalignment must be compensated for. In the sensor calibration mode the heading offset is determined.

The main idea is to use the heading from the GPS as a reference heading, and compare this to the output of the magnetometer. The difference between these two headings is the offset. The GPS has the highest heading precision when it is moved fast. To generate a precise GPS heading the vehicle must manually be driven on a straight line. When the linear velocity, measured with the GPS, is above 2 m/s, the headings estimated from the GPS are saved for 60 successive readouts. The average heading is then calculated and used as true heading Θ_{true} of the robot. The offset angle Θ_{offset} is then calculated as:

$$\Theta_{\text{offset}} = \Theta_{\text{true}} - \Theta_{\text{magnetometerHeading}} \tag{6.1}$$

This offset is stored and used in the manipulation of the magnetometer data described in section 6.1.3.

6.1.3 Magnetometer data manipulation

Before the data from the magnetometer can be used in the control-loop the heading must be converted to be within $[0, 2\pi[$ with positive direction counter clockwise. From the heading topic, heading is indicated with positive direction clockwise between $[0°, 360°[$. The offset calculated in the sensor calibration mode is then added to the output from the magnetometer to achieve correct heading. This offset is however only added when the controller is not in sensor calibration mode (controlMode 3).

$$\Theta_{\text{magnetometerHeading}} = \begin{cases} \Theta_{\text{magnetometer}} & \text{if controlMode} = 3 \\ \Theta_{\text{magnetometer}} + \Theta_{\text{offset}} & \text{otherwise} \end{cases} \tag{6.2}$$

where:

$\Theta_{\text{magnetometerHeading}}$ is the modified heading used in the controller.

$\Theta_{\text{magnetometer}}$ is the heading directly from the topic given by the magnetometer.

6.2 WaypointPlannerNode

The purpose for the WaypointPlannerNode is to store the route that ALTAR has to follow, and provide it as reference to the controller. As described in worksheet 5 the reference to the controller is a line segment consisting of a start point and an end point. In addition to the line segment the waypoint planner must also provide the angle between the current line segment and the next line segment. This angle is used to determine when to leave the current line segment and start flowing the next one. The waypoints to generate the route from are provided to the WaypointPlannerNode by the MCC through the waypoint topic.

The line segments published on the topics are generated by selecting the first waypoint in the list as starting point, and the next waypoint in the list as end point. The angle is calculated by generating the first two possible line segments, and calculating the angle between them. If only one line segment can be generated (only two waypoints exists in the list), the angle is set to -1 to indicate that this is the last line segment available. When the line segment is generated it gets published on the line segment topic with the fields described in table 6.4.

Topic field name	**Description**
startLongitude	Longitude position of the line segment starting point.
startLatitude	Latitude position of the line segment starting point.
endLongitude	Longitude position of the line segment end point.
endLatitude	Latitude position of the line segment end point.
nextAngle	The angle between *this* line segment and the next line segment.

Table 6.4. Fields of the line segment topic.

The waypoint planner node publishes a new line segment to the line segment topic in the following cases:

1. When it is requested by use of the reqLineSegment service.
2. When the number of waypoints in the waypoint planner increases from one to two.
3. When the number of waypoints in the waypoint planner increases from two to three.
4. When the list of waypoints are cleared from the server node by use of the clearWaypoints service.

In case 1, the first waypoint is deleted from the list to generate the new line segment. In case 2, the waypoint planner informs other nodes that it has just received enough information to generate a single line segment by publishing the line segment. In case 3, the line segment is already published on the topic, but when going from two to three waypoints the nextAngle must be appended and the line segment topic is therefore updated with the angle information. In case 4, an empty line segment is generated (all coordinates are 0, and nextAngle $= -1$), to indicate that no information about the route is available.

6.3 MagnetometerNode

The magnetometer node has the purpose of reading the output data from the magnetometer and publishing it on a topic. The node initially sets up the chosen data output format, the chosen output parameters, and the data output rate. The format chosen for this project is the "$OHPR" format. In this format, the sentence starts with the phrase $OHPR followed by the comma delimited data output and a checksum. This output format is chosen, as it is easy to identify start and end of each sentence, and the comma delimiting makes it easy to break up the sentence to obtain the needed data. Another advantage of using this output format is that is almost similar to the output from the GPS module which eases implementation. The desired output from the magnetometer is the heading (Azimuth) of the vehicle. However the readout of the heading while driving is very fluctuating, which is assumed to be caused by the built-in filtering which uses accelerometers to determine pitch and roll and correct the magnetometer readings. Based on observations on the behaviour of the heading while driving, it is therefore chosen to readout the raw magnetic field readings, since these turns out to have a much more stable output while driving. The chosen output sentence looks as follows:

$OHPR hhh.h,Mx,My,Mz,*cc

In the sentence *h* denotes angle in degrees, *Mx, My, Mz* denotes the three magnetometer readouts and *cc* denotes the checksum of the sentence. An output sentence with the chosen setup can attain a maximum length of 48 ASCII characters corresponding to 480 bits including start and stop bits when send over the serial connection. The output rate from the magnetometer is set to 20 Hz. A baud rate of 19 200 bits/s is chosen, since this full fills the requirement to the data rate.

As mentioned the readouts of the magnetic fields are used for calculating the heading. Assuming that the fields where the vehicle is supposed to drive are close to ground level, variations in the Z field can be neglected. The direction of the vehicle is therefore calculated using only readings from X and Y axis of the magnetometer. The angle θ is calculated as follows:

$$\theta = tan^{-1}\left(\frac{My}{Mx}\right)$$

Since the current heading is defined in an interval of $[0°, 360°[$, the behaviour of the tangent function needs to be taken into consideration. An illustration of the angle calculated from the arcus tangent function is seen in figure 6.2. The vectors V_1, V_2 and V_3 are three random vectors composed of the components Mx and My.

Figure 6.2 shows how the angle of each of the vectors will be determined in the interval $]\pi/2, -\pi/2[$ caused by the arcus tangent function. These angles needs to be converted to an interval of $[0, 2\pi[$ as shown in figure 6.3. This conversion is carried out as follows:

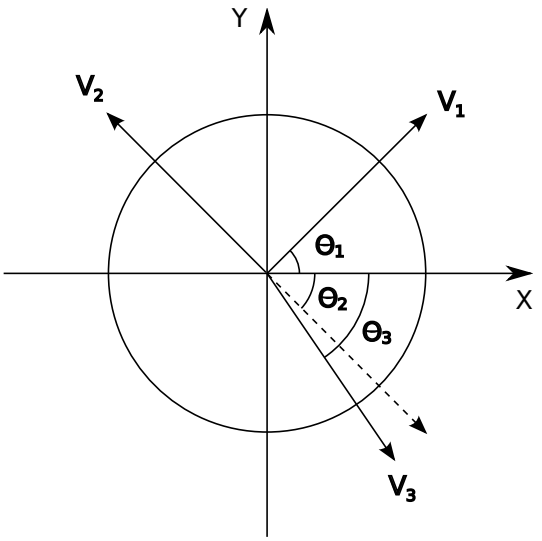


Figure 6.2. Angle θ calculated from Mx and My.

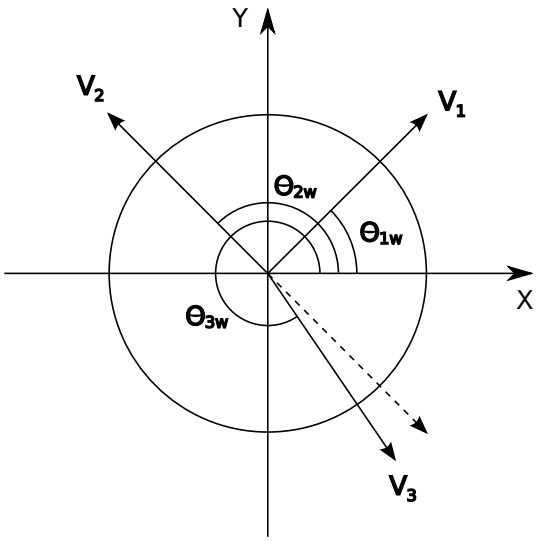


Figure 6.3. Wanted θ.

if $Mx < 0$

$$\theta_{\mathrm{w}} = \theta + \pi$$

if $Mx > 0$ and $My < 0$

$$\theta_{\mathrm{w}} = \theta + 2\pi$$

The heading is converted into degrees and a modulus operation is used to limit the angle to $[0°, 360°[$. The heading topic has two fields which are defined in table 6.5.

Topic field name	Description
heading	heading extracted directly from the data string field *hhh.h*, this is the heading determined by the magnetometer.
headingMag	The heading calculated as above from the raw magnetometer reading Mx and My.

Table 6.5. Data published on the Heading topic.

6.3.1 Calibrating Magnetometer

As a consequence of using the raw magnetic field readings, the magnetometer's built-in calibration function can not be used. This calibration function has the purpose compensating for distortions in the magnetic field readings [9].

Figure 6.4 shows a plot of the raw magnetic field readings from a test drive. These data are used to determine a static calibration.

As seen in the figure, the data from the measured magnetic field shapes a distorted circle. If the magnetic field is assumed to be constant these data has to represent a perfect circle with with centre in (0,0) to give correct headings. This distortion can be the result of many things. Since the RobuROC4 is made out of metal and has electric motors, this can be one of the reasons for the distortion. If the magnetometer is not perfectly aligned

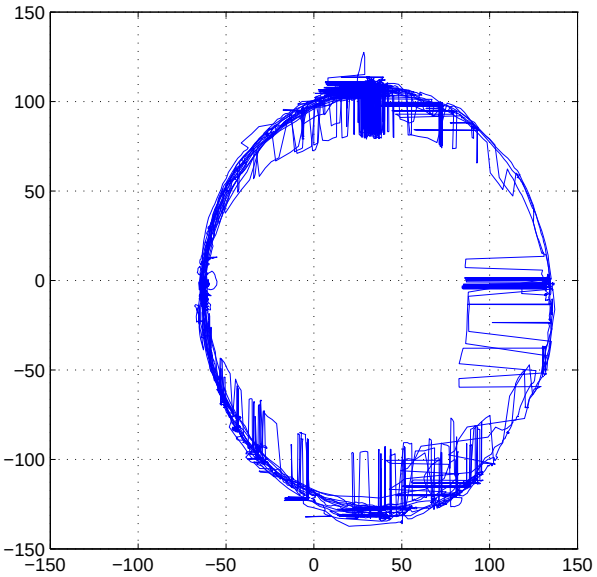


Figure 6.4. Raw magnetic field reading from the OS5000-US magnetometer without calibration.

in the horizontal plane this will also add a distortion when the angle is only determined on basis of the magnetic field in two axes. Since an advanced magnetometer calibration algorithm is outside the scope of this project, a static calibration is made based on the test data shown in figure 6.4.

From the figure it is also obvious that the readings are influenced by a significant amount of noise. A study of the raw magnetometer readings shows that these distortions occur each time the magnetometer degausses its sensors. At this stage these fluctuations are ignored due to their relative short duration of 0.2 s.

To determine the offset and the scaling factors, a circle with centre in (0,0) and a radius of 100 is plotted along with the measured data. The measured data can now be calibrated by fitting it to this circle. A plot of the circle and the calibrated data is seen in figure 6.5. The found constants for the calibration are listed in table 6.6. In the calibration the offset is first applied followed by the scaling.

Calibration factor	Value	Description
X_{off}	36.5	Offset in X axis
Y_{off}	-12.5	Offset in Y axis
α	1.005	Scaling of X
β	0.851	Scaling of Y

Table 6.6. Calibration factors for magnetometer.

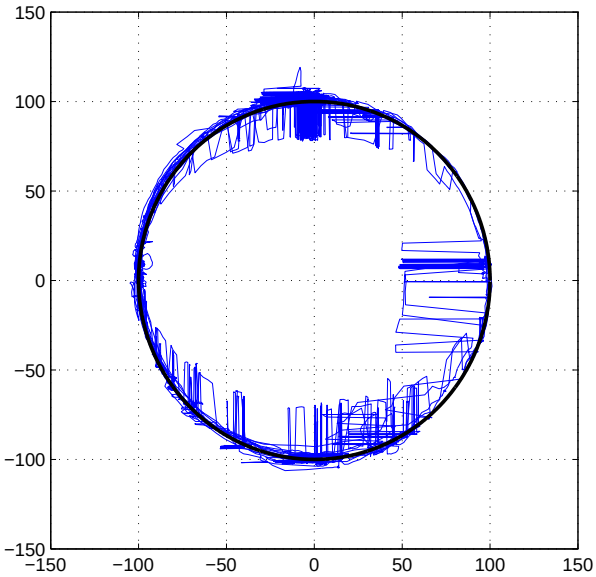


Figure 6.5. Magnetic field data after calibrating.

6.4 GPS Node

The purpose of the GPS Node is to handle the communication with the GPS described in section 2.1.2 and extract data from it. The communication with the GPS uses the serial communication interface provided by the GPS. The GPS Node must handle the extraction of the position in the position output data given by the GPS. The GPS is set up to provide data with 20 Hz with the following NMEA sequence:

$PASHR,POS,d1,d2,m3,m4,c5,m6,c7,f8,f9,f10,f11,f12,f13,f14,f15,f16,s17*cc

An output string of this type can take a maximum value of 114 ASCII characters, which corresponds to 1140 bits including the appended start and stop bits in the serial communication. With output data rate of 20 Hz, a baud rate of 115 200 bits/s is chosen since this is the standard setup and easily fast enough.

From the data sequence, the data listed in table 6.7 is extracted and published on the GPSPosition Topic.

6.5 RobuROC4Node

The main purpose of this node is to handle communication with the RobuROC4. The connection to the vehicle is Ethernet and the transmission protocol is UDP. The protocol used for transmitting and receiving data to and from the vehicle is predefined and described in the RobuROC4 communication manual .

The RobuROC4Node listens to topics and handles requests from other nodes e.g. setting velocity, setting position and service subscriptions. These requests are read from

Topic field name	NMEA field	Description
posMode	d1	Position mode is extracted directly from the data string field *1*, this number indicates if the GPS runs in autonomous mode or it uses RTK information.
nSats	d2	The number of satellites is directly extracted from the data string field *2*.
latitude	m4, c5	The Latitude is extracted from the data string by combining data field *4* and *5*. Data field *4* contains the latitude position, and data field *5* contains whether the latitude position is North(N) or South(S), if it is S the latitude is given as the negative latitude on the topic.
longitude	m6, c7	The Longitude is extracted from the data string by combining data field *6* and *7*. Data field *6* contains the longitude position, and data field *7* contains whether the longitude position is East(E) or West(W), if it is W the longitude is given as the negative longitude on the topic.
heading	f10	The heading is extracted directly from the data string field *10*.
linearSpeed	f11	The linear speed in knots is extracted from the data string field *11* and converted to m/s.

Table 6.7. Data used from the NMEA string.

the topics, decoded according to the topic message type, and encoded for the transmission protocol before the message is sent to the RobuROC4.

The RobuROC4 returns data in two different situations. The first situation is as a response to a request e.g. when a subscription is set up, or some properties of the RobuROC4 are requested. The second situation is when outputting data from the internal sensors periodically.

Since data can arrive periodically and not only once per request, two different threads are created and used for transmitting and receiving data from the RobuROC4. One thread is used for listening on the topics, and transmitting data to the RobuROC4, and the second thread is used to listen for incoming messages from the RobuROC4. In the thread handling incoming data from the RobuROC4, the data is decoded from the RobuROC4 protocol, and repacked to fit the ROS topic structure before it is published on the topics.

The data published from the nodes comply with the topics shown in figure 6.1 and described table 6.2.

6.6 ServerNode

The server node implements the protocol described in 7.1. The node handles all data sent from the RobuROC4's internal sensors, the magnetometer, and the GPS. All this data is unpacked from the ROS message format and repacked to the message format specified in the protocol. All received data from the topics is stored internally in the node. An internal timer controls when to send data to the MCC. The frequency at which the data is sent, is controlled by the MCC. The software allows data to be sent to MCC at a frequency ranging from 0.015 Hz to 100 Hz. This node as well handles commands, including setting waypoints, sent from the MCC.

7 Communication Protocol

This worksheet describes the communication protocol developed for communication between the ALTAR and the Mission Control Center (MCC). The first part describes the functionality implemented in the protocol together with a description of each field. The second part contains network calculations to determine the needed bandwidth for the connection between the MCC and the robot.

7.1 Mission Control Communication

As stated in worksheet 3, a protocol has to be implemented in order to enable communication between the MCC and the ALTAR. The overall functionality of the protocol is specified through requirement c.1, c.2 and c.3. c.1 states that the MCC must be able to add and remove line segments on the ALTAR. c.2 states that theALTAR must be able to return sensor data to the MCC upon request. c.3 specifies that the ALTAR must be able to return sensor data periodically.

During this project the communication interface is a WiFi connection. This is not the ideal solution for the final implementation of the product due to its limited range. Changing the wireless connection to one with lower carrier frequency can extend the range, but also reduces the bandwidth. This consideration is taken into account while designing the protocol. The protocol is designed in a way that limits the protocol overhead of the data to a minimum.

For this project the designed protocol is implemented on top of a TCP-protocol. Since TCP manages dropped packages and corrupted data, the designed protocol does not have any checksum to check for these events.

The protocol has three different groups of packets: *system error*, *status message*, and *command message*. *system error* is a group of packets which can be send from the ALTAR to the MCC in order to report errors. This could be everything from low battery to report that the robot is stuck.

status message is a group of packets which are transporting sensor data from the ALTAR to the MCC. This could e.g. be data with current position, battery level or current heading. When the MCC subscribes to a sensor, data will be received as status messages.

The third and last group of packets is *command message*. These packets carries commands from the MCC to the ALTAR which will return a packet dependent on the command. Table 7.5 shows a full list of the different commands which can be send.

Each packet of the protocol carries different fields depending of its type. All packets

however have a field called *header*. The header is a 1 byte field to distinguish the three groups. This field is defined such that:

- `0x00` is a *system error*
- `0x01` - `0xFE` is a *command message*
- `0xFF` is *status message*

The header of *command message* ranges from `0x01` to `0xFE` to be able to give each command an ID. This allows the MCC to distinguish which responses belongs to which commands. If a command with header `0x01` is sent, the response will have to carry the same header in order for the MCC to acknowledge that the command has been received and understood by the ALTAR.

In the following sections the structure of the different packets will be given depending on what data they contain. The protocol is implemented in `C++` using the boost serialisation library [10] to serialise the packet classes.Tthe implemented packed classes are found in folder `/software/externalComm/`. In the ALTAR the protocol is handled by the ServerNode described in section 6.6.

7.1.1 System error

This group always has the packet structure shown in figure 7.1. The **errorCode** field is then changed depending on the kind of error which has occurred. The error codes have not been specified at this time since error handling has been of low priority. The size of each field can be found in table 7.1.

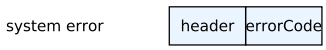


Figure 7.1. *Error message* packet structure.

Field name	Size [bytes]	Description
header	1	identifies the messages group
errorCode	1	identifies the type of error

Table 7.1. Field description of an *error message*.

7.1.2 Status message

The *status message* contains sensor readings sent from the ALTAR to the MCC. The packet structure shown in figure 7.2 varies depending on which data is send. The field **sensorID** specifies which kind of sensor data is returned. The size of the **data** field is varied depending on the sensor type. The size of each field can be found in table 7.2.

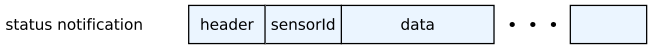


Figure 7.2. *Status message* containing N bytes of data from a sensor.

Field name	Size [bytes]	Description
header	1	identifies the messages group
sensorId	1	identifies the sensor
data	N	data from sensor

Table 7.2. Field description of a *status* message.

Table 7.3 specifies the different sensor data outputs which are possible to receive from ALTAR through the protocol. The messages marked with "-" indicates that this message type has not yet been implemented in the protocol.

Status type	Field names	Data types	Size [bytes]
GPS	latitude, longitude	double, double	16
Magnetometer	heading	double	8
IOcard	-	-	-
Telemeter	-	-	-
Battery	state, remainingPower	uint8, uint8	2
Localisation	theta, lat-, longitude, state	double, double, double, uint32	28
Differential	linearSpeed, angularSpeed	float, float	8

Table 7.3. Description of data in the *status messages*.

Table 7.4 gives a short description of each of the status. For more information about the RobuROC4 specific status: *IOcard*, *Telemeter*, *Battery*, *Localisation*, and *Differential* refer to the RobuROC4 user's Manual .

Status type	sensorId	Description
GPS	`0x10`	Current position of the robot measured by GPS.
Magnetometer	`0x11`	Current heading determined from the magnetometer.
IOcard	`0x02`	Analogue and digital readings from the IO card inside the RobuROC4.
Telemeter	`0x03`	Information about the ultrasonic sensors placed in both ends of the RobuROC4.
Battery	`0x04`	Remaining power on RobuROC4 battery.
Localisation	`0x05`	The RobuROC4's estimate of location based on counted wheel rotations.
Differential	`0x06`	The RObuROC4's estimated linear speed and angular speed.

Table 7.4. Definition of *status messages*.

7.1.3 Command message

In the following section all the *command messages* are defined. The fields of these are dependent on the specific command message type. The different types of command messages are listed in table 7.5.

Message type	typeId	Description
Subscription request	0x01	Request for subscription to status messages.
Cancellation request	0x02	Request to cancel subscription from status messages.
Set operation mode	0x03	Enables the opportunity of changing operation mode.
Get operation mode	0x04	Enables the opportunity of getting current operation mode.
Abort operation	0x05	Brings the vehicle to halt.
Set differential	0x06	Set the linear and angular velocity of the robot.
Add waypoint	0x07	Add waypoint to list on ALTAR.
Clear waypoint	0x08	Delete all waypoints from list on ALTAR.
Get waypoints	0x09	Get waypoints from list on ALTAR.

Table 7.5. Definition of *command messages*.

Subscription request

This command allows the MCC to subscribe to one of the status defined in table 7.3 with a specific time interval. This period is defined in multiples of 10 ms. When a subscription has been made the ALTAR will keep sending status messages until it is unsubscribed. Figure 7.3 and table 7.6 specifies the fields of this message.

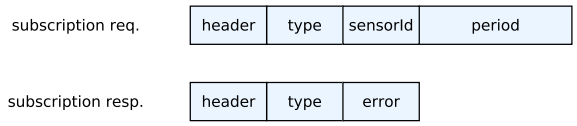


Figure 7.3. Message to make the robot periodically return the status of a specific sensor.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the message group
type	1 (uint8)	identifies the command type (0x01)
sensorId	1 (uint8)	identifies the sensor
period	2 (uint16)	specifies period of sensor status messages in multiples of 10 ms
error	1 (uint8)	indicates if message has correct format

Table 7.6. Field description of a *subscription* message.

Cancellation request

In order for the MCC to unsubscribe a given status, a *cancellation request* must be send to the ALTAR with the given *sensorID*. Figure 7.4 and table 7.7 specifies the fields of this message.

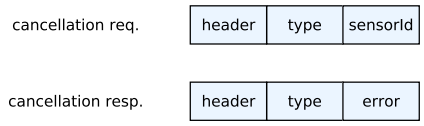


Figure 7.4. Message that cancels the periodic status from a specific sensor.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x02`)
sensorId	1 (uint8)	identifies the sensor
error	1 (uint8)	indicates if message has correct format

Table 7.7. Field description of a *cancellation* message.

Set operation mode

The ALTAR has been implemented with three different operation modes. To change between these the *set operation mode* command is send with a parameter defining which mode to operate in. Figure 7.5 and table 7.8 specifies the fields of this message.

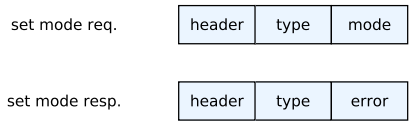


Figure 7.5. Message to set operation mode of the ALTAR.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x03`)
mode	1 (uint8)	indicates the operation mode of the robot
error	1 (uint8)	indicates if message has correct format

Table 7.8. Field description of a *set mode* message.

Get operation mode

Through this message the MCC can request the robot to return its current operation mode. Figure 7.6 and table 7.9 specifies the fields of this message.

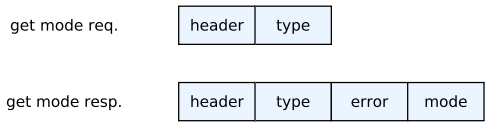


Figure 7.6. Command that makes the ALTAR return its current operation status.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x04`)
mode	1 (uint8)	indicates the operation mode of the robot
error	1 (uint8)	indicates if message has correct format

Table 7.9. Field description of a *get mode* message.

Abort operation

In case of errors the MCC can order an immediate halt of the ALTAR by sending *Abort operation*. Figure 7.7 and table 7.10 specifies the fields of this message.

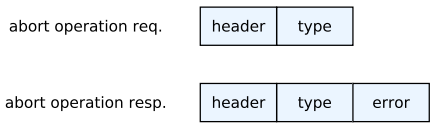


Figure 7.7. Command the ALTAR to stop.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x05`)
error	1 (uint8)	indicates if message has correct format

Table 7.10. Field description of an *abort operation* message.

Set differential

When the *set differential* is send to the ALTAR, it leaves its current operation mode and enters manual mode. The *linearSpeed* and *angularSpeed* from the message is then send to the vehicle to allow manual control. Figure 7.8 and table 7.11 specifies the fields of this message.

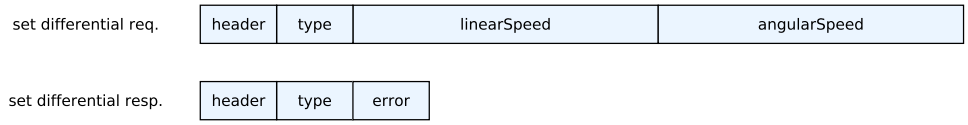


Figure 7.8. Command used to control the RobuROC4 remotely.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x06`)
linearSpeed	4 (float)	specifies wanted linear speed in m/s
angularSpeed	4 (float)	specifies wanted angular speed in rad/s
error	1 (uint8)	indicates if message has correct format

Table 7.11. Field description of a *set differential* message.

Add waypoint

This message adds a waypoint in the end of the waypoint list of the waypointPlannerNode. Waypoints in the beginning of the list are processed first. Figure 7.9 and table 7.12 specifies the fields of this message.

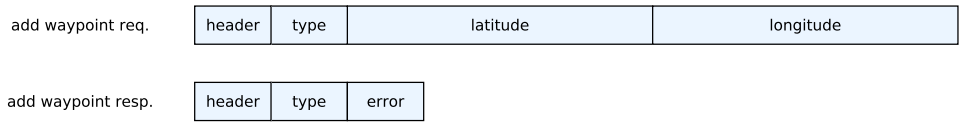


Figure 7.9. Command to add a new waypoint to the ALTAR's route.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x07`)
latitude	8 (double)	specifies latitude
longitude	8 (double)	specifies longitude
error	1 (uint8)	indicates if message has correct format

Table 7.12. Field description of an *add waypoint* message.

Clear waypoints

The only way to alter waypoints already specified in the list is by sending a *clear waypoint* message. This clears all waypoints in the list and new waypoints can then be added. Figure 7.10 and table 7.13 specifies the fields of this message.

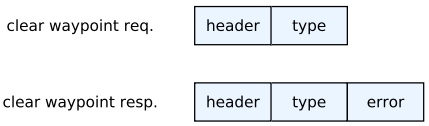


Figure 7.10. Command that clears all waypoints in the current route.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x08`)
error	1 (uint8)	indicates if message has correct format

Table 7.13. Field description of a *clear waypoint* message.

Get waypoints

The MCC has the possibility to request waypoints stored in the robot's waypoint list. In general these waypoints are requested with a *from* and *to* field, which specifies an interval in the waypoint list that should be send. If the *from* field is set to 0 or 1 waypoints will be send from the first waypoint in list. If 0 or a number larger than list length is set as the *to* field all waypoints in between *from* to the end of list is send. When the ALTAR sends a packet with waypoints the field N defines how many waypoints are send. Figure 7.11 and table 7.14 specifies the fields of this message.

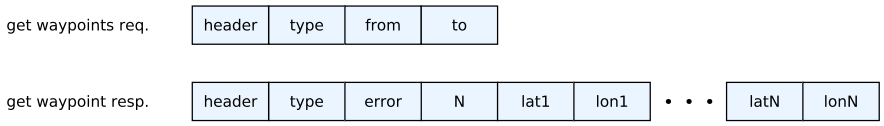


Figure 7.11. Command that makes the ALTAR return N waypoints in the current list.

Field name	Size [bytes]	Description
header	1 (uint8)	identifies the messages group
type	1 (uint8)	identifies the command type (`0x09`)
from	1 (uint8)	specifies first waypoint of the returned route
to	1 (unit8)	specifies last waypoint of the returned route
N	8 (uint64)	indicates how many waypoint are returned
lat_i	8 (double)	latitude of the i'te waypoint
lon_i	8 (double)	longitude of the i'te waypoint
error	1 (uint8)	indicates if message has correct format

Table 7.14. Field description of a *get waypoint* message.

7.2 Network calculation

This section covers calculations on the network load, which the developed communication protocol puts on the wireless link. Calculations are carried out for different scenarios. In this project WiFi is used as wireless link, but other wireless links are also considered. WiFi requires setting up an access point which in some fields can be difficult and expensive, for instance if the field is located far away from the MCC. The usage of public available wireless links such as General Packet Radio Service (GPRS) could therefore be beneficial. Furthermore internet via satellites is considered, since GPRS is not always available. The chosen internet via satellite connection is Iridium[11] since it claims global coverage and has a phone with a antenna that could be mounted on the RobuROC4. The theoretical operation bandwidth for the three mentioned technologies are given in table 7.15, since the data rate depends on many different factors, a typical and maximum is given.

Technology	Typical [kbit/s]	Maximum [kbit/s]
GPRS [12]	56	114
Internet via satellite phone [11]	9.6	128
WiFi IEEE 802.11a [13]	25000	54000

Table 7.15. Theoretical data rates for different wireless communication technologies.

The data load and the required data rate in different scenarios will be compared to the bandwidth of the different wireless technologies. The different scenarios are: *Receiving status updates periodically from the ALTAR*, *Adding waypoints to the ALTAR*, and *Driving the ALTAR externally.*

Many scenarios could be considered, however these cases are chosen, because they

are considered as typical and plausible examples on operation modes.

During the calculations a TCP IPv6 header is taken into account. A TCP header is 20 bytes and an IPv6 header is 40 bytes, resulting in a total header of 60 bytes for every packet sent over the wireless link. In all cases up and down streams are merged, meaning that possible difference in up and down stream rates on the wireless link is not taken into account. In all cases it is assumed that no retransmission of the packages is necessary (no packet loss). The data sizes used are based on the message descriptions in section 7.1.

7.2.1 Receiving status updates periodically

This scenario concerns data sent to the MCC from the ALTAR periodically. Subscription to sensor data is not take into consideration.

Three cases will be considered: Low, Medium and High. The information sent periodically is shown in the list below. The numbers indicates is the size of a single packet (in Bytes) without TCP and IPv6 headers. These sizes are calculated from the tables in section 7.1.2.

$$\begin{aligned}
\text{Low: GPS } &(16 + 2\,\text{B}) \\
\text{Medium: GPS } &(16 + 2\,\text{B}) + \text{magnetometer } (8 + 2\,\text{B}) \\
\text{High: GPS } &(16 + 2\,\text{B}) + \text{magnetometer } (8 + 2\,\text{B}) + \text{differential } (8 + 2\,\text{B})
\end{aligned}$$

It is assumed that all data are sent with the same frequency f_{up} in individual packets, meaning that the TCP and IPv6 headers must be added for each sensor. For the *High* case with an update frequency $f_{\text{up}} = 20\,\text{Hz}$ the needed data rate is calculated as:

$$r = 8 \cdot (n_{\text{packets}} \cdot (s_{\text{TCP/IP-header}} + s_{\text{protocol-header}}) + s_{\text{GPS}} + s_{\text{mag}} + s_{\text{diff}}) \cdot f_{\text{up}}$$

Where s_* denotes a size in bytes of packet content or header. n_{packets} is the number of packets sent within every period. In this case three packets are sent, therefore $n_{\text{packets}} = 3$. The multiplication by 8 is to convert from bytes to bits. Inserting the numbers results in the following data rate:

$$r = 8 \cdot (3 \cdot (60 + 2) + 16 + 8 + 8) \cdot 20 = 34.88\,\text{kbit/s}$$

This calculation is additionally calculated for $f_{\text{up}} = 1$ and $f_{\text{up}} = 10$. The results are shown in table 7.16 alongside with the results of the calculation for the *Medium* and *Low* case.

Case	**data rate @ 1 Hz [kbit/s]**	**data rate @ 10 Hz [kbit/s]**	**data rate @ 20 Hz [kbit/s]**
low	0.624	6.24	12.48
medium	1.184	11.84	23.68
high	1.744	17.44	34.88

Table 7.16. Data rates for the three scenarios with three different update frequencies.

The calculated data loads shows that for low update frequencies even the satellite phone internet connection can provide the necessary data rate. In addition it is seen that a

GPRS connection should be enough in all the mentioned combinations of packages and update frequencies.

7.2.2 Adding waypoints to the ALTAR

To add a waypoint a package of 18 bytes is send to the ALTAR and a response of 3 bytes is send back to the MCC. Therefore the total data is 141 bytes when a TCP IPv6 header is used for response and request. The time consumption for adding a waypoint is considered for the three different wireless links and shown in table. The typical data rates listed in table 7.15 are used.

Technology	Time consumption [s]
GPRS	2.52 ms
Internet via satellite phone	14.69 ms
WiFi IEEE 802.11a	5.64 μs

Table 7.17. Theoretical time consumption for adding a waypoint using different technologies.

This shows that independent of which of the three mentioned connection types is chosen, the needed time for sending a waypoint to the ALTAR from the MCC is very small compared to the time used by the ALTAR to drive from one waypoint to another.

7.2.3 Driving the ALTAR externally

Driving the ALTAR from the MCC is possible by sending a set differential packages to the ALTAR. Each package send requires 133 bytes when a TCP IPv6 header (request: 10 bytes + response: 3 bytes + TCPheader: $2 \cdot 60$ bytes). It is assumed that the ALTAR can be driven by setting the velocity with a frequency of 10 Hz. With this frequency the needed data rate is calculated to be 10.64 kbit/s. Hence driving via GPRS connection is a possible solution.

7.2.4 Assessments

As shown in the examples above only a very slow connection is needed to maneuver the ALTAR from one waypoint to another. If information must be returned to the MCC periodically a faster connection is needed, however in most cases a GPRS connection is more than enough. It is assessed that the protocol reduces the data load per package to a minimum, and as shown most of the data is created by the headers of the TCP and IPv6. If the TCP protocol should be used it could be beneficial to combine multiple packages in one TCP package only requiring one header.

8 Acceptance Test

This worksheet contains the results from the acceptance test conducted to determine the performance of the implemented controller structure. First the tests are described and afterwards the results are presented.

The acceptance test is performed on the ALTAR system, consisting of the RobuROC4 fitted with the external sensors and with the designed controller structure implemented. The tests are carried out to test the specific requirements described in the requirement specification 3.3. The requirements set for the communication protocol are judged while carrying out these 2 tests. The tests are carried out according to the procedure described in appendix B.

8.1 Test Descriptions

8.1.1 Test 1

This test determines whether the ALTAR works as an autonomous system, capable of navigating from GPS waypoints and communicate through WiFi. In requirement **r.1** and **r.2** it is specified, that the RobuROC4 must be able to navigate between waypoints, and stop within a maximum distance of 55 mm from the end point. The test also deals with requirement **c.1** regarding adding line segments, and requirement **c.2** and **c.3** regarding output from the sensors mounted in and on the RobuROC4 and the periodic exchange of these outputs.

The test is performed by defining 5 waypoints that tests different turning angles during the run. The test determines if the ALTAR is capable of steering onto, and follow a line segment while fulfilling the above mentioned requirements. The orthogonal deviation from the followed line segments is observed to determine precision, from the data output of the GPS module.

8.1.2 Test 2

This test determines whether the robot complies with requirement **r.3** specifying that the wheels at no time must drive in opposite direction of each other. This requirement is tested by setting a line segment of two GPS waypoints and place the ALTAR heading in opposite direction of the end point. This will force the robot to make a 180° turn, thus the maximum allowed angular velocity is applied. By visual observation, it is determined whether any of the wheels are running in reverse while the test is carried out. Additionally this test determines if the system complies with the second part of requirement **c.1** regarding

removal of line segments, as the last given waypoint from test 1 will still be saved in the RobuROC4. If the *clear waypoints* command message does not work, the ALTAR will start to drive on a line segment defined by the old waypoint and the first added for test 2.

8.2 Test Results

8.2.1 Test 1

A laptop is used as MCC during the tests. A subscription to periodically receive data from the GPS was sent wireless to the RobuROC4, for a rate of position status updates at 10 Hz. The vehicle was manually driven to 5 locations on the parking lot in accordance to the description in appendix B. The MCC was used to log the coordinate of each of these 5 locations by saving the last received GPS status from robot at each location. The 5 coordinates are marked with **x** in figure 8.1. This fulfils requirement **c.3**, as the robot returned the sensor data periodically as intended. It also partly fulfils requirement **c.2**, since data from an external mounted sensor was returned. No sensor data from internal sensors was sent during the test, hence it can only be considered partly fulfilled. The MCC was then used to send the 5 waypoints to the robot, which verifies the first part of requirement **c.1**. Figure 8.1 shows how the robot navigated to the 5 waypoints. The blue curve shows where the controller aimed and the red shows the the actual path which the vehicle was driving.

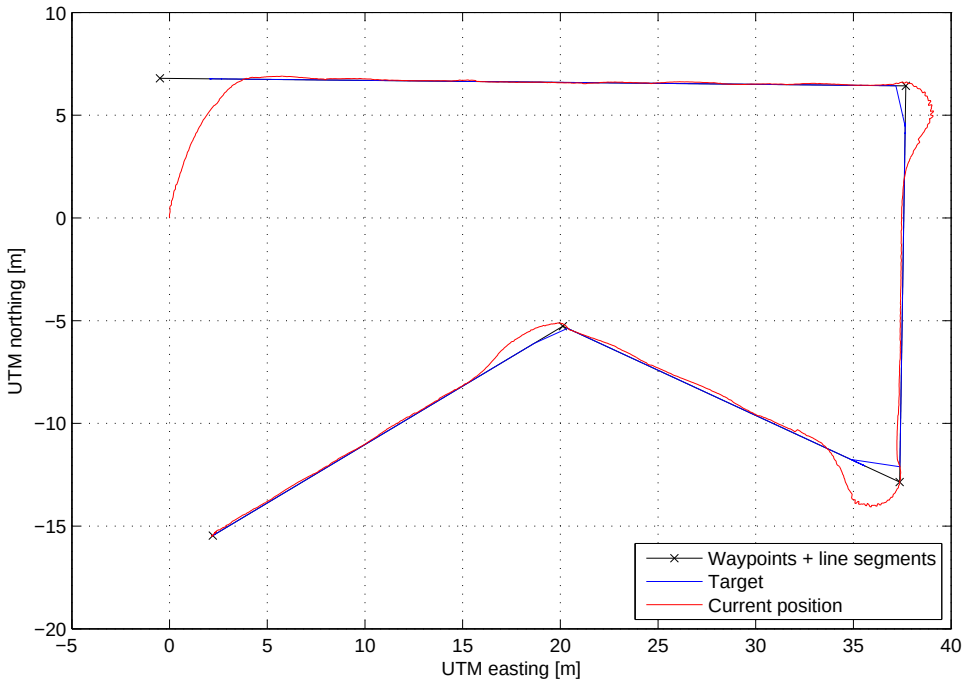


Figure 8.1. Path of the vehicle during test 1.

The robot's deviation from the line segment for test 1 is shown in figure 8.2. This distance is measured as the orthogonal distance from the robot's position to the line segment. The 3 peaks around 38 s, 53 s, and 73 s are caused by the change of line segment.

The interesting part is when the robot moves between two waypoints and follows the line segment. This is shown in figure 8.3 for the case where the robot moves from the third waypoint to the fourth waypoint, which is in the time interval from 57 s to 68 s. This is the worst case regarding line tracking, which is also seen from figure 8.2. According to requirement **r.1** the robot must stay within a distance of 55 mm of the line segment. This requirement is not fulfilled, since the deviation peaks at 230 mm.

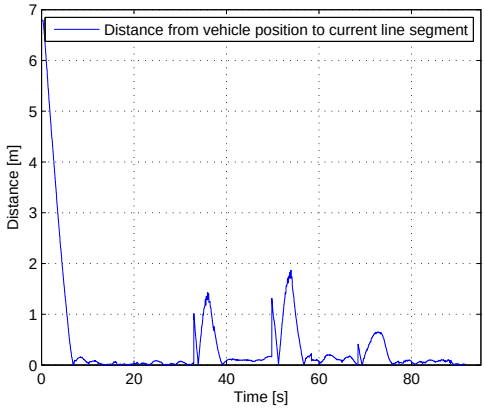


Figure 8.2. Distance from the vehicle's position to the line segment that is followed.

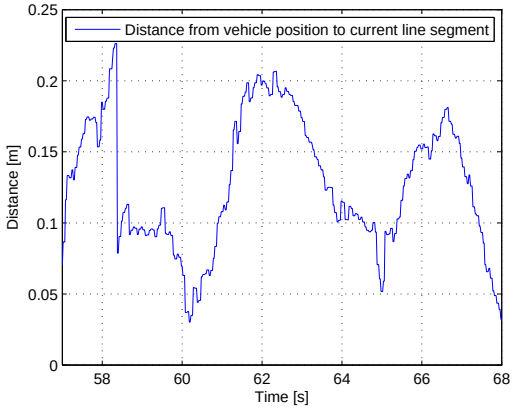


Figure 8.3. Close up of distance plot in figure 8.2 between third and fourth waypoint.

Figure 8.4. Distance from the vehicle's position to the end point at the end of test 1.

Figure 8.4 is used to determine whether requirement **r.2** can be complied with. The figure shows a close up of the distance from the RobuROC4 to the line segment end point in the last two seconds of the run. From this figure it is seen that the distance during the last second of the run stays below 25 mm and the robot stops 10 mm from its final end point. This means that requirement **r.2** can be complied with since it stops within the limit of 55 mm of the end point.

8.2.2 Test 2

Before the waypoints for test 2 were sent, the route from test 1 was deleted. This was done wirelessly from the MCC using the *clearWaypoints* command. This verifies the second part of requirement **c.1**, meaning that this requirement is fulfilled. The travelled path of the robot during test 2 is shown in figure 8.5.

From this test it was verified by visual inspection, that requirement **r.3** is fulfilled since none of the wheels turned in the opposite direction of each other during the turning.

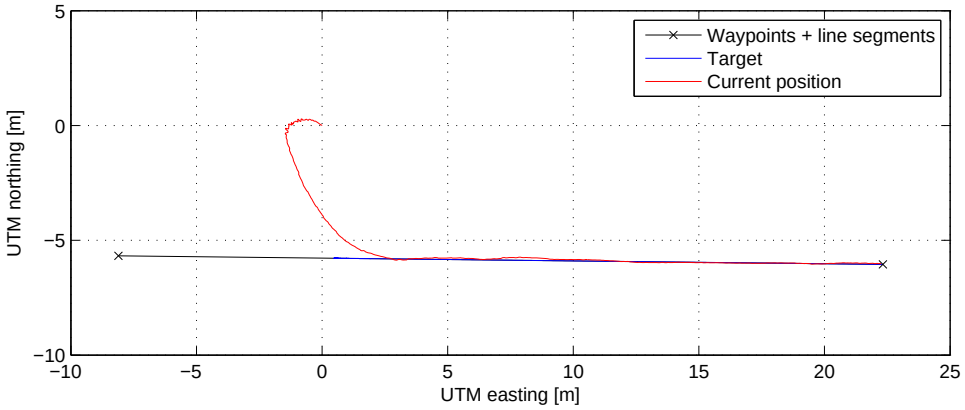


Figure 8.5. Trace of the vehicle during test 2.

The distance from robot to the line segment for test 2 is shown in figure 8.6. A close up is shown in figure 8.7. Special attention should be paid to the path of the vehicle close to its initial position in figure 8.5. The vehicle seems to move northward briefly and then turn around and steer towards the line segment. This is caused by the placement of the GPS antenna. It is not placed in the centre of the vehicle, but as described in section 2.1.5, it is placed in the left rear corner of the vehicle. Figure 8.8 shows a close up of the distance from the ALTAR to the line segment end point in the last second of the run. As seen in test 1 the distance stays below the distance specified in requirement **r.2**.

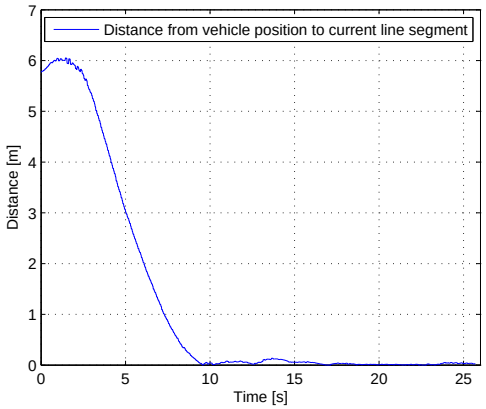


Figure 8.6. Distance from the vehicle's position to the line segment that is followed.

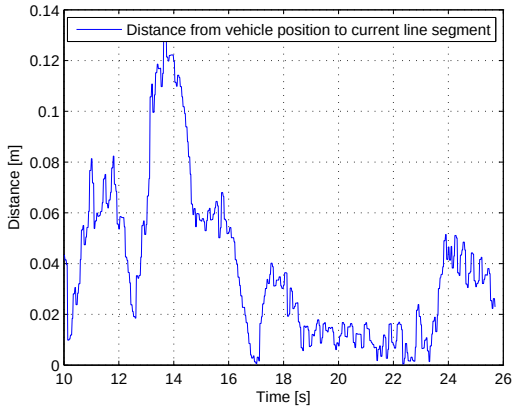


Figure 8.7. Close up of figure 8.6.

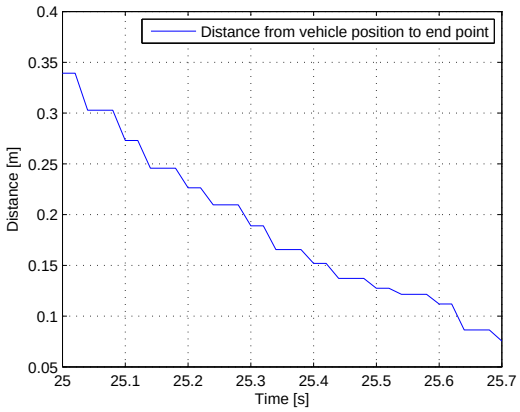


Figure 8.8. Distance from vehicle position to the end point at the end of test 2.

The results of the test, in terms of fulfilling the requirements are shown in table 8.1.

Requirement	short description	status
r.1	Deviate less than 55 mm from line segment.	Not fulfilled
r.2	Drive to waypoint and stop within 55 mm of it.	Fulfilled
r.3	Wheels must not turn opposite of each other.	Fulfilled
c.1	Send, add and remove waypoints using WiFi.	Partly fulfilled
c.2	Get data from internal and external mounted sensors.	Partly fulfilled
c.3	Return data periodic.	Fulfilled

Table 8.1. Status on requirement after tests.

8.3 Assessments

From the tests conducted it is seen that the controller is not able to track line segments with a deviation less than 55 mm. Several factors could have effected the outcome of the tests. Those that are considered to be significant are described here. The readings from the magnetometer and the position of the GPS antenna are analysed.

Figure 8.9 shows the measured heading, the reference angle, and the angle error. A close up of the measured heading is shown in 8.10. From the close up it is apparent that the calculated heading contains fluctuations of up to 12 ° with a period of 3 s. This is caused by the automatic degaussing of the magnetic field sensors in the magnetometer. The effect of this is a large sudden error angle, and the vehicle turns to try reducing the error angle caused by the faulty reading. As seen in figure 8.10 the result of these spikes is that the vehicle turns away from the reference by up to 2 °. This might explain some of the deviation from the line segment since the vehicle turns away from the line segment.

Figure 8.11 shows calibrated magnetic field data from the magnetometer. The raw data output of the magnetometer is calibrated with parameters from table 6.6. From the close up in figure 8.12 it is seen how the degaussing effects the measurements when each axis of the magnetometer are degaussed independently. When comparing figure 8.10 and 8.12 it is seen that the heading mainly is affected by the degaussing on the field

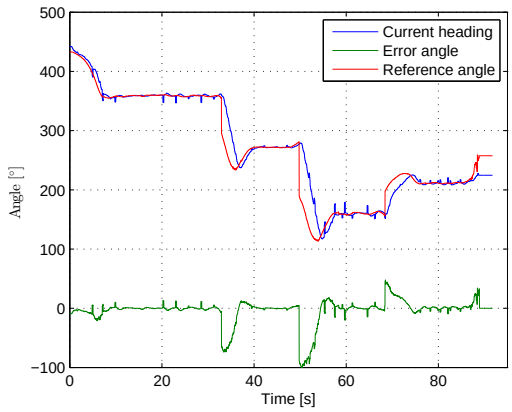


Figure 8.9. Heading of the vehicle and the error angle fed to the controller.

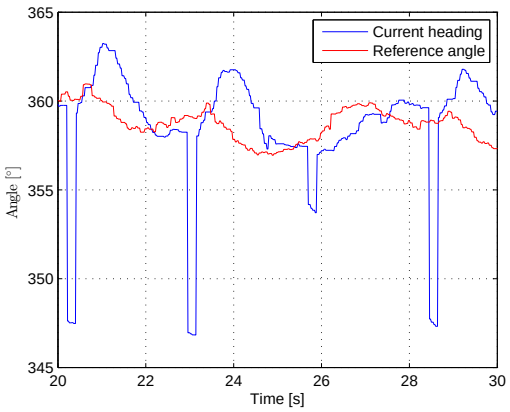


Figure 8.10. Close up of the measured heading and the reference heading.

in the x-direction. This is a special case because the x-component is around 0 and the resulting vector only points in the y-axis. The heading is the argument of the resulting vector that describes the magnetic flux density. Therefore degaussing on y-component will only scale the length of the vector in this case and not change the argument of the vector. Degaussing on the x-component will however alter the direction of the vector and corrupt the measurement. This may become more apparent from figure 6.5. However in the more general case, the heading is effected by degaussing in both the x-component and y-component.

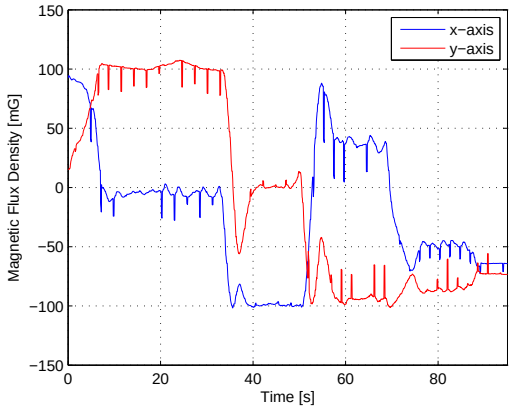


Figure 8.11. Raw data output from the magnetometer shown for two axis.

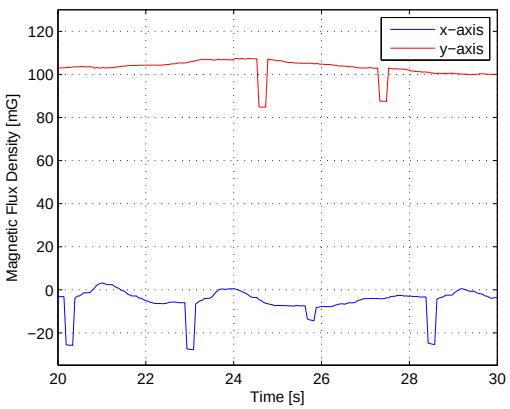


Figure 8.12. Close up of the raw data output from magnetometer.

Another thing that might cause deviation from the line segment, is the placement of the antenna. It is placed in the rear left corner of the vehicle as described in section 2.1.5. From the overshoot present after each turn, it can be concluded that the distance at which the robot shifts from one line segment to another should be increased. Some of this error can however be explained by the fact that the GPS is mounted in the corner of the RobuROC4. The turning distance then appears to change from the expected 0.34 m to approximately twice that during right turns or approximately 0 m when a left turn is performed.

9 Conclusion and Perspectives

9.1 Conclusion

This project was inspired by the ASETA project, which suggests the use of autonomous vehicles for agricultural purposes. The vehicle should be able to drive in a beet field and handle weed treatment. During this project it was chosen to focus on designing and implementing a controller structure that allowes a vehicle to drive in a field following line segments defined by waypoints. Since the field contains crops of a value to the farmer, the vehicle should drive in a way that limited crop damage. Furthermore it was chosen to use wireless communication to control the vehicle remotely. Therefore a communication protocol was designed.

The provided vehicle for the task was a Robosoft RobuROC4. To determine position and heading of the vehicle an Ashtech MB-100 GPS and an OceanServer OS5000-US magnetometer was provided.

A requirement specification was made to specify the needed functionality and precision of the system. A precision requirement was based on an analysis of the dimensions on a beet field combined with the dimensions of the used vehicle. This combination stated that the vehicle was allowed to diverge from a straight line by no more than 55 mm. To reduce damage done to the crops when turning, a requirement stated that the right and left wheel pairs at no time was allowed to drive opposite of each other. For the communication protocol, it was required that the vehicle could be controlled externally and that the sensor data from the vehicle could be returned.

A model of the RobuROC4 was derived and used to develop a controller structure and controllers. A kinematic model was derived by analysing the physical construction of the vehicle and the dynamics of the vehicle was determined conducting tests.

A controller structure was designed, based on how a "good helmsman" would steer a boat onto a straight line. It was chosen to generate aiming point on the straight line and a reference angle is calculated on basis of the aiming point and the current position of the vehicle. In the controller structure the angular velocity is decoupled from the linear velocity, by decreasing the linear velocity, since the angular velocity is considered the most important. In addition, the angular velocity was reduced when needed to comply with the requirement regarding opposite turning direction on the wheel pairs.

It turned out that the provided magnetometer was very sensitive to physical shocks because the build-in filtering of the heading uses accelerometers to compensate for variations in pitch and roll. Since the magnetometer was mounted on a plastic tube, it was subjected to sudden movements. It was therefore chosen to calculate the vehicles heading manually from the raw data from the magnetic field sensors in the magnetometer. These data were however influenced by noise from internal degaussing on each of the sensor

axis. The position was determined by using the GPS position directly.

A software structure was developed and implemented using Robot Operating System (ROS). The software was divided into separate ROS nodes that handled specific tasks. Three nodes were used to handle the connections to the RobuROC4, magnetometer, and GPS. One node acted as server and handled incoming requests via the wireless connection and the last two nodes contained planning and controlling tasks.

To be able to control the vehicle externally, a communication protocol was designed. Command messages were designed, to make it possible for a client to control the robot by managing waypoints, setting control mode, or by setting velocities directly. Status messages were designed to send sensor data back to the client periodically. Finally error messages were designed to report errors in the ALTAR to the MCC.

An acceptance test was carried out to verify if the requirements, set in the requirement specification, could be fulfilled. This was divided into two subtests. In the first test, five waypoints were sent to the robot from a laptop acting as Mission Control Center (MCC). The robot was then able to follow the line segments between these waypoints and stopped 7 mm from the last waypoint. This fulfilled the demand that robot must be able to stop within 55 mm of the last waypoint. During the first test the ALTAR deviated up to 0.23 m from the line segment when following the line segment, and up to 2 m in connection with change of line segment. These deviations imply, that the requirement of a maximum deviation of 55 mm could not be fulfilled. In the second test the vehicle was faced away from the end point of a single line segment. The vehicle then turned around and turned onto the line segment without having the left and right wheel pair turning in opposite direction of each other. This test showed that the system fulfills the requirement regarding opposite wheel velocities.

The two subtests in the acceptance test also tested the 3 requirements set for the protocol. The first of these requirements was fulfilled, since the robot received the waypoints using a wireless connection. It was also possible to clear the waypoints from the first subtest before the waypoints of the second test were sent. The second requirement to the protocol was considered partly fulfilled, since only data from one external sensor was tested. It was not tested if the ALTAR was able to send data from the internal sensors, why the requirement could not be fully assessed. The last requirement was tested when the waypoints for the two tests were created. The robot was setup to return GPS sensor data with a rate of 10 Hz and these were received successfully. The last requirement is therefore considered fulfilled.

Even though the acceptance test did not show the required precision, the derived controller structure is considered useful since the overall functionality of tracking straight lines is obtained. As stated in the assessments of the acceptance test, the magnetometer degauss is assumed to be the main reason for the missing precision. Every time an axis is degaussed the heading is affected. This causes the ALTAR to turn away from the line segment and cause a larger orthogonal error.

Assessing the ALTAR in correlation with the original problem statement of designing and implementing software that could navigate the RobuROC4 via waypoints the goal has been accomplished. Although the needed precision of driving in terms of crop damage has not been achieved, the controller structure has still proven functional. Crop damage has been taken into consideration while developing the controller and different precautions has been made. This includes limiting the rotation and turning towards the next line segment at a distance before a waypoint has been reached.

The developed protocol keeps the protocol overhead to a minimum and thereby reducing the needed bandwidth of the communication between the ALTAR and the MCC. Although all the functionalities of the protocol was not tested during the acceptance test, the protocol it is expected to work since it has been tested in piece by piece during development.

The acceptance test has shown that to improve the product even further sensor readings should be considered first. If magnetometer readings are improved it is possible that all the requirements could be complied with.

9.2 Perspectives

The ALTAR has proven functional in terms of following a route made of line segments. The precision is not as good hoped for, but improvements in terms of sensor readings could be the solution to this problem. The calculated heading from the raw magnetic field has proven disturbed by degaussing of sensors. By compensating for this degauss it is assessed that the overall precision of the system could be improved significantly.

Another possible improvement could be to take the physical placement of the GPS antenna into consideration. During this project the fact that it is placed in the corner of the vehicle has not been compensated for in any way. The main reason for this was the lack of a reliable heading while driving. By implementing a filter removing the degauss noise, the position of the ALTAR could be translated into the centre of the vehicle to give a correct error distance to line segment.

At this time the magnetometer calibration has to be performed by hand every time the surroundings are changed. Since this is a cumbersome task to do, a calibration algorithm has to be implemented before this product can be used effectively.

The ALTAR is intended to be part of the ASETA project. The project should result in site-specific weed management carried out by autonomous robots. This can have several advantages. It might reduce the amount of chemicals used on weed removal. This is an economical advantage for the farmers and the environment, which will be subjected to less chemicals. Since the robots are autonomous another beneficial factor is that the farmer will need less employees to run the farm.

But why limit the use of autonomous robots to agricultural purposes. Robots steered by waypoints can serve many purposes. An example of an area where robots like the ALTAR could be used is within surveillance of large areas. In some cases it is inconveniently to place stationary cameras to cover a large area. The robots like the ALTAR could be used to patrol or investigates areas where an alarm has been started.

In factories the robot could be used for transportation of equipment and spare parts in the production area. Since the GPS coverage is reduced indoor the relative position within the factory should be determined by other means.

Acronyms

ASETA Adaptive Surveying and Early treatment of crops with a Team of Autonomous vehicles

ALTAR Autonomous Line Tracking Agricultural Robot

COM Centre Of Mass

COV Centre Of Vehicle

GPS Global Positioning System

GPRS General Packet Radio Service

ICR Instantaneous Centre of Rotation

MCC Mission Control Center

MIMO Multiple Input Multiple Output

NMEA National Marine Electronics Association

ROS Robot Operating System

RTK Real Time Kinematic

SISO Single Input Single Output

UA Unmaned Aircraft

UDP User Datagram Protocol

UGV Unmanned Ground Vehicle

UMTS Universal Mobile Telecommunications System

UTM Universal Transverse Mercator

WiFi Wireless Fidelity

Bibliography

[1] W. Kazmi, M. Bisgaard, F. Garcia-Ruiz, K. D. Hansen, and A. la Cour-Harbo, "Adaptive surveying and early treatment of crops with a team of autonomous vehicles," in *European Conference on Mobile Robots 2011*, 2011.

[2] ROBOSOFT, *robuROC4 The Powerful Outdoor Mobile for Civil and Military Applications*. [Online]. Available: http://www.robosoft.com/eng/sous_categorie.php?id=1027

[3] sukkerroer.nu, "Optimal saaning," November 2011. [Online]. Available: http://www.sukkerroer.nu/roedyrkning/retningslinier-og-anbefalinger/optimal-saaning/

[4] R. J. Putman, *Mammals as Pests*. Chapman and Halls, 1989, p. 38.

[5] GPS.gov, "Gps accuracy." [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/

[6] K. Kozłowski and D. A. Pazderski, "Modeling and control of a 4-wheel skid-steering mobile robot," *Control*, vol. 14, no. 4, pp. 477–496, 2004.

[7] K. Y. Pettersen and E. Lefeber, "Way-point tracking control of ships," in *Proceedings of the 40th IEEE Conference on Decision and Control, 2001*, Orlando, Florida USA, December 2001, pp. 940–945 vol.1.

[8] ROS.org, "Robot operating system," Nov 2011. [Online]. Available: http://www.ros.org/wiki/

[9] C. Konvalin, "Compensating for tilt, hard-iron, and soft-iron effects," 2009. [Online]. Available: http://www.sensorsmag.com/sensors/motion-velocity-displacement/compensating-tilt-hard-iron-and-soft-iron-effects-6475

[10] Boost.org, "Boost serialization library," Novmemer 2004. [Online]. Available: http://www.boost.org/doc/libs/1_48_0/libs/serialization/doc/index.html

[11] Iridium, "Iridium openport broadband ad," February 2008. [Online]. Available: http://www.iridium.com/DownloadAttachment.aspx?attachmentID=765

[12] J. J. Parson and D. Oja, *New Perspectives on Computer Concepts*. Course Technology, Cengage Learning, 2011.

[13] Ieee 802.11a. Radio-Electronics.com. Accessed: December 16th 2011. [Online]. Available: http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11a.php

Appendix

A Dynamic Model Parameter Estimation

This measurement journal describes the process of estimating the parameters of the skid steering RobuROC4 vehicle. A description of the robot is found in the datasheet [9]. In this measurement the accelerations and velocities, both linear and angular, must be determined.

A.1 Test setup and measurement equipment

The measurement is carried out using the Vicon Motion equipment in room C2-111. The measurements are sampled with 100 Hz. The equipment is capable of saving current position and angles of the subject under test. Afterwards the velocities are calculated from the data by forward difference.

A.2 Test procedure

The test is carried by executing three subtests, two for testing the linear velocity and linear acceleration, and one for testing the angular velocity and angular acceleration.

The test for the linear velocity needs two test due to lack of space in the motion tracking room. In the first linear velocity test the RobuROC4 is placed close to one wall in the motion tracking room with the front pointing against the opposite wall, and is thereafter asked to drive at maximum linear velocity (2 m/s) for 1.5 s, and angular velocity of 0 rad/s. This first test will only reveal the accelerating and decelerating behaviour because the maximum velocity in not reached. To test the maximum linear velocity of the RobuROC4 the second linear velocity test is carried out. Here the RobuROC4 is placed in one corner of the motion tracking room, with the front pointing towards the opposite corner. This setup makes it possible to ask the RobuROC4 to drive at maximum linear velocity (2 m/s) in 2 s. This test will only reveal the maximum linear velocity because the corners of the room is not covered by motion tracking system, and the accelerations are not measured here.

To determine the angular accelerations and maximum angular velocity, only one test must be carried out. The RobuROC4 is placed in the centre of the motion tracking and is set to rotate with maximum linear velocity (5 rad/s) and with linear velocity of 0 m/s for 3 s which is enough to obtain the maximum angular velocity.

A.3 Results

The data logged from the tests are saved in the following files:

- Linear velocity test 1:
  /MATLAB_files/dynamic_model_generation/linear_velocity_measurement_1.mat
- Linear velocity test 2:
  /MATLAB_files/dynamic_model_generation/linear_velocity_measurement_2.mat
- Angular velocity test:
  /MATLAB_files/dynamic_model_generation/angular_velocity_measurement.mat

In the following two sections the data from the three tests are processed and the maximum velocities and accelerations are determined.

Linear velocity

The linear velocities measured in the first test is shown in figure A.1. As seen in the figure it is not possible to determine the maximum velocity.

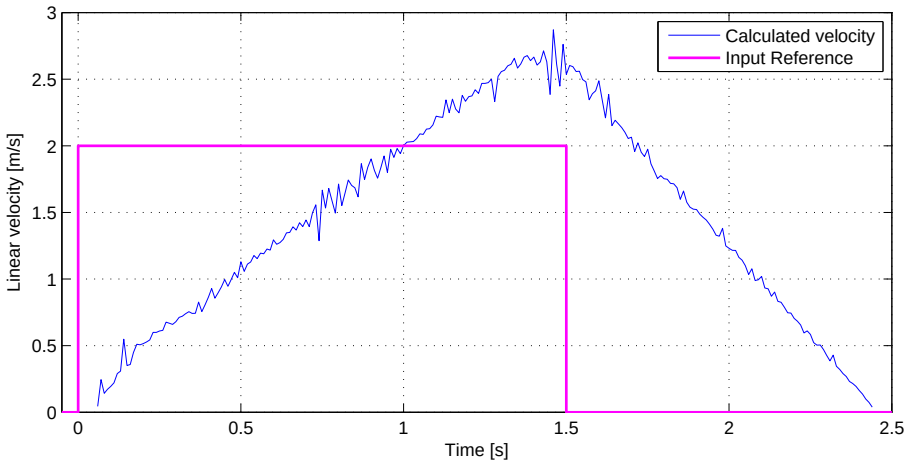


Figure A.1. Linear velocity of RobuROC4 (first run).

The linear velocities measured in the second test is shown in figure A.2. The reason why the velocity is only shown for a period of 1.8 s is that the tracking system was not able to capture the motion of the robot when it was in the corners of the room.

In figure A.3 the combination of the two runs of linear velocity is shown. The combination is made to visualise a run where both the acceleration behaviour and the maximum velocity can be determined. From figure A.2 the linear acceleration and maximum linear velocity can be extracted.

Angular velocity

The measured angular velocity is shown in figure A.4. From this figure the angular acceleration and maximum angular velocity is extracted.

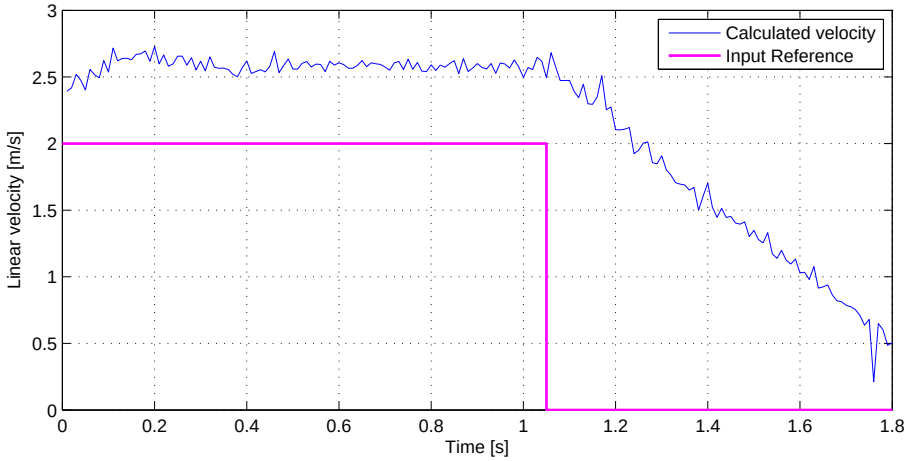


Figure A.2. Linear velocity of RobuROC4 (second run).

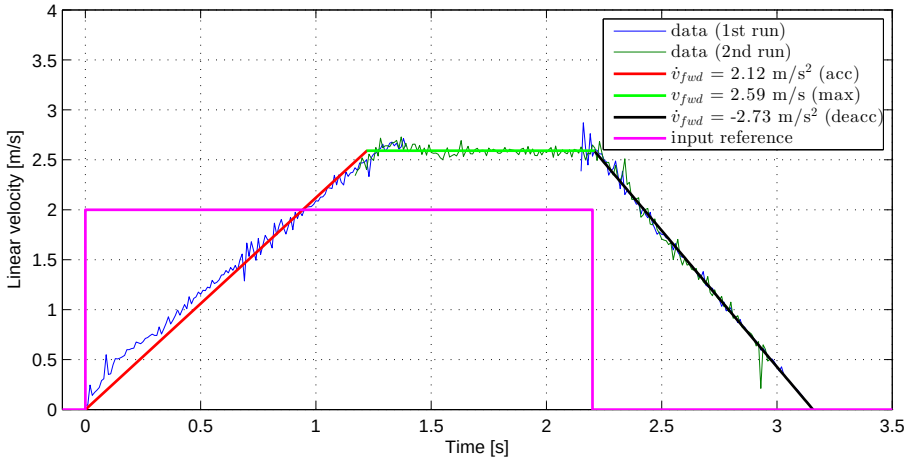


Figure A.3. Linear velocity of RobuROC4 (combined run).

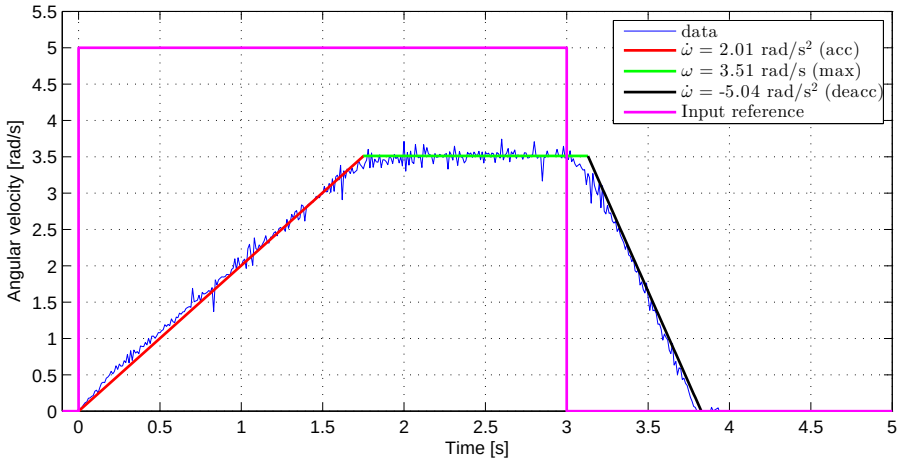


Figure A.4. Angular velocity of RobuROC4.

B Acceptance Test Journal

B.1 Formalities

This test journal documents the procedure and execution of the acceptance test performed on the Autonomous Line Tracking Agricultural Robot (ALTAR) developed through this project.

Date: 09-12-2011

Participants:

Niels H. Andersen

Mikkel U. Kajgaard

Rune Madsen

Jesper Mogensen

Anders Wittendorff

B.2 Purpose

The purpose of the acceptance test is to determine if the system meets the requirements specified in the requirement specification in worksheet 3.

B.3 Test Object

The test object is the ALTAR described in worksheet 2. The designed software structure, including the designed controller structure, is running on the laptop specified in table 2.2.

B.4 Test Setup

The test setup for the acceptance test is shown in figure B.1. The tests are carried out in a parking lot on tarmac. A base station for the RTK GPS setup is included. This base station is placed in a distance of about 100 m to the parking lot where the test is carried out. A laptop is set up acting as a dummy MCC, and is connected directly to the ALTAR via WiFi. The source code for the dummy MCC is found on the cd `/software/dummy_MCC`.

The position of the ALTAR is determined by the GPS attached to the RobuROC4. This GPS position is used since this is the most precise method to determine the position of the vehicle. The position of the RobuROC4 is logged by saving the internal controller parameters of the ControllerNode described in worksheet 6. By logging the internal controller parameters the GPS coordinates are already converted to UTM coordinates.

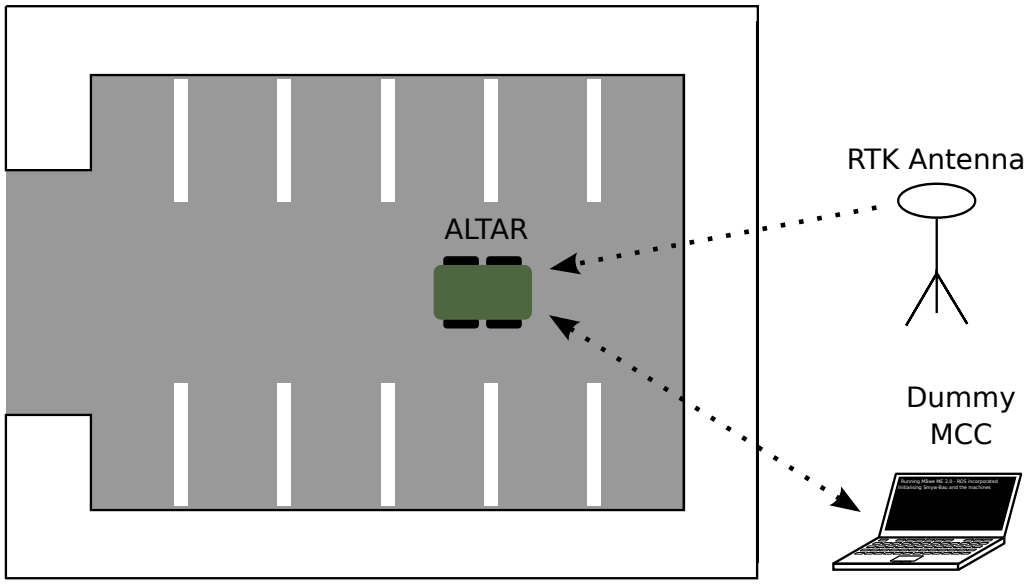


Figure B.1. Setup for the acceptance tests

B.5 Used Equipment

- HP Pavilion g6 laptop used as dummy MCC .
- Ashtec MB-100 GPS in an RTK-Setup

B.6 Test Procedure

Initial

Following points are initiated before both tests are carried out.

1. The ALTAR and dummy MCC are powered on.
2. The software is started on the ALTAR by using the launch file `/software/ROS RobuROC4/RobuROC4.launch`.
3. The dummy MCC software is executed on the dummy MCC.
4. The magnetometer is calibrated by setting the ALTAR in control mode 3 (via the dummy MCC) and driving it manually in a straight line until the software on the ALTAR respond that the sensor is calibrated.

Test 1

1. The ALTAR is set to control mode 2 (via the dummy MCC) and manually driven to 5 positions according to the ones shown in figure B.2. These positions are logged in the dummy MCC one at a time.
2. The ALTAR is driven to the start position as shown in figure B.2, and set to control mode 1 (via the dummy MCC).
3. The logged positions are sent to the ALTAR as waypoints (this makes the robot start driving).
4. When the ALTAR has reached the last waypoint the waypoints are cleared (via the dummy MCC).

Test 2

1. The ALTAR is set to control mode 2 and manually driven to the 2 positions shown in figure B.3. These are logged in the dummy MCC one at a time.
2. The ALTAR is manually driven to the start position as shown in figure B.3 and set to control mode 1.
3. The logged positions are sent to the ALTAR as waypoints (this makes the robot start driving).
4. It is noticed whether the right or left wheel sets turns in opposite direction at any time during the run.
5. When the ALTAR has reached the last waypoint the waypoints are cleared (via the dummy MCC).

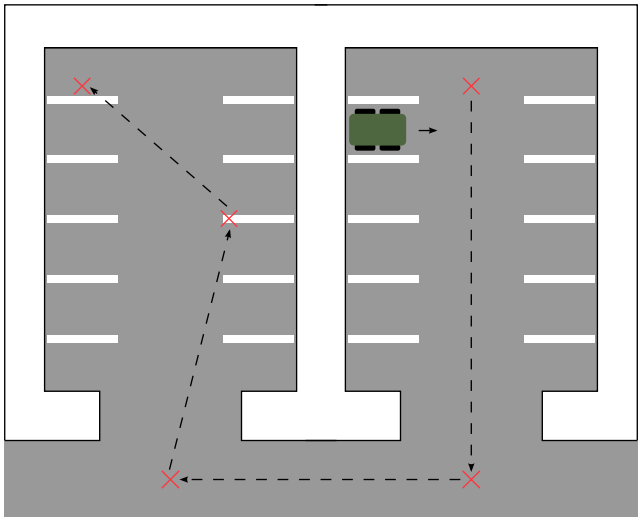


Figure B.2. Route for test 1 (Waypoints are indicated by red crosses).

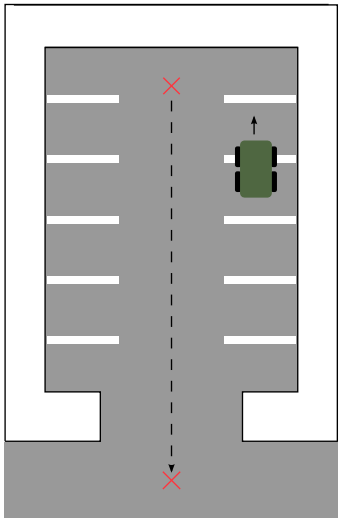


Figure B.3. Route for test 2.

B.7 Results

The logged data the tests are saved, and is found on the cd `/MATLAB_files/acceptance_test`. No opposite direction of the right and left wheel pairs was experienced during the two tests.

B.8 Uncertainties

During the tests, some uncertainties are considered important when judging the results of the test.

- The external sensors are as mentioned in provided hardware 2.1.5 mounted on a plastic tube, which is observed to have movement when the robot is driving,

especially when accelerating and braking. This is likely to cause fluctuations on the obtained data from the GPS

- The GPS has an accuracy of 1 cm + 1 ppm according to the datasheet for the GPS ◎. Having a distance to the RTK Base station antenna of 100 m gives an accuracy of 1.01 cm