Autonomous Road Striper

CA2 Project

31-05-2012

Aalborg University

12gr832 - 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:
Autonomous Road Striper

Subject:
Multivariable Control Systems

Project period:
February 1^{st} - May 31^{st}, 2012

Group:
Group 832

Group members:
Niels H. Andersen
Rune Madsen
Maxime Nollet
Alexandre Slove

Supervisors:
Henrik Schiøler
Karl Damkjær Hansen

Copies: 7

Page count: 115

Attachments: CD

Appendix: 3

Completion: 31^{st} May 2012

Synopsis:

The aim of this project is to design a road striping robot. This robot should be able to both map and stripe an unstriped road in an autonomous way. Requirements regarding precision and velocity of the process have been set.
A model of the robot has been deduced including both the kinematic and the dynamic behaviour of the robot. Based on this model a controller has then been designed to allow the robot to track a predefined trajectory. Pictures of the road taken from a camera are processed to detect the edge of the road and generate a trajectory that the robot must follow.
An acceptance test has been performed to verify if the robot fulfils the stated requirements. The performed test shows that the robot is able to both map a road and follow a trajectory even though the requirements are not fulfilled due to lack of precision.

Preface

This report is the result of the project produced by the group 832 during the 2nd semester of the Master programme in Control and Automation at Aalborg University during the spring 2012. The objective of this semester project is to set up a multivariable control system. In this scope, the project conducted during this semester is the implementation of an autonomous vehicle able to stripe a road.

The report is divided in chapters, sections, and subsections which are numbered. For example, chapter 7, section 1, subsection 2 is numbered 7.1.2. In the report, several acronyms are used which are indexed at the end of the report.

Some content of the report refers to datasheets of used hardware in the project, or implementation code which can be found on the CD attached to this report. The path to these files are given in the following way: `/folder_name/file_name.ext`. This CD also contains the MATLAB® files used to generate the figures contained in the report as well as the data used to generate these. In order to run these files the script folder contained on the CD `/matlab_files/scripts` must be added to the MATLAB® path. Several videos are captured and generated during this project to document performance of the implemented system. These videos are also found on the CD `/videos/`.

Niels Hyltoft Andersen

Rune Madsen

Maxime Nollet

Alexandre Slove

Contents

1 Introduction

This project treats the topic of painting road stripes on public highways and motorways. Generally road stripes have the purpose of enhancing safety for drivers, bikers and pedestrians. This is done by guiding the traffic e.g. by using special stripe types when road lanes are intertwined or overview of the road is bad due to hills or turns. Since the stripes must contribute to road safety it is essential that these stripes are painted clearly and accurately. In Denmark the specific rules about how and where these stripes should be placed are dictated by The Danish Road Directorate [1].

When road stripes have to be painted on a new road, the task is divided into two distinct sub tasks. The first one is to pre-mark the road by applying a temporary marking on the road to show exactly where the stripes should be placed [2][3]. In cases where the task at hand is to repaint fainted old stripes the pre-marking can be omitted since the old stripes can be used as guides. The second part is to follow these pre-markings and apply the paint which draws the road stripes. Traditionally these two tasks are performed by two different machines each controlled by at least one operator [4].

To reduce the cost of striping roads, it is suggested to automate the process. This could decrease the number of workers needed and eventually also increase the speed of the striping process. One of the major problems of automating the process is that the precise location of the road is not necessarily known. The lack of information could either be because no precise information is available about old roads which must be re-striped or because a newly laid road diverges from the planned.

In the traditional way of performing road striping, workers could compensate for the incomplete information by visually determining where the road actually is and does therefore not depend on precise information about the road. To be able to automate the process of road striping this property must be transferred to an automated process performed by a robot.

This project will work towards automating the process of striping roads by progressing from the following problem statement:

How can the process of road striping be automated using an autonomous mobile robot without having precise information about the road on beforehand.

Part I

Problem Analysis and Product Specification

2 Analysis

The following chapter gives a description of different existing solutions and regulations of road striping, and also focuses on the approaches that could be used to automate the process. The procedure used today for road striping will be investigated and altered to fit an automated approach of performing the task. Similarly existing solutions for detecting roads, will be looked into in order to allow a visual detection of the road. In the end a delimitation will narrow down the project to describe exactly what will be the goal of this project.

2.1 Existing Solutions and Regulations

2.1.1 Existing Solutions

As explained in the introduction, striping new roads is performed in two steps. The first step is to place thin pre-marks in order to indicate the placement of the final stripe. The second step is to use the pre-marks on the road to be used as guides to paint the final road stripes. When an old road is re-striped, the first task is skipped and the old stripes are used as the reference for the new stripes. This section deals with the road stripe techniques used nowadays and introduces the type of machines involved. Three kind of machines will be distinguished, and for each of these, the processes of pre-marking and final painting will be explained.

The most simple machine used for striping is the hand-pushed one. This machine is pushed by an operator following either the side of the road or the pre-marks depending on the task that is performed as shown in the figure 2.1(a). For the pre-marking, the machine is equipped with two actuating arms which can be freely adjusted to change the distance from a marker on the vehicle to where the pre-marks should be painted on the road.

When the task at hand is to paint the final stripes, the operator's job is to guide the device to make sure that the stripe and the pre-mark matches. The wheels can also be powered by motors instead of pushed by the operator to achieve a more constant velocity, resulting in a more homogeneous thickness of paint.

These kinds of machines are the cheapest and can be a good alternative to bigger and more expensive machines for short distances. Nevertheless, they are limited to the walking speed of the operator, thus they are not suitable for long distances.

The ride-on machines, such as the one in figure 2.1(b), is another alternative to road striping, these machines are vehicles on which all the painting materials are included.

These machines let the operator sit on top of them and their speed can reach up to 10 km/h[4]. They require one operator to drive them and make them follow the pre-marks to stripe the road. Graco®[5], which is one of the major manufacturer of road marking equipment, also made a system for these machines, allowing road striping without needing any pre-marks.

(a) Hand pushed device.[2]

(b) Ride-on device.[5]

Figure 2.1. Example of two devices used to pre-mark or mark a road.

The faster solution is the truck mounted machines which allow both pre-marking and final marking. For the pre-marking, the truck is also equipped with two actuating arms: one pre-marking the road while the other allows the driver to keep a constant distance to the side of the road. Either a laser or a camera can be mounted on the end of the arm such that the driver can verify his position with respect to the road edge, while keeping an eye on the road. The pre-marking task can be performed at 32 km/h using this device. [6]

(a) Embedded device.

(b) Dedicated truck.

Figure 2.2. Example of two truck mounted devices requiring two operators. [6]

For the final marking using the truck-mounted solution, two overall methods exist: A manual one requiring two operators and a video guided one requiring only a driver. To perform the manual method, one operator drives the truck close to the pre-marks, while the other one controls an arm at the back of the truck at which the paint gun is attached to. Two different vehicles performing this task are shown in figure 2.2(a) and 2.2(b). The

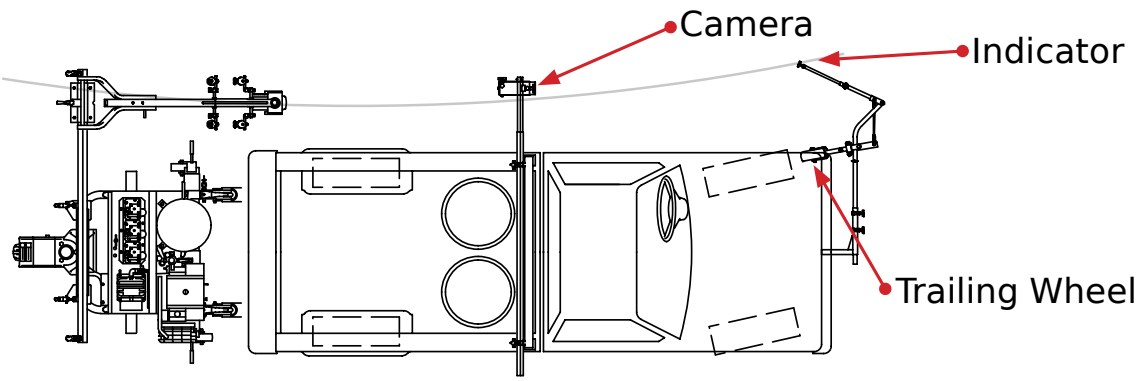


Figure 2.3. A one-operator truck device using camera. [4]

video guided method is more used nowadays. In this configuration, which is presented in figure 2.3, the vehicle is equipped with two arms on the same side, one handling an indicator while the other one holds a camera allowing the driver to watch a live video stream of the exact position. Then the paint gun will follow the indicators path and the road can be striped using only one operator. The striping speed limit of the truck mounted system is determined by the flow rate of the paint and the width of the striped line. As an example the Graco's RoadLazer™ Line Striper [4] is able to reach a maximum striping speed of 19 km/h to perform a 101 mm wide line but this maximum speed drops to 11 km/h when the line width is 304 mm.

2.1.2 Road Stripe Regulations

This section contains the overall regulations regarding road striping in Denmark. Only road stripes that are placed in the length direction of the road will be considered. All the regulations that are stated in the following section are stated by The Danish Road Directorate in [7], [8] and [9]. These three booklets can also be found on the CD ◎ `/literature/road_striping_rules`. It should be noted that the following chapter only sums up the most important regulations relevant to this project. A full walkthrough of all regulations in connection with road stripes is outside the scope of this project. For a full description refer to the three booklets from The Danish Road Directorate.

There exist many different types of road stripes that are used for different purposes and at different road layouts. The general purpose of road markings is to guide traffic, warn about upcoming danger and regulate traffic. Stripes in the length direction of the road are especially used to guide traffic into the right traffic lanes and regulate traffic, e.g. by prohibiting overtaking at places where the surroundings reduces visibility of the road. Road stripes may never be placed in such a way that they guide traffic towards dangerous obstacles.

Throughout the rest of this report the words *roadway*, *road lane* and *distance between lines* will be used to refer to different parts of the road. To avoid confusion, the three expressions can be seen in figure 2.4 and are defined as follows:

Roadway: Is the whole width of the asphalt from edge to edge.

Road lane: If it is the outermost traffic lane it is the area between the inside of the edge line, (or the edge of road if an edge line is not present) to the middle of the closest centre or separation line. If it is not the outermost traffic lane, it is the area between the middle of two centre or separation lines.

Distance between lines: Is the distance inside two lines in a traffic lane.

Stripes in the length direction of the road can be divided into three different categories:

Centre lines: Separates traffic lanes of opposite direction of travel.

Separation lines: Separates traffic lanes of same direction of travel.

Edge lines: Limits the part of the road available to motor vehicles.

Figure 2.4 shows an example of the different stripes needed for a three-lane highway. The grey arrows indicate the direction of travel.

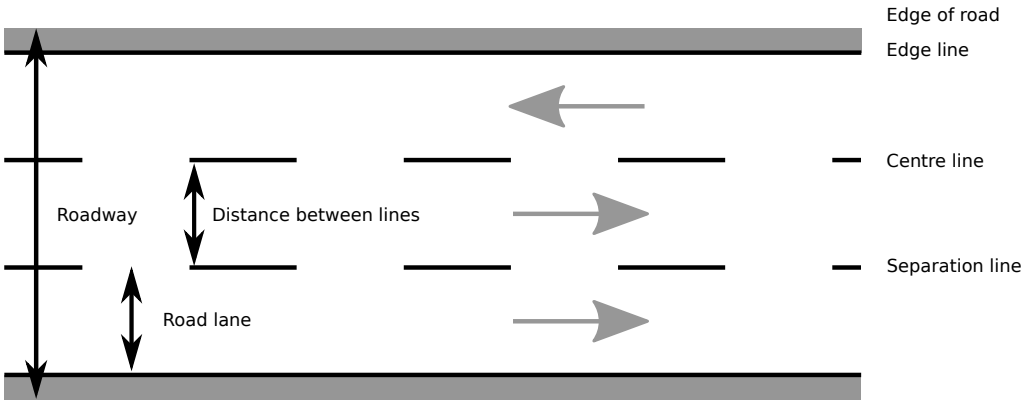


Figure 2.4. The layout of a road with three lanes.

Centre lines *must* be present at all numbered roads outside cities when the road lane is wider than or equal to 2.8 m. Furthermore they *should* be present at all numbered roads inside cities and essential roads managed by the council, when the road lane is wider than or equal to 2.8 m. If the road lane is not wide enough on these types of roads, centre lines can not be placed without a granted exemption. For all other types and widths of roads centre lines *can* be present.

Separation lines *must* be present on motorways and roads where there are three or more road lanes with same direction of travel. Furthermore they *should* be present whenever road lane is wider than or equal to 3 m or the road has two or more road lanes of same direction of travel. The distance between lines *must* be at least 2.75 m, otherwise an exemption must be granted.

Edge lines are only mandatory in two cases. The first one is when demarcation of special road lanes is needed. This could be lanes such as crawler lanes, emergency lanes or bus lay by lanes. The second case is when there is no traffic light at an intersection on major roads. They *should* also be present at all major roads with a road lane width of 5.8 m or wider.

Another important part of the regulations, regarding road stripes, is which types of stripes that should be placed in different road situations. Without going through all the different types, some of the most common can be found in table 2.1.

Different road stripes	Use	Dimensions [m]
	Can be used as centre or separation line to allow overtaking from both sides of the line.	L = 5/5/2.5 S = 10/10/5 W = 0.15/0.1/0.1
	Can be used as centre or separation line to prohibit overtaking from one side of the line.	L = 5/5/2.5 S = 10/10/5 W = 0.15/0.1/0.1 D = 0.15/0.1/0.1
	Can be used as centre or separation line to prohibit overtaking from both sides of the line.	W = 0.15/0.1/0.1 D = 0.15/0.1/0.1
	Can be used as edge line.	W = NA/0.1/0.1 D = NA/0.1/0.1
	Can be used as edge line when the space between the edge line and the edge of the road allow bikes and scooters to use this part of the road.	W = 0.3/0.3/0.3 D = 0.3/0.3/0.3

Table 2.1. In the dimensions column, L denotes length of a stripe, S denotes punctuation space, W denotes width of the stripe and D denotes distance between two stripes. At every measurement three values are given. The first one is the value that is used for motorways, the second is for highways and the last value is used in cities where the speed limit is equal to or less than 60 km/h.

The dimensions are allowed to diverge from $-5\,\%$ to $+10\,\%$. However the relationship between the length of a punctuated line and the space in between each punctuation may diverge up to $\pm 10\,\%$. Edge lines are normally placed with a minimum of 10 cm from the road edge.

2.2 Automation Approaches

2.2.1 Different Approaches to the Problem

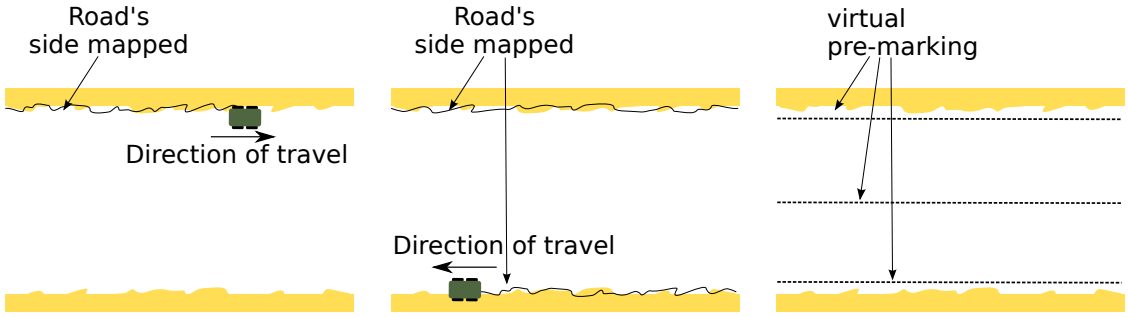
This section will introduce different overall approaches that could be used to automate the striping process of a road. Two main approaches will be discussed, both requiring the same hardware and producing the same overall outputs. The result of both methods is a virtual pre-marking defined by some trajectories defined by GPS-coordinates and a final stripe painted on the road. Which of these methods that will be the optimal one, depends on the actual layout of the road and the requirements in terms of time consumption and precision.

It is suggested to adapt the existing procedure of first doing a pre-marking of the road and then afterwards paint the stripes. In this project the pre-markings will not be present as an actual marking on the road. Instead these will be trajectories saved in the robot to

indicate placement of the stripes. To achieve a good precision of the striping process, the plan is to do a complete mapping of the road. By first evaluating the shape of the road it is then possible to determine precise virtual pre-markings.

The map of the road should result in precise GPS coordinates of the outline of the road. In order to achieve this, the robot must drive along the road and using a camera visually detecting the edges of the road. The video detection of the edges should be combined with GPS information of the robots own location to create this map.

When a precise map of the road is known, the pre-markings can be processed to smoothen out irregularities in the edge of the road. This principle is illustrated in figure 2.5. By storing the mapping of the road permanently a later restriping of the road can be performed without remapping the road. This makes the re-striping process faster than the first striping.



(a) Pre-mark one side. (b) Pre-mark the other side. (c) Compute the side pre-mark and then deduce the central one.

Figure 2.5. The two steps detection of the road's side and computation of the centre and side pre-marks.

The pre-marking can be split into two different approaches. The first one consists of mapping the road while the autonomous robot drives in the middle of the road as presented in figure 2.6. While travelling on the road, the robot can determine both sides of the road, at the same time, using video input. These data are then processed, either real time or later, to generate the virtual pre-marks of each stripe. This approach allows the robot to pre-mark the two edge stripes as well as the centre at the same time. This results in a faster mapping, but some uncertainties are expected since the edge of the road is far away from the camera. Problems can however occur if the roadway is too wide, then the distance from the robot to the edge of the road can become so great that they can become impossible to detect.

The second approach is to handle this problem by making the robot follow each of the road sides individually as illustrated in figure 2.5. Since the distance from the camera to the edge of the road is much smaller in this situation, the expected accuracy is better. The drawback of the procedure is that it requires the robot to drive the same stretch of road twice, which increases completion time. This procedure can however be used no matter the width of the roadway. Once both sides have been recognised using visual input, the data are processed to generate pre-marks in the same way as the first approach.

Depending of the configuration of the road, the data can be processed to compute either a centre line or multiple separation lines.

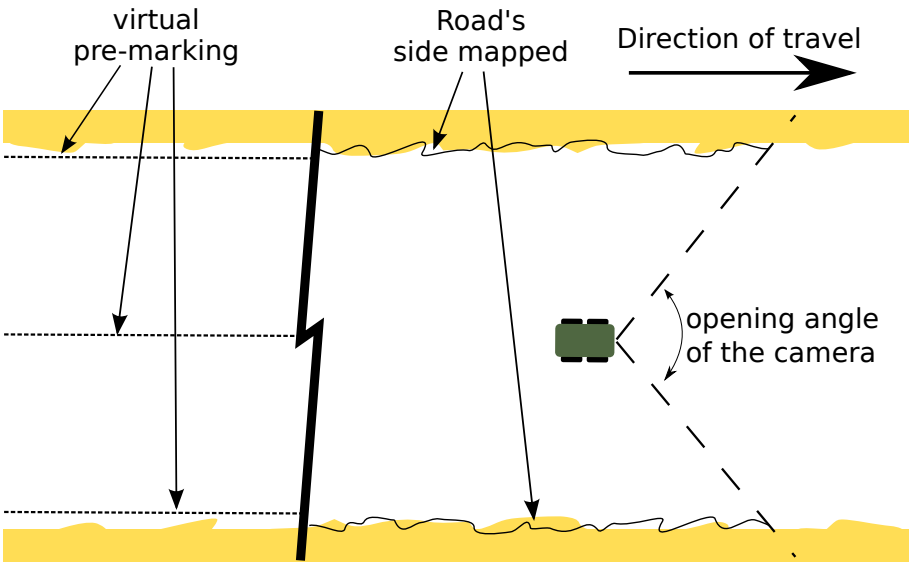


Figure 2.6. Pre-marking from the middle of the road.

Regarding the actual painting of the road, two approaches are considered, one consists of painting the road during the mapping, while the other one consists of using pre-marks as reference to paint the road. In the first case the mapping and painting task is merged such that the robot never runs on the same path twice. This approach is time saving but the downside is that without pre-marks, the smoothing of the stripe as well as the correction of sudden deviations is hard to perform. To solve this problem the second approach uses knowledge of the road ahead to generate the trajectory. This allows a better smoothing of the stripes and decreasing the imperfections. Again the downside of this improvement in accuracy is an increase in over all time consumption.

2.2.2 Localisation from Visual Input

To automate the process of striping the road, the capability of following the road must be adopted by the system. The task of detecting and following roads using visual input has received great attention for several years, and several methods have been developed.

Some methods first detect the stripes on the road using the large contrast between the road and the stripes. When the stripes are found the lanes of the road are hereafter extracted [10]. Since the task at hand is to stripe blank roads, no further analysis of methods relying on having stripes will be carried out.

In [11] Montemerlo et al. describes how they won the DARPA Grand Challenge with a Volkswagen Touareg R5. The states (i.e. position and velocity) of the vehicle are estimated by use of an unscented Kalman filter[12]. To estimate the states of the vehicle observations from a GPS, a GPS Compass, an Inertial Measurement Unit (IMU), and wheel encoders

are used. To determine desired driving speed and where to drive, a combination of laser range finders and a camera is used. The laser range finders are used to detect driveable terrain up to 25 meters ahead of the vehicle. This provides safe driving speeds up to 25 mph. To be able to increase the driving speed the driveable area is expanded by use of the camera. This concept is illustrated in figure 2.7. The boxed area is the driveable area found by the laser scanners. The area marked by red is the driveable areas extracted by the vision system.

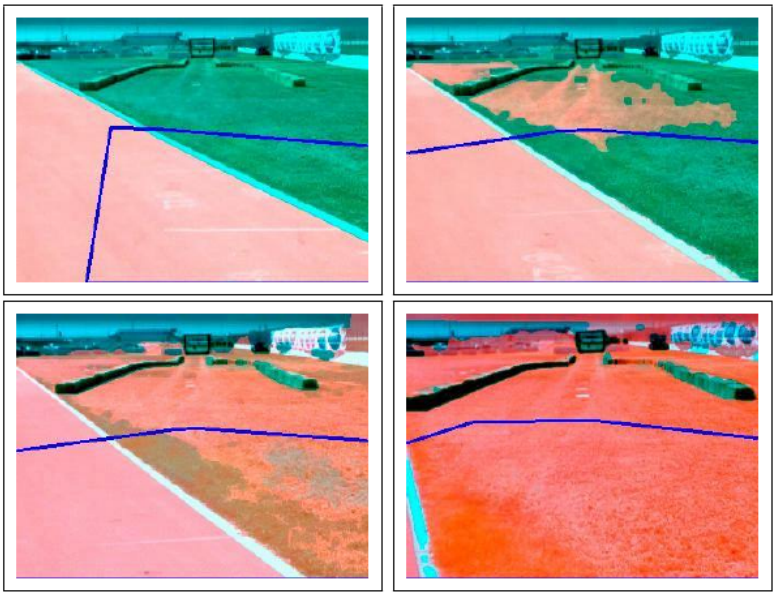


Figure 2.7. Principle of vision system used in the Stanley autonomous vehicle.[11]

The vision system uses the area found by the laser scanners to make a model of the structure of the driveable area in an image captured by the camera. Similar structures are hereafter found on the complete image to form the driveable area. From the images in figure 2.7 it is seen that the area detected as driveable changes from being only the road to include off-road terrain. The more off-road terrain included in the area detected by the laser scanners the more off-road terrain will be detected by the vision system as driveable. To make the changes in the driveable area smooth, a filter is applied to the parameters in the model describing the structure of the driveable area.

In [13] Aufrère et al. presents a method of detecting a road only using a monochrome camera. Using only a monochrome camera has several advantages. The main advantage is that the methods can be used at night using an infrared camera. Another advantage of using monochrome images compared to colour images is that less computational power is required in the analysis.

The method presented by Aufrère et al. relies on the fact that the vehicle is placed on the road. An initial guess of how the roads looks like is made from a triangular prototype (see figure 2.8(a)). From the initial prototype, the structure of the road is extracted and similar structures are found in the image (see figure 2.8(b)). The structure of the road is

comprised by a Gaussian brightness model. To eliminate the holes in the road and areas outside the road detected as road, some filtering is applied to achieve the final guess of the road (see figure 2.8(c)). In the next captured image a prototype deduced from the road found in the previous iteration is used instead of the initial prototype. This is done to make a better guess of the curvature of the road.

(a) Initial prototype. (b) Clustering. (c) Filtering.

Figure 2.8. The three steps Aufrère et al. uses to determine where road is in an image.[13]

From the road found in the image, the position and orientation of the camera with respect to the road is estimated along with the width of the road. This is done by creating a model of the road in the current image, and finding the sides of the road. By using the model and the found edges of the road, the parameters regarding the position and orientation of the camera can be estimated. This estimation is made using a Kalman filter to limit the variation in the parameters.

One of the major problems in detecting roads is the change of lightning conditions due to shadows from trees or other obstacles in the roadside. In [14] Álvarez et al. describes a road detection algorithm based on Illuminant Invariance to handle the problems with the shadows covering the road. The main idea is to use the nature of light, and a parameter of the camera to convert the captured RGB image to an greyscale image where shadows are not present, in the described approach the grey scale image is the feature space.

To determine which pixels in the input image is road, a likelihood-based classifier is used. To be able to use such classifier a model of the road must be available. The used model is comprised by a normalised histogram of the intensity of a number of pixels (seeds) in the bottom part of the image. The seeds are chosen in the bottom part since this is the region assumed most likely to be road. From the model, a threshold is set, and the greyscale image is thresholded to yield a binary image. The part of the binary image which includes the original seeds is extracted and considered as the road. This is done by using a connected-component procedure. The extracted road is hereafter post processed to fill small holes in the area detected as road.

Delimitation

I this chapter the process of striping roads has been discussed to determine how to automate the process. First existing machinery for road striping was analysed by looking into existing commercial solutions. The Danish road striping regulations were looked into, to determine what types of stripes exists and where they should be laid on the road.

To automate the process of striping the road, different ideas have been presented. One common thing that all the presented ideas require is that the robot knows its location with respect to the road. The task of determining the location of a vehicle on a road using a visual input was therefore analysed by looking into previous used methods for this task.

Ideally a road striping robot should be able to work completely without aid from human hand. It should be possible to place it on a road, ask it to start and it should paint all needed stripes on the road. However to accomplish such a task, the robot will have to be extremely complex in terms of functionality.

A fully automated robot must be able to decide exactly which kind of stripes that should go where at different road designs e.g. bends where the driver has poor visibility or at intersections. This job of choosing the right stripes is considered to be a too complex task to solve in this project. Instead the focus of this project will be upon developing a robot which can stripe an empty road without any obstacles.

The possibility of restriping an existing road is left out of the project. This is due to the fact that if the old stripes are not completely removed, the goal of the robot would be to place new stripes exactly on top of the old ones. This would require some changes to the proposed approach of mapping the edge of the road and from this determine the placement of the stripes.

3 Requirements Specification

This chapter specifies the requirements set for the product of this project. The first part describes some general thoughts about the topic, while the second part contains the specific requirements that should be complied with to be able to stripe a road successfully.

3.1 General Requirements

As stated in the analysis in the previous chapter the overall functionality of the robot is to automatically paint stripes on roads which have not been striped before or where old stripes have been completely removed beforehand. Additionally, the robot should not include any decision making regarding which kind of stripe goes where.

For an autonomous road striping robot to be a useful alternative to the existing machines, the robot constructed in this project should be faster and/or cheaper in commission. The final product should be evaluated by comparing with results achieved by existing solutions. This comparison should both include price, speed and accuracy.

The accuracy of the road striping process can be divided into two different overall requirements. One concerns the accuracy of the placement of the stripe with respect to the road. This requirement must make sure that the stripes are placed such that they follow the road and do not fluctuate from side to side. The other requirement concerns the actual shape of the stripes. All stripes must have a certain width and punctuated stripes must have a certain relation between the length of each piece and punctuation space between each segment.

The first requirement must be taken care of by the navigation software for the robot. The second requirement must be taken care of by a painting gun controller. Since the decision of which stripes to paint where is considered outside the scope of this project, it is chosen not to include a painting gun controller in the product.

The operating environment of the robot is assumed to be a blank road without any markings. It is also assumed that the precise location of the road is not known beforehand. The surroundings of a road can be any obstacle found along the road outside cities. This implies that the robot must be able to handle things like hills, valleys, forests, shadows cast on the road. On the other hand it is assumed that no other objects on the road are present i.e. no vehicles, leaves, branches etc.

The final performance of the system will largely depend on hardware of choice. For this project a preassembled robot platform including a variety of sensors has already been chosen. The performance of this project will therefore to some extent be limited by this choice. A full description of this robot platform can be found in appendix A.

3.2 Specific Requirements

This section contains a list of specific requirements that must be kept in order to achieve a useable road striping robot. As described in section 2.1.2 there are no specific regulations concerning the precision of the placement of the stripe on the road. The evaluation of the placement and smoothness of the road stripes is currently performed by visual inspection.

To compare the performance of the robot constructed in the project with existing solutions some measurable requirements should be stated. Since the way of evaluating stripes currently is by visual inspection, a reasonable approach could be to analyse images of existing stripes and extract a measure of smoothness from these. It is however considered a too large task to define such a measure of smoothness for road stripes from visual input. The only requirement set up for the precision is therefore that the robot must be able to drive along the road with a specified position in the transverse direction of the road. Due to time limitations no analysis of precision of the positioning of existing stripes have been carried out, an estimate is made. Being able to drive within $\pm 5\,\mathrm{cm}$ from a specified position in the transverse direction of the road is considered reasonable.

r.1 All stripes must be placed within $\pm 5\,\mathrm{cm}$ from a specified position in the transverse direction of the road.

Since it is chosen not to analyse existing road layouts, no specific requirement will be set on smoothness of the road stripe. During the implementation and design of the robot, the smoothness will be considered when the influence of different methods and parameters are evaluated. Generally choices should be made such that the striping is performed as smooth as possible.

Ideally the road striping process should be made as fast as possible. In order to compete with existing road striping machines, the average speed of the robot should be at least comparable. Since the RobuROC4, chosen as platform for this project, has a maximum speed of approximately $9\,\mathrm{km/h}$, the average speed is limited by this. To leave some headroom for turning, an expected driving speed of $7\,\mathrm{km/h}$ seems realistic. It is assumed that the vehicle will have to drive along the two sides of the road individually in order to map the road, and thereafter for every of the n stripes that must be painted on the road. This adds up to a total of $2 + n$ times that each part of the road has to be driven. From this an average striping speed can be calculated as:

$$v_{avg} = \frac{7}{2 + \mathrm{n}}\,\mathrm{km/h}$$

r.2 The minimum average speed of the striping process must be $\frac{7}{2+\mathrm{n}}\,\mathrm{km/h}$.

Part II

Design & Implementation

4 Design Introduction

In order to achieve the functionality described in the Requirements Specification, a combination of hardware and software must be assembled. From this point forward in the whole construction including both hardware and software will be referred to as the Autonomous Road Striper (ARS). From the analysis performed in the previous part the overall task of striping the road can be divided into the following three major objectives.

- Map the road.
- Generate pre-marks on the road.
- Stripe the road from the pre-marks.

In order to paint a road, the first task is to find out exactly where it is located. To do this, a visual input in form of a camera is mounted on the robot. To determine the road from the video input a road detection algorithm must be developed. This should be able to detect exactly which part of the video feed that is a part of the road, and what is a part of the surroundings. By knowing where the robot is and what heading it has, it is then possible to save whereabouts of the road that must be used for later pre-marking. Additionally the detection of the road should also allow the robot to follow the road in order to map the road ahead. As discussed in section 2.2 the mapping of the road can be performed in several different ways. Since precision of the detection of the road edge is vital, it is chosen to drive at the edge of the road and map one edge at the time. The sacrifice for this higher precision is the speed at which the mapping can take place, since it requires running over each part of the road twice, to create the map.

Each time a frame from the video input has been put through the road detection algorithm, two different tasks must be performed. First of all the data of the road must be saved such that this data can be used to generate precise pre-markings when the whole road has been mapped. Since this pre-marking exactly specifies where each stripe should be laid, the accuracy of this is absolutely vital to the final performance of the system. Places where the edge of the road is uneven, the pre-marking should be smoothed out such that the stripes are smooth without wobbling. Secondly a trajectory following the edge of the road some distance ahead must be generated right away, in order to make the robot follow the edge of the road.

The final and last task of the robot is to follow a trajectory laid out in the pre-marking phase and apply paint to the road. This should be done as fast as possible, but still keep a very high precision such that the stripes are painted exactly where the pre-markings are. To accomplish this task a controller which is able to follow a trajectory very precisely must be developed.

To achieve the functionality described above, a system combining the right hardware and software must be developed. The general structure of the ARS can be seen in figure 4.1.

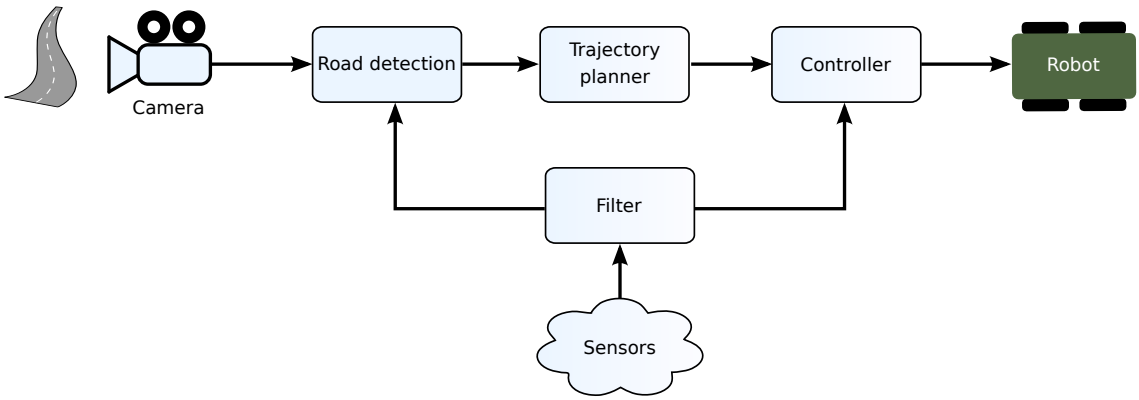


Figure 4.1. Block diagram of the setup.

Starting from the left, the *Camera* captures images and passes these on to the *Road detection*. When the location of the road has been determined by this algorithm, it is passed on to the *Trajectory planner*. Since information about the road is passed on in a local frame, the first task is to convert these data into a global frame. The *Trajectory planner* then saves the raw road edge data for later use to be able to generate pre-markings on the road. At the same time the new road information is send as a trajectory to the *Controller*, which then steers the robot along the road. When the whole road has been mapped, the same *Controller* is used to follow the final pre-markings when it paints the road.

The various *Sensors* provide information about current location, heading, linear, and rotational velocity of the robot. In order to remove noise on these measurements a filter is implemented. When the sensor data has been filtered it is send to both the *Road detection* and the *Controller*. The *Road detection* saves information about current position and heading when an image is taken from the video feed in order to allow computation of the exact location of the road in a global frame. The *Controller* uses the data to steer the robot onto a trajectory either when mapping or striping the road. A full description of the hardware and the various sensors mounted on the robot can be found in appendix A.

The rest of the Design and Implementation part deals with the design and implementation of these blocks. Each block will be dealt with in the following chapters where the theory and thoughts behind each block is stated. In the end of the Design and Implementation part is a chapter called Implementation, that briefly describes how the software has been implemented on the robot using a special Application Programming Interface (API) called Robotic Operating System (ROS).

5 Modelling of the RobuROC4

This chapter contains the modelling of the RobuROC4. The basis will be a kinematic model which will be expanded to include the dynamics of the robot as well. Since the RobuROC4 has internal controllers which are unknown, a grey-box model with estimated parameters will be used in order to determine the behaviour of the robot. When all the system equations have been determined, differential equations for each state will be deduced. In the end a state space model will be presented as the final model of the system and a validation of this is performed.

The global position of the robot will be defined in the Universal Transverse Mercator (UTM) coordinate system [15, p. 102]. In the rest of the report x and y components will be used instead of easting and northing components respectively. The velocity of the RobuROC4 in the x and y direction can be determined from the two local parameters v and θ respectively denoting the linear velocity and heading of the robot. During this project the robot is assumed to drive on a flat surface where changes in the z direction can be disregarded.

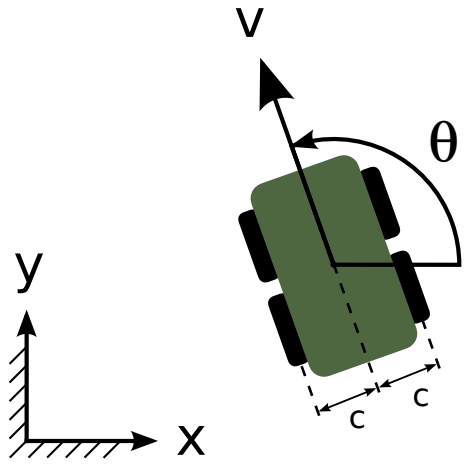


Figure 5.1. Definition of angle and velocity with respect to the global coordinate system.

From figure 5.1 it can be seen that the velocity in the global x, y coordinate system can be found as equation 5.1 and 5.2.

$$\dot{x} = v\,cos(\theta) \tag{5.1}$$

$$\dot{y} = v\,sin(\theta) \tag{5.2}$$

A kinematic model from [16] is used to model the local linear and rotational velocity. This paper is based on a four wheeled skid steering vehicle, but does not take any slippage into

account. The kinematic model is stated as equation 5.3 and 5.4.

$$v = \frac{r\,(\omega_R + \omega_L)}{2} \tag{5.3}$$

$$\dot{\theta} = \frac{r\,(\omega_R - \omega_L)}{2\,c} \tag{5.4}$$

Where:

$\omega_{R/L}$ is rotational velocity of right and left wheel

r is radius of wheel

c is half the width of the vehicle

The RobuROC4 has forward velocity v_{ref} and rotational velocity ω_{ref} as inputs. From these two it is possible to steer the robot. Tests have shown that the kinematic model in 5.3 and 5.4 are in fact used by the internal controller of the RobuROC4 to calculate its individual wheelset velocities. By solving these two equations with respect to ω_R and ω_L the reference for each wheelset can be found as equation 5.5 and 5.6.

$$\omega_{R-ref} = \frac{v_{ref} + c\,\omega_{ref}}{r} \tag{5.5}$$

$$\omega_{L-ref} = \frac{v_{ref} - c\,\omega_{ref}}{r} \tag{5.6}$$

5.1 Dynamics of the System

In order to introduce the dynamic behaviour of the RobuROC4 into the model, a grey-box model of the robot is used. Since the internal structure of the robot is unknown in terms of gears and controllers, a qualified guess upon its structure consisting of a simple PI-controller and a model of a DC motor is used. The closed loop model of the right wheelset can be seen in figure 5.2. Note that the load torque τ_{Rl} has been moved outside the plant and is subtracted from the torque τ_R produced by the engine.

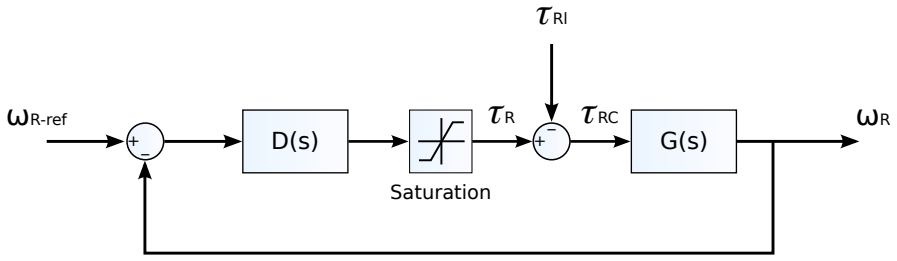


Figure 5.2. Right wheelset feedback loop in the grey-box model.

Assuming a PI-controller is used as controller $D(s)$ to control the rotational velocity of the wheels, this can be written as:

$$D = K_p + K_i\frac{1}{s} \tag{5.7}$$

Using a first order model of a DC engine, expression 5.8 can be found from Newton's second law and used as plant $G(s)$.

$$J_w \dot{\omega}_R = \tau_{RC} - b\,\omega_R$$
$$G = \frac{\omega_R}{\tau_{RC}} = -\frac{1}{J_w\, s + b} \tag{5.8}$$

Where: (5.9)

b is the wet friction coefficient

J_w is the total inertia of gear and wheel

By adding the contributions from the two inputs ω_{R-ref} and τ_{Rl}, equation 5.10 can be found.

$$\omega_R = \frac{D\,G}{1 + D\,G}\,\omega_{R-ref} - \frac{G}{1 + D\,G}\,\tau_{Rl} \tag{5.10}$$

In the exact same way an expression for the left wheel rotation can be found as:

$$\omega_L = \frac{D\,G}{1 + D\,G}\,\omega_{L-ref} - \frac{G}{1 + D\,G}\,\tau_{Ll} \tag{5.11}$$

The load torques that are a part equation 5.10 and 5.11 needs to be determined in order to progress with these equations. The two torques are chosen to be modelled with two different contributions. One which is caused by the resisting force when the robot is accelerated forward ($\tau_{\dot{v}}$) and one which is the resisting torque when the robot has a rotational acceleration ($\tau_{\ddot{\theta}}$).

$$\tau_{Rl} = \tau_{\dot{v}} + \tau_{\ddot{\theta}}$$
$$\tau_{Ll} = \tau_{\dot{v}} - \tau_{\ddot{\theta}}$$

The resisting force (F_{res}) of the vehicle during forward acceleration can simply be found from Newton's second law. This force can be converted into a torque on the wheel by multiplication with wheel radius (r). The torque is divided by two since there are two wheel pairs to overcome this force.

$$F_{res} = m\,\dot{v}$$
$$\tau_{\dot{v}} = \frac{F_{res}\,r}{2} = \frac{m\,\dot{v}\,r}{2}$$

The resistance torque from rotational acceleration ($\tau_{\ddot{\theta}}$) is also found from Newton's second law. Again the expression is divided by two due to the two wheel pairs.

$$\tau_{\ddot{\theta}} = \frac{J_v\,\ddot{\theta}}{2}$$

The two final load torques can the be written as equation 5.12 and 5.13 by combining the equations above.

$$\tau_{Rl} = \frac{m\,\dot{v}\,r}{2} + \frac{J_v\,\ddot{\theta}}{2} \tag{5.12}$$
$$\tau_{Ll} = \frac{m\,\dot{v}\,r}{2} - \frac{J_v\,\ddot{\theta}}{2} \tag{5.13}$$

This concludes the deviation of the system equations from which the final model will be deduced. Next, the equations will be combined to find the differential equations for each state.

Since x and y are already stated as differential equations the following deduction will find an expression for θ. The kinematic model gives us the following equation for the rotation of the vehicle $\dot{\theta}$, as a function of the wheel rotations.

$$\dot{\theta} = \frac{r}{2\,c}\,\omega_R - \frac{r}{2\,c}\,\omega_L$$

Inserting the dynamic model of the two wheel rotations into this equation results in the expression bellow. Note that a constant $k1$ has been introduced to simplify notation.

$$k1 = \frac{r}{2\,c}$$
$$\dot{\theta} = k1\left(\frac{D\,G}{1+D\,G}\,\omega_{R-ref} - \frac{G}{1+D\,G}\,\tau_{Rl}\right) - k1\left(\frac{D\,G}{1+D\,G}\,\omega_{L-ref} - \frac{G}{1+D\,G}\,\tau_{Ll}\right)$$

Inserting ω_{R-ref}, ω_{L-ref}, τ_{Rl} and τ_{Ll} simplifies the equation to:

$$\dot{\theta} = \frac{D\,G}{1+D\,G}\,\omega_{ref} - k1\,\frac{G}{1+D\,G}\,J_v\,\ddot{\theta} \tag{5.14}$$

To find the final expression for θ, the equation 5.14 is Laplace transformed.

$$\theta\,s = \frac{D\,G}{1+D\,G}\,\omega_{ref} - k1\,\frac{G}{1+D\,G}\,J_v\,\theta\,s^2$$
$$\theta = \frac{\frac{D\,G}{1+D\,G}}{s + k1\,\frac{G}{1+D\,G}\,J_v\,s^2}\,\omega_{ref}$$

Inserting the controller D and plant G specified in 5.7 and 5.8 results in:

$$\theta = \frac{K_p\,s + K_i}{s^3\,(J_w + k1\,J_v) + s^2\,(b + K_p) + s\,K_i}\,\omega_{ref}$$

Inverse Laplace transform results in the final differential equation for the state θ.

$$\dddot{\theta} = \frac{K_p}{J_w + k1\,J_v}\,\dot{\omega}_{ref} + \frac{K_i}{J_w + k1\,J_v}\,\omega_{ref} - \frac{b + K_p}{J_w + k1\,J_v}\,\ddot{\theta} - \frac{K_i}{J_w + k1\,J_v}\,\dot{\theta} \tag{5.15}$$

In an approach similar to the one for θ we can derive the differential equation for v. From the kinematic model can we write:

$$v = \frac{r}{2}\,\omega_R + \frac{r}{2}\,\omega_L$$

Inserting the dynamic model for the two wheelsets and introducing a constant $k2$ yields:

$$k2 = \frac{r}{2}$$
$$v = k2\left(\frac{D\,G}{1+D\,G}\,\omega_{R-ref} - \frac{G}{1+D\,G}\,\tau_{Rl}\right) + k2\left(\frac{D\,G}{1+D\,G}\,\omega_{L-ref} - \frac{G}{1+D\,G}\,\tau_{Ll}\right)$$

Inserting ω_{R-ref}, ω_{L-ref}, τ_{Rl} and τ_{Ll} yields:

$$v = \frac{D\,G}{1 + D\,G}\,v_{ref} - k2\,\frac{G}{1 + D\,G}\,m\,r\,\dot{v} \tag{5.16}$$

Laplace transform of equation 5.16 results in:

$$\begin{aligned} v &= \frac{D\,G}{1 + D\,G}\,v_{ref} - k2\,\frac{G}{1 + D\,G}\,m\,r\,v\,s \\ &= \frac{\frac{D\,G}{1+D\,G}}{1 + k2\,\frac{G}{1+D\,G}\,m\,r\,s}\,v_{ref} \end{aligned}$$

By introducing the expressions from our controller D and G we arrive at the following:

$$v = \frac{K_p\,s + K_i}{s^2\,(J_w + k2\,m\,r) + s\,(b + K_p) + K_i}\,v_{ref} \tag{5.17}$$

Inverse Laplace transform results in the final differential equation for the state v.

$$\ddot{v} = \frac{K_p}{J_w + k2\,m\,r}\,\dot{v}_{ref} + \frac{K_i}{J_w + k2\,m\,r}\,v_{ref} - \frac{b + K_p}{J_w + k2\,m\,r}\,\dot{v} - \frac{K_i}{J_w + k2\,m\,r}\,v \tag{5.18}$$

This concludes the deduction of the state equations. Equation 5.1, 5.2, 5.15, and 5.18 are the state equations which will be used to set up a state space model. However in equation 5.15 and 5.18 derivatives of the inputs are present. Since this is not allowed in the state space formulation, these must be avoided.

5.2 State Space Representation

To arrive at the final state space form, we introduce new state variables to avoid derivative of inputs. The first state equation that will be rewritten is 5.15. To simplify the expression we introduce four constants.

$$\begin{aligned} a1 &= \frac{2\,c\,(b + K_p)}{2\,J_w c + r\,J_v} \\ a2 &= \frac{2\,c\,K_i}{2\,J_w\,c + r\,J_v} \\ b0 &= \frac{2\,c\,K_p}{2\,J_w\,c + r\,J_v} \\ b1 &= \frac{2\,c\,K_i}{2\,J_w\,c + r\,J_v} \end{aligned}$$

Equation 5.15 now looks as follows:

$$\dddot{\theta} = -a1\,\ddot{\theta} - a2\,\dot{\theta} + b0\,\dot{\omega}_{ref} + b1\,\omega_{ref} \tag{5.19}$$

The first two state variables are simply θ and $\dot{\theta}$ which denotes the angle and angular velocity of the vehicle respectively. The third state variable can now be expressed as follows to avoid the derivative of our input:

$$q_1 = \ddot{\theta} - b0\,\omega_{ref}$$

From this we can find an expression of the derivative of $\dot{\theta}$:

$$\ddot{\theta} = q_1 + b0\,\omega_{ref}$$

We can now rewrite the derivative of q_1. In this we insert $\dddot{\theta}$ from equation (5.19).

$$\dot{q}_1 = \dddot{\theta} - b0\,\dot{\omega}_{ref} = -a1\ddot{\theta} - a2\,\dot{\theta} + b1\,\omega_{ref}$$

Expressing $\dot{q}_1$ with respect to our new states results in the final state space equation which does not include any derivative of input.

$$\dot{q}_1 = -a1\,q_1 - a2\,\dot{\theta} + (b1 - a1\,b0)\,\omega_{ref} \tag{5.20}$$

The same problem with derivative of inputs is present in equation 5.18, and the same approach is taken to avoid these. First constants are introduced to simplify the equation:

$$c1 = \frac{2\,(b + K_p)}{2\,J_w + r^2\,m}$$
$$c2 = \frac{2\,K_i}{2\,J_w + r^2\,m}$$
$$d0 = \frac{2\,K_p}{2\,J_w + r^2\,m}$$
$$d1 = \frac{2\,K_i}{2\,J_w + r^2\,m}$$

Equation 5.18 now looks as follows with the defined constants:

$$\ddot{v} = -c1\,\dot{v} - c2\,v + d0\,\dot{v}_{ref} + d1\,v_{ref}$$

To avoid derivative of the input v_{ref}, a state defined as follows can be used:

$$q_2 = \dot{v} - d0\,v_{ref}$$

This implies that the expression of $\dot{v}$ can be written as:

$$\dot{v} = q_2 + d0\,v_{ref} \tag{5.21}$$

$\dot{q}_2$ can now be stated as:

$$\dot{q}_2 = \ddot{v} - d0\,\dot{v}_{ref} = -c1\,\dot{v} - c2\,v + d1\,v_{ref}$$

The final state equation can then we written as equation 5.22.

$$\dot{q}_2 = -c1\,q_2 - c2\,v + (d1 - c1\,d0)\,v_{ref} \tag{5.22}$$

Combining equation 5.1, 5.2, 5.20, 5.21, and 5.22 we can write the final continuous dynamic model into state space form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{\theta} \\ \dot{q}_1 \\ \dot{v} \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 & sin(\theta) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -a2 & -a1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -c2 & -c1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \dot{\theta} \\ q_1 \\ v \\ q_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & b0 \\ 0 & b1 - a1\,b0 \\ d0 & 0 \\ d1 - c1\,d0 & 0 \end{bmatrix} \begin{bmatrix} v_{ref} \\ \omega_{ref} \end{bmatrix} \tag{5.23}$$

Before this model can be used for controller design and Kalman filtering it has to be discretised. The method imposed here is the Euler discretisation which can be stated as equation 5.24. This is not the most accurate discretisation method, but remains sufficiently accurate as long as the sampling time (Ts) is kept short compared to the system dynamics.

$$z[k+1] = z[k] + \left.\frac{dz(t)}{dt}\right|_{t=k\,Ts} Ts \tag{5.24}$$

Performing the Euler discretisation on all the states in the system yields a discretised system as seen in 5.25.

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \theta[k+1] \\ \dot{\theta}[k+1] \\ q_1[k+1] \\ v[k+1] \\ q_2[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & cos(\theta)\,Ts & 0 \\ 0 & 1 & 0 & 0 & 0 & sin(\theta)\,Ts & 0 \\ 0 & 0 & 1 & Ts & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & Ts & 0 & 0 \\ 0 & 0 & 0 & -a2\,Ts & 1 - a1\,Ts & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & Ts \\ 0 & 0 & 0 & 0 & 0 & -c2\,Ts & 1 - c1\,Ts \end{bmatrix} \begin{bmatrix} x[k] \\ y[k] \\ \theta[k] \\ \dot{\theta}[k] \\ q_1[k] \\ v[k] \\ q_2[k] \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & b0\,Ts \\ 0 & (b1 - a1\,b0)\,Ts \\ d0\,Ts & 0 \\ (d1 - c1\,d0)\,Ts & 0 \end{bmatrix} \begin{bmatrix} v_{ref}[k] \\ \omega_{ref}[k] \end{bmatrix} \tag{5.25}$$

5.3 Parameter Estimation

The state space model presented in section 5.2 contains some physical parameters that are unknown. In order to use the model these must first be determined. Some of these are easily accessible from data sheets of the robot, while others requires experiments to be carried out.

The first parameter that will be determined is the moment of inertia of the individual wheel sets (J_w). Since the wheels, gear and motors are inseparable this inertia will contain contributions from all of them. To find this Newton's second law is used. By knowing the acceleration and the applied torque on the wheel, J_w can be calculated. From the datasheet the maximum total torque that the RobuROC4 can deliver is 218.86 Nm [17]. Since there are two wheelsets the torque on one wheelset must be half of that. By applying a pulse on the velocity input of the robot while all wheels are lifted from the ground, the unloaded acceleration of the wheel can be measured to be $acc_\omega = 7.18$ rad/s^2. From this acceleration the inertia can be calculated as:

$$\begin{aligned} J_w &= \frac{\frac{\tau_m}{2}}{acc_\omega} \\ &= \frac{\frac{218.86}{2}}{7.18} = 15.2\ kg\,m^2 \end{aligned}$$

The second parameter that will be determined is the moment of inertia of the RobotROC4 (J_v). This will be calculated from the assumption that the robot can be considered a ridged body shaped as a box with even distribution of mass.

$$\begin{aligned} m &= 140\ kg \quad \text{(mass of vehicle)} \\ l &= 0.925\ m \quad \text{(length of vehicle)} \\ w &= 0.680\ m \quad \text{(width of vehicle)} \end{aligned}$$

$$\begin{aligned} J_v &= \frac{1}{12} m \left(l^2 + w^2\right) \\ &= \frac{1}{12} 140 \left(0.925^2 + 0.680^2\right) = 15.4\ kg\, m^2 \end{aligned}$$

The parameters K_p, K_i and b which determines the model of the motor are estimated with a parameter estimation toolbox in MATLAB® called Senstools [18, cha. 5]. This toolbox takes in as argument a sampled input signal and the resulting output measured on the robot. By defining a model with the desired parameters as unknowns, and providing an initial guess of the parameters, the toolbox will try to determine the unknown parameters. This is done by iterating through different values of the parameters, while trying to minimise the error between simulated and measured output. The iteration stops when Senstools can not minimise the error any further. The found parameters are returned in the MATLAB® workspace.

The parameter estimation is performed with a run in the forward velocity. To get the data needed for the estimation, the RobuROC4 was applied with a pulse in forward velocity while being placed on the floor. Using a motion tracking system, the velocity of the robot was recorded.

Equation 5.17 from the modelling section is used as the model of the robot.

$$\frac{v}{v_{ref}} = \frac{K_p\, s + K_i}{s^2 \left(J_w + k2\, m\, r\right) + s \left(b + K_p\right) + K_i}$$

Before the estimation starts Senstools must be given an initial guess of the parameters. Since these are unknown the following have been found as initial guess that makes Senstools converge to a solution:

$$\begin{aligned} K_p &= 20 \\ K_i &= 50 \\ b &= 20 \end{aligned}$$

By feeding Senstools with the model, the initial parameters and measured input and output from the experiment the estimation can be performed. The script running the estimation can be found on the CD /matlab_files/parameter_estimation/par_est.m. Figure 5.3 shows the output of the estimation. In this figure the measured output of an approximately 4.5 s pulse is plotted together with the estimated model. An error of 9.6 % is present between the measured data and the model with the estimated parameters.

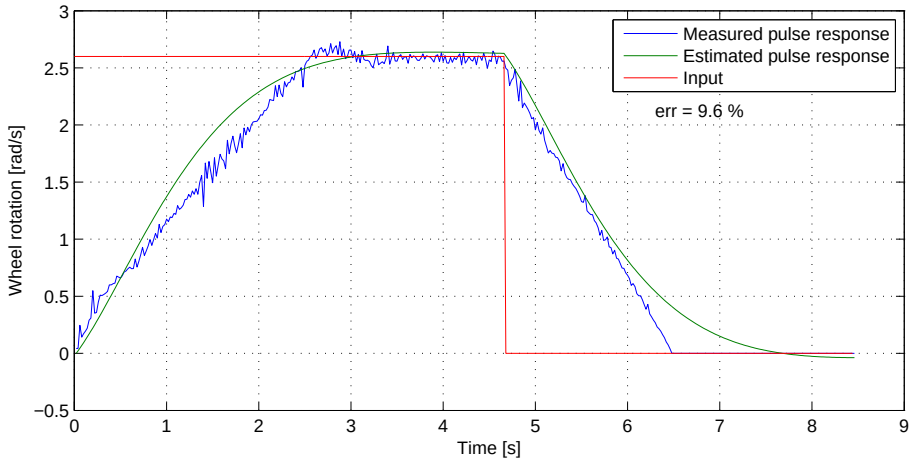


Figure 5.3. The resulting model with the estimated parameters plotted together with the measured output.

The estimated parameters can be found in table 5.1. It could seem that K_i is large compared to the K_p gain, but looking at figure 5.3 the combination of the found parameters seems to model the dynamics of a forward acceleration in a satisfactorily way.

Parameter	Value	Description
r	0.287 m	Radius of wheel
c	0.34 m	Width of vehicle
J_w	15.2 kg m^2	Moment of inertia of wheel
J_v	15.4 kg m^2	Moment of inertia of vehicle
K_p	16.63	Proportional gain in motor
K_i	126.60	Integral gain in motor
b	64.36	Wet friction of gear and motor

Table 5.1. Table with values of the parameters in the model.

From the parameters specified in table 5.1, all the constants that makes up the state space model can be calculated. The values of these constants can be found in table 5.2.

Constant	Value
$a1$	3.7323
$a2$	5.8342
$b0$	0.7664
$b1$	5.8342
$c1$	3.8630
$c2$	6.0384
$d0$	0.7932
$d1$	6.0384

Table 5.2. Table with constants in the model.

5.4 Model Validation

To test if the found model combined with the estimated parameters is a good representation of the RobuROC4, model validation tests are carried out. The measurement journal describing the exact procedure of each test can be found in appendix B. The first test that will be carried out is a pulse response of 1.3 m/s in the linear velocity. In oppose to the pulse used for estimating the parameters of the model, the input in this test is half the maximum velocity of the vehicle. Additionally the velocity is measured by the Real Time Kinematic (RTK) GPS instead of the motion tracking system. The result of this test can be seen in figure 5.4. Since the GPS is mounted on a pole that can bend and move a small amount when the robot is driving, the measurements of the velocity appears to be varying more compared to the measurements made by the motion tracking system. Generally it is observed that the measured acceleration seems to be faster than the model, this contradicts the previous measurements seen figure 5.3, where the model seems to follow this better. One explanation for this can be that when the vehicle accelerates, the pole at which the GPS is mounted is pushed a bit backwards. When the acceleration stops the pole then moves a bit forward. This creates an acceleration that seems faster than it actually is. This would also explain the overshoot that the measurements show in linear velocity. Generally the model seems to describe the linear motion in a satisfactory way.

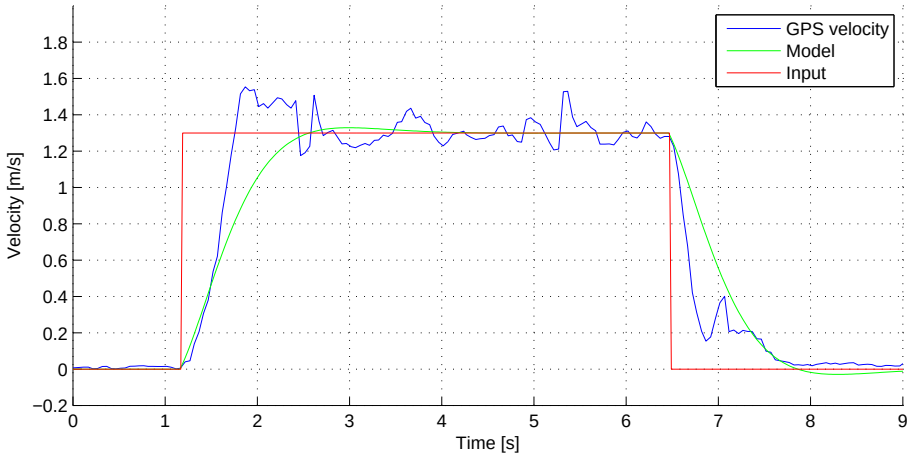


Figure 5.4. Comparison between model and measured output on a pulse in linear velocity.

The second test performed compares measurements and model of pure rotation with the robot. Since the parameters have been estimated from the forward velocity dynamics only, it must be verified that the model also shows the right behaviour when the robot turns. The measured data from figure 5.5 and 5.6 are both from the motion tracking system. A pulse of 2.96 rad/s and 0.33 rad/s is applied to the robot.

It is clearly seen that robot turns a lot less than expected. Further investigating of the data shows that the measured rotational velocity is approximately a constant factor of 0.53 less than what the model states in both tests. This seems to be cause by slip between

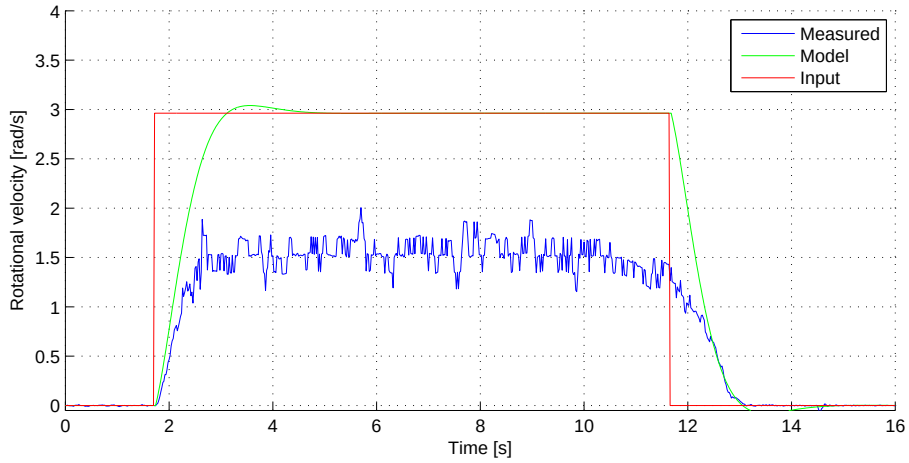


Figure 5.5. Comparison between model and measured output on a pulse in rotational velocity.

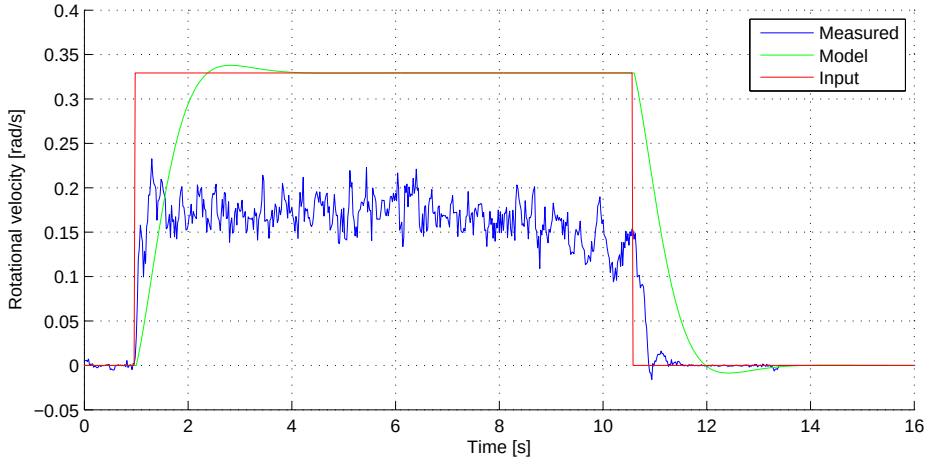


Figure 5.6. Comparison between model and measured output on a pulse in rotational velocity.

the wheels and ground surface. Since the dynamic model is based on the kinematic model where no slip is taken into consideration, the dynamic model does as well not consider slip. To compensate for this, the rotational velocity is multiplied by a factor of 1/0.53 before it is given as input to the robot. This makes sure that the robot turns as much as it is commanded to by the input. Except from this constant the model seems to behave in the same way as the robot. It should be noted however that the deceleration of the robot is faster than predicted by the model. This is likely to be cause by the extra friction that is present when the robot rotates around its own axis.

The third test that is performed on the model looks into how precise the model estimates the behaviour of the robot when both linear and rotational velocity are applied to the robot at the same time. To compensate for the slip detected in the previous test, the rotational input is multiplied by the factor stated above and left out of the plots. The position and velocity of the robot was measured by the RTK GPS while the angular velocity was measured with the optical gyro.

Figure 5.7 shows the linear velocity of the robot during the test. This is kept constant at 1 m/s during the whole run. As seen in figure 5.4 the acceleration is faster than

estimated by the model and it also overshoots by approximately 0.2 m/s before it settles on the reference speed. At time 6.5 s to 8.5 s a large deviation can be observed in the velocity. It looks like the linear velocity increases and then decreases again without any change in the input. This is caused by the placement of the GPS antenna in the left back corner of the vehicle. More information about this can found in appendix A . This location of the antenna causes the position of the vehicle to look in a certain way when plotted during a drive. When the robot is turning left the antenna will take a shorter path around the corner than the centre of the vehicle. The opposite happens when the robot turns right. This is clearly seen in figure 5.9 where the two turns seems to have different radius although the input is the rotational velocity input is the same. The seemingly larger turning radius of the vehicle results in the added velocity seen in figure 5.7.

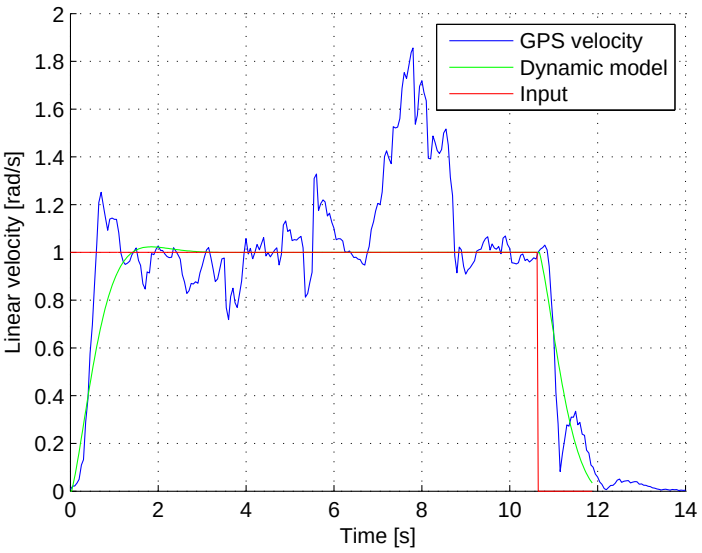


Figure 5.7. Comparison of the estimated and measured linear velocity of the robot.

Looking into figure 5.8 shows that the model struggles to keep up with the rotational velocity. It seems that the model does react as fast as the robot. It is expected to see that the model does not fit as well as the linear acceleration since the model estimation was only made from data in a forward acceleration. This could be caused by a too large moment of inertia. If this calculated constant is too large, it will slow down the dynamics of the model too much. It is however chosen not to change this parameter since this will have most effect while changing the rotational input rapidly. While following a trajectory the input should in general be smoother and not have the characteristics of a step input.

In figure 5.9 the position of the robot is plotted during the experiment. The robot accelerates forward from a standstill in point $(0, 0)$ in the offset UTM coordinate system and performs two successive turns. It should be noted that in this figure the pure kinematic model has also been plotted in order to compare the performance of this with the dynamic model. Overall the dynamic model estimates the position quite well. In comparison with the kinematic model the dynamic model gives an overall better result. The biggest

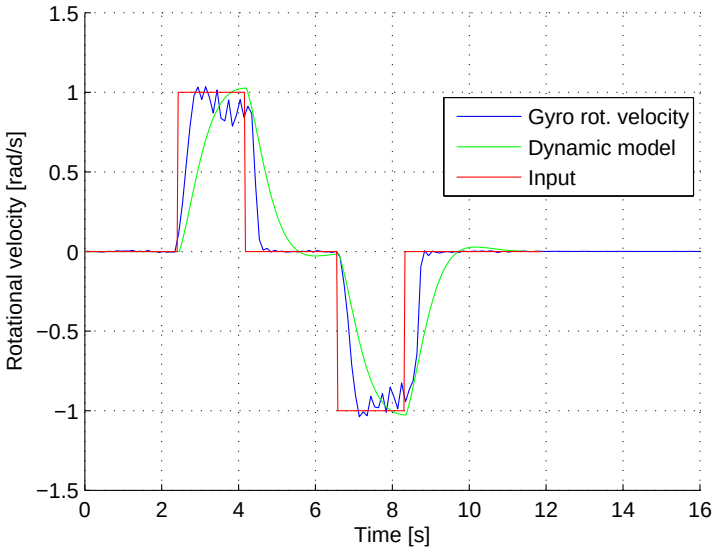


Figure 5.8. Comparison of the estimated and measured rotational velocity of the robot.

difference between the two models appears in the corners where the dynamic model seems to perform better. Some of the deviation seen in figure 5.9 is caused by the placement of the GPS as described earlier. Taking the GPS placement into account the differences between the dynamic model and the measurements is considered to be within acceptable limits.

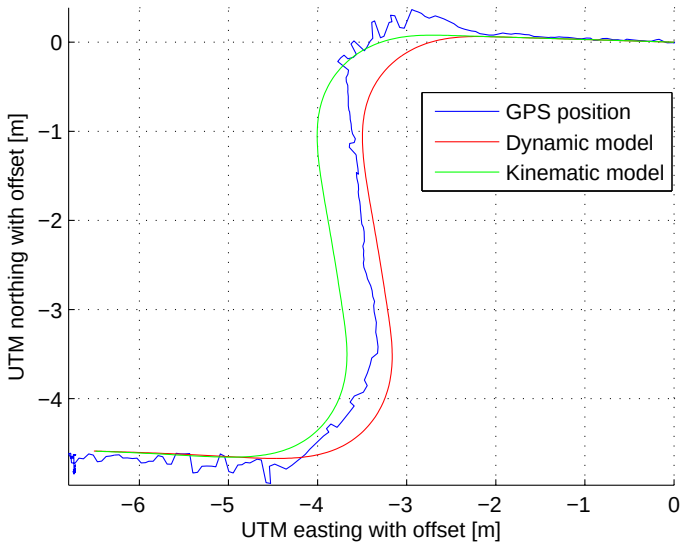


Figure 5.9. Comparison of estimated position from model and GPS data from robot.

The fourth and final test is a long feed forward drive where the input to both the model and robot is a control signal generated by a human operator through a PlayStation 3 controller. In opposite of test three the input in this test is smoother. The result of this test can be seen in figure 5.10. Again the robot starts at $(0, 0)$ in the offset UTM coordinate system. As expected the deviation between the model and the robot becomes much more apparent

when the system is fed forwarded for a longer period of time. Especially interesting is the area around $(-25, -17)$, where the model starts to diverge completely from the recorded GPS position. The reason for this is that the RobuROC4 has a build in limitation that prevents the motors that drives the wheels from overheating. When the robot has to turn on the asphalt a lot of torque is needed due to the four wheel skid steering principle. When the robot is asked to turn on a hard surface for a longer period of time the RobuROC4 enters a safety mode to protect the electric motors. This mode almost entirely prevents the robot from turning. To solve this problem the driving should altogether be kept as smooth as possible to avoid this limitation.

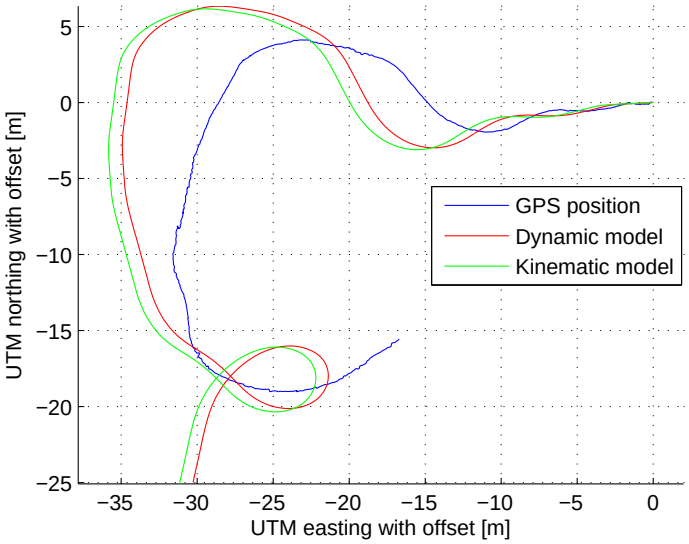


Figure 5.10. Comparison of estimated position from model and GPS data from robot.

From the tests analysed above, it is concluded that the model and its parameters found in this chapter is acceptable for controller and filter design.

6 Controller

In this chapter a controller used for following a trajectory will be defined. First the overall solution to the problem of tracking a trajectory is discussed. The dynamic model is hereafter included and the controller is designed. Lastly simulation results are presented to show the performance of the controller.

The overall way of tracking a trajectory is inspired from Schioler et al. [19]

In chapter 5 the angle of the vehicle was represented as an angle in radians. In this chapter a slightly different approach is used. Instead of an angle, the heading is represented as a unit direction vector: $d = [\cos(\theta) \ \sin(\theta)]^T$. The kinematic model for the vehicle is then as follows:

$$\dot{p} = d\,v \tag{6.1}$$

$$\dot{d} = R\,d\,\omega \tag{6.2}$$

Where:

p is the position of the vehicle in the global coordinate system

d is the unit direction vector describing the heading of the vehicle

v is the linear velocity in the longitudinal direction of the vehicle

ω in the angular velocity of the vehicle

R is a 90° counter clockwise rotation matrix

A fictitious reference robot, with a kinematic model identical to the vehicle model, is following a trajectory $\tau(t)$ (see figure 6.1). This trajectory is defined by its position in the global UTM frame, an angle, a rotational, and a linear velocity for each time instance.

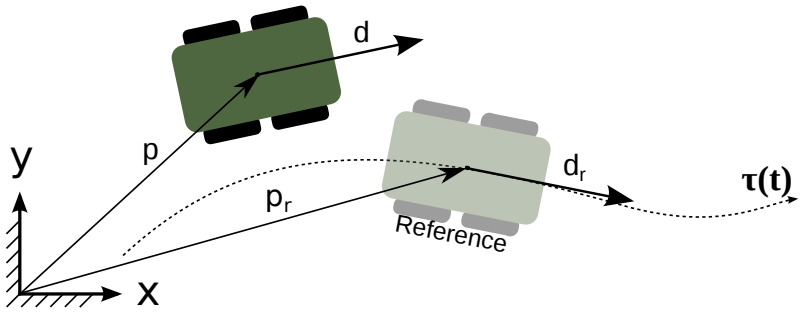


Figure 6.1. The trajectory following problem.

It is desired to define the position of the robot with respect to the reference position to be able to use a controller to minimise this error and control the vehicle onto the trajectory. The first step in this is to calculate the difference between the derivative of the states of

the vehicle and the reference.

$$\dot{p} - \dot{p}_r = d\,v - d_r\,v_r \tag{6.3}$$

$$\dot{d} - \dot{d}_r = R\,(d\,\omega - d_r\,\omega_r) \tag{6.4}$$

Since it is desired to have a linear system for control purposes equation (6.3) and (6.4) are linearised using the reference robot as operating point:

$$\begin{aligned}\dot{p} - \dot{p}_r = f(d,v) &\approx f(d_r,v_r) + (d-d_r)\left.\frac{\partial f(d,v)}{\partial d}\right|_{\substack{d=d_r\\v=v_r}} + (v-v_r)\left.\frac{\partial f(d,v)}{\partial v}\right|_{\substack{d=d_r\\v=v_r}} \\ &= (d-d_r)\,v_r + (v-v_r)\,d_r \end{aligned} \tag{6.5}$$

$$\begin{aligned}\dot{q} - \dot{q}_r = g(d,\omega) &\approx g(d_r,\omega_r) + (d-d_r)\left.\frac{\partial f(d,\omega)}{\partial d}\right|_{\substack{d=d_r\\\omega=\omega_r}} + (\omega-\omega_r)\left.\frac{\partial f(d,\omega)}{\partial \omega}\right|_{\substack{d=d_r\\\omega=\omega_r}} \\ &= R\,(d-d_r)\,\omega_r + R\,d_r\,(\omega-\omega_r) \end{aligned} \tag{6.6}$$

The errors in position and direction with respect to the global coordinate system are calculated and projected onto the coordinate axis of the reference coordinate system (see figures 6.2 and 6.3). The errors in the reference coordinate system are calculated as:

$$p_{ex} = (p - p_r)^T\,d_r \tag{6.7}$$

$$p_{ey} = (p - p_r)^T\,R\,d_r \tag{6.8}$$

$$d_{ex} = (d - d_r)^T\,d_r \tag{6.9}$$

$$d_{ey} = (d - d_r)^T\,R\,d_r \tag{6.10}$$

The arrows marked with red and blue colour in figure 6.3 are duplicates of arrows elsewhere in the figure. They have just been offset to illustrate how the difference between the direction of the real robot and the reference robot is calculated and to show how the direction difference error is projected onto the coordinate axis of the reference robot coordinate system.

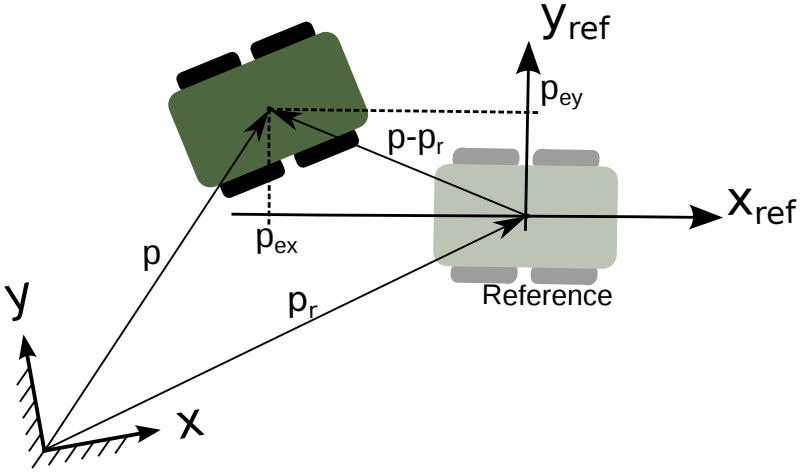


Figure 6.2. Illustration showing the position errors p_{ex} and p_{ey}.

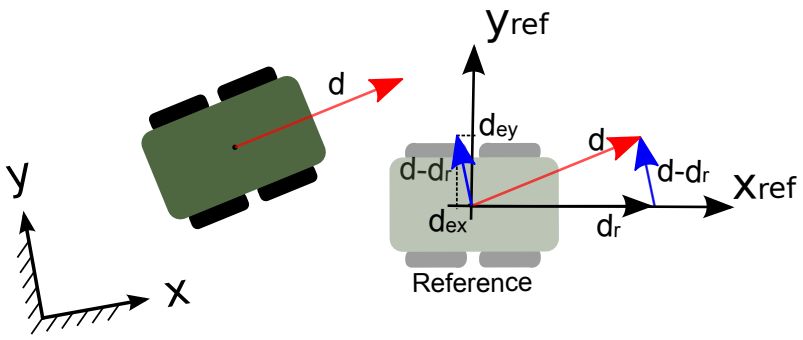


Figure 6.3. Illustration showing the direction errors d_{ex} and d_{ey}.

To investigate how the calculated errors changes with respect to time, differentiation with respect to time of equations (6.7)- (6.10) is carried out. The linearised difference between the change in position and the change in direction given in equations (6.5) and (6.6) is used to calculate the derivative of the errors.

$$\begin{aligned}
\dot{p}_{ex} &= (\dot{p} - \dot{p}_r)^T d_r + (p - p_r)^T \dot{d}_r \\
&= v_r (d - d_r)^T d_r + (v - v_r) d_r^T d_r + (p - p_r)^T R d_r \omega_r \\
&= v_r d_{ex} + v - v_r + p_{ey} \omega_r
\end{aligned} \tag{6.11}$$

$$\begin{aligned}
\dot{p}_{ey} &= (\dot{p} - \dot{p}_r)^T R d_r + (p - p_r)^T R \dot{d}_r \\
&= v_r (d - d_r)^T R d_r + (v - v_r) d_r^T R d_r + (p - p_r)^T R R d_r \omega_r \\
&= v_r (d - d_r)^T R d_r + (v - v_r) 0 - (p - p_r)^T d_r \omega_r \\
&= v_r d_{ey} - p_{ex} \omega_r
\end{aligned} \tag{6.12}$$

$$\begin{aligned}
\dot{d}_{ex} &= (\dot{d} - \dot{d}_r)^T d_r + (d - d_r)^T \dot{d}_r \\
&= (R (d - d_r))^T d_r \omega_r + (R d_r)^T d_r (\omega - \omega_r) + (d - d_r)^T R d_r \omega_r \\
&= (R R (d - d_r))^T R d_r \omega_r + (d - d_r)^T R d_r \omega_r \\
&= -(d - d_r)^T R d_r \omega_r + (d - d_r)^T R d_r \omega_r \\
&= 0
\end{aligned} \tag{6.13}$$

$$\begin{aligned}
\dot{d}_{ey} &= (\dot{d} - \dot{d}_r)^T R d_r + (d - d_r)^T R \dot{d}_r \\
&= (R (d - d_r))^T R d_r \omega_r + (R d_r)^T R d_r (\omega - \omega_r) + (d - d_r)^T R R d_r \omega_r \\
&= (R R (d - d_r))^T R R d_r \omega_r + (\omega - \omega_r) - (d - d_r)^T d_r \omega_r \\
&= (d - d_r)^T d_r \omega_r + (\omega - \omega_r) - (d - d_r)^T d_r \omega_r \\
&= \omega - \omega_r
\end{aligned} \tag{6.14}$$

These four equations are directly put into state space form yielding:

$$\begin{bmatrix} \dot{p}_{ex} \\ \dot{p}_{ey} \\ \dot{d}_{ex} \\ \dot{d}_{ey} \end{bmatrix} = \begin{bmatrix} 0 & \omega_r & v_r & 0 \\ -\omega_r & 0 & 0 & v_r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{ex} \\ p_{ey} \\ d_{ex} \\ d_{ey} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v - v_r \\ \omega - \omega_r \end{bmatrix} \tag{6.15}$$

From the state space form it is seen that the state d_{ex} cannot directly be modified and thereby be minimised. It will however converge to zero when the other part of the direction error d_{ey} converges to zero. To minimise the errors the total system is divided into a propulsion and a direction controller, where the propulsion controller minimises the position error in the x-direction p_{ex}, and the direction controller minimises the direction error d_{ey} and the position error p_{ey}.

To minimise the position error in the x-direction p_{ex} a proportional controller with gain K_a is used by defining $\dot{p}_{ex} = -K_a\, p_{ex}$. By inserting this into equation (6.11) the control signal v is found to be:

$$v = -K_a\, p_{ex} - v_r\, d_{ex} + v_r - p_{ey}\, \omega_r \tag{6.16}$$

For the propulsion controller the dynamics of the robot is disregarded since it is considered that the linear velocity of the robot will not change a significantly while running. The reason for this is that the trajectories will be defined such that the reference velocity will remain constant.

Having reduced the position error in the x-direction p_{ex} to zero the state space form in equation (6.15) can be reduced to a state space form only including the position error in the y-direction p_{ey} and the direction error d_{ey}, which can be minimised by using a direction controller controlling the rotational velocity of the robot ω:

$$\begin{bmatrix} \dot{p}_{ey} \\ \dot{d}_{ey} \end{bmatrix} = \begin{bmatrix} 0 & v_r \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_{ey} \\ d_{ey} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} \omega - \omega_r \end{bmatrix} \tag{6.17}$$

Since the robot should not have a constant angular velocity the dynamics of the robot is included in the system description. Including the rotational velocity part of the state space form given in equation (5.23) the state space model of the direction errors expands to:

$$\begin{bmatrix} \dot{p}_{ey} \\ \dot{d}_{ey} \\ \ddot{\theta} \\ \dot{q}_1 \end{bmatrix} = \begin{bmatrix} 0 & v_r & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -a2 & -a1 \end{bmatrix} \begin{bmatrix} p_{ey} \\ d_{ey} \\ \dot{\theta} \\ q_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ b0 \\ b1 - a1\, b0 \end{bmatrix} \begin{bmatrix} \omega - \omega_r \end{bmatrix} \tag{6.18}$$

As seen in 6.18 is it chosen to continue using the difference between the input rotational velocity to the robot and the rotational velocity of the reference robot as input to the state space model. It is chosen to do it this way to simplify the model. When making this simplification it is assumed that the trajectory is defined smooth enough such that the dynamics do not make significant impact on the trajectory reference signal.

Having set up the model in this way, the state feedback with feedback gains K_1 to K_4 is used to define the control signal ω as:

$$\omega = \begin{bmatrix} K_1 & K_2 & K_3 & K_4 \end{bmatrix} \begin{bmatrix} p_{ey} & d_{ey} & \dot{\theta} & q_1 \end{bmatrix}^T + \omega_r \tag{6.19}$$

6.1 Controller Design

Having designed a controller structure for both propulsion and direction, the controller gains must be chosen appropriately. In this section both controller gains for the propulsion and the direction controllers will be settled.

As seen in the model describing the change of direction (6.18), the dynamics depend on the reference linear velocity. Therefore a reference forward velocity must be settled before the design of the state feedback can be started. From the Requirements Specification in chapter 3, requirement **r.2** states that the average driving speed should be at least 7 km/h (≈ 1.94 m/s). To have some margin to this requirement, it is chosen to use a linear velocity reference (v_r) at 2.2 m/s.

Since the controller must be implemented on a computer the model is discretised. The sampling time is chosen from the rule of thumb stating that the sampling frequency should be 20 to 40 times the bandwidth of the system [20, p. 599]. From the model of the directional dynamics (see equation (6.18)) the maximum bandwidth is found to be around 0.7 Hz. Sampling with 30 times the bandwidth gives a sampling frequency of 21 Hz and the sampling frequency is therefore chosen as 20 Hz, corresponding to a sampling time Ts of 0.05 s.

Discretisation of the state space model in equation (6.18) using the Euler discretisation method yields:

$$q[k+1] = A\,q[k] + B\,u[k] \tag{6.20}$$

Where:

$$q[k] = \begin{bmatrix} p_{ey}[k] & d_{ey}[k] & \dot{\theta}[k] & q_1[k] \end{bmatrix}^T$$

$$u[k] = \begin{bmatrix} \omega[k] - \omega_r[k] \end{bmatrix}^T$$

$$A = \begin{bmatrix} 1 & v_r\,Ts & 0 & 0 \\ 0 & 1 & Ts & 0 \\ 0 & 0 & 1 & Ts \\ 0 & 0 & -a2\,Ts & 1 - a1\,Ts \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ b0\,Ts \\ (b1 - a1\,b0)\,Ts \end{bmatrix}$$

First the gain K_a for the propulsion controller is determined. From simulations and tests the gain is settled to be 2. This gives a good compromise between minimising the direction error in the x-direction p_{ex} without changing linear velocity much when the error is small giving a smooth linear velocity.

For the direction controller the first estimate of the gains is determined by solving the LQR problem by minimising the cost function:

$$J = \sum_{k=0}^{\infty} q[k]^T\,Q\,q[k] + u[k]^T\,R\,u[k]$$

by calculating the optimal feedback gain K when defining the system input as:

$$u[k] = -K\,x[k]$$

In the cost function the Q and R matrices specifies the penalty for having states and inputs different from zero respectively. The penalties in the two matrices are defined using Bryson's rule [20, p. 481] and is calculated from the maximum acceptable values of the states and inputs respectively as:

$$Q_{ii} = 1/\text{square of the maximum acceptable value of the i'th state}$$
$$R_{ii} = 1/\text{square of the maximum acceptable value of the i'th input}$$

In table 6.1 the maximum acceptable value for the states and inputs are given along with the calculated matrix entry.

State / Input	Maximum acceptable value	Penalty matrix entrance
p_{ey}	$0.1\,\text{m}$	$Q_{11} = 100\,\frac{1}{\text{m}^2}$
d_{ey}	$10\,^\circ \approx 0.17\,\text{rad}$	$Q_{22} = 33.16\,\frac{1}{\text{rad}^2}$
$\dot{\theta}$	$3.5\,\text{rad/s}$	$Q_{33} = 0.08\,\frac{1}{\text{rad}^2/\text{s}^2}$
q_1	∞	$Q_{44} = 0-$
$\omega - \omega_r$	$0.1\,\text{rad/s}$	$R_{11} = 100\,\frac{1}{\text{rad}^2/\text{s}^2}$

Table 6.1. Penalties for the Q and R matrices used when calculating feedback gains.

The maximum acceptable values are chosen by reasoning of what type of error is most critical. The maximum value for $\dot{\theta}$ is chosen to be the maximum rotational velocity that the vehicle can obtain. The maximum acceptable value for the q_1 state is chosen to be infinity since the state is only introduced to model the rotational dynamics of the vehicle and reducing this state to zero is not the objective of the controller.

The maximum acceptable value for the input $(\omega - \omega_r)$ is chosen small to limit the input to the vehicle. By doing so the movement of the vehicle will be smoother, as it is desired. A rotational velocity of $0.1\,\text{rad/s}$ could seem small, but taking into account that this is the difference from the reference, it is considered reasonable as long as the ARS stays close to the trajectory.

Having defined the penalty matrices, the LQR problem is solved in MATLAB® using the discrete LQR function which takes the system and penalty matrices as inputs and returns the optimal gains: `K = dlqr(A,B,Q,R)`. For the defined matrices the optimal gains are found to be:

$$K = \begin{bmatrix} 0.95 & 3.18 & 1.50 & 0.31 \end{bmatrix}$$

To show the performance of the controller a simulation has been conducted. In the simulation, a reference trajectory is generated using the kinematic model. The simulation uses the complete model defined in the State Space Representation section 5.2 to model the behaviour of the vehicle. In figure 6.4 the simulated position is shown together with the trajectory. The trajectory starts at position (0,0) and the vehicle starts at position

(1,-1). As seen in the figure the vehicle converges onto the trajectory after a few meters and stays close to the trajectory until the endpoint is reached. In the turns the vehicle does however diverge from the trajectory.

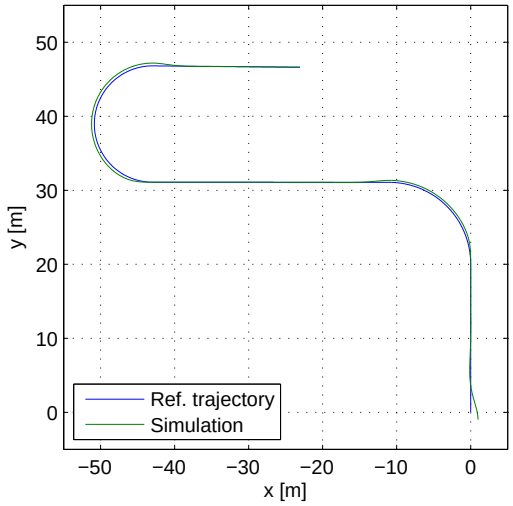


Figure 6.4. Simulation of position of the vehicle.

In figure 6.5 the values of the four error signals defined in equations (6.7) - (6.10) are shown. In the beginning large errors are present which is expected since the vehicle is not on the trajectory at this time. A requirement for the direction controller to be valid is that the position error in the x-direction p_{ex} converges to zero which is the case as seen on the figure. The reason why the magnitude of the p_{ex} error increases in the beginning is due to the dynamics of the vehicle. While the kinematic model from which the trajectory is generated accelerates at full speed instantaneously, it takes some time for the vehicle to reach max speed which implies that the trajectory outruns the vehicle. When the vehicle exceeds 2.2 m/s it starts to catch up with the trajectory. This does however take about 8 s since the robot does not have much extra speed compared to the 2.2 m/s at which the trajectory is defined.

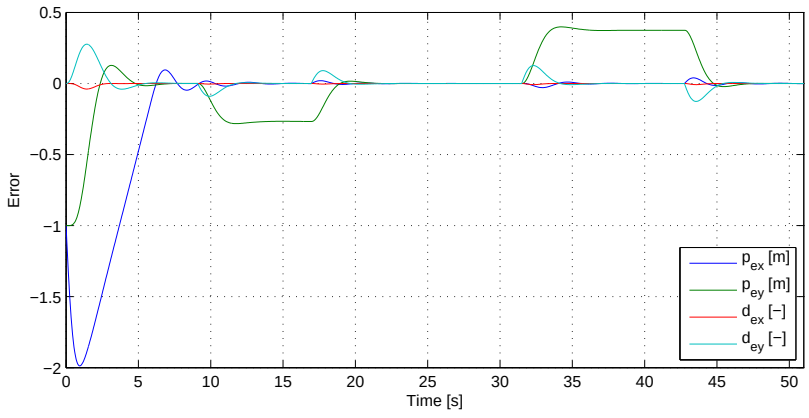


Figure 6.5. Errors during the simulation.

From the figure it is seen that the position error p_{ey} converges to zero directly. Doing so implies that the vehicle turns and the direction error d_{ey} does therefore increase for a short while until the position error in the y-direction p_{ey} is reduced. In figure 6.5 it is easily seen that the vehicle diverges from the trajectory in the turns since the position error in the y-direction p_{ey} increases to around 0.25 m in the first turn and 0.4 m in the second.

From this it is seen that the controller structure is capable of controlling the model of the vehicle to follow a trajectory. It is however desired to reduce the position error in the y-direction. To do so the gains are manually changed. While keeping the gain for the propulsion controller the gains for the direction controller are chosen to be:

$$K = \begin{bmatrix} 1.00 & 3.10 & 0.90 & 0.00 \end{bmatrix}$$

The two first gains are almost kept as determined in the LQR controller. The third gain is reduced to approximately be the half the original value. By reducing this gain, the direction error is considered less critical and the position error should therefore decrease. The fourth gain is set to zero since this showed a more stable behaviour in the simulations.

In figure 6.6 and 6.7 the overall position and the controller errors are shown for a simulation with the modified controller. Compared to the LQR controller the modified controller shows significantly lower errors in the turns. The position error in the y-direction is determined to be less than 0.2 m in the first turn, and around 0.25 m in the second turn.

These errors can seem large compared to requirement **r.1** which states that all stripes must be placed within ±5 cm of the reference. The turning radius of the two turns is however 10 m and 7.5 m which is less than the radius of turns on highways and motorways. These errors will therefore be smaller on real roads with wider turns and are therefore considered less critical.

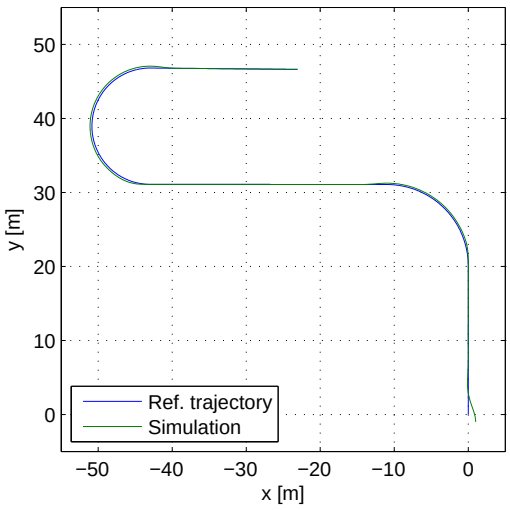


Figure 6.6. Simulation of position of the vehicle using the modified controller.

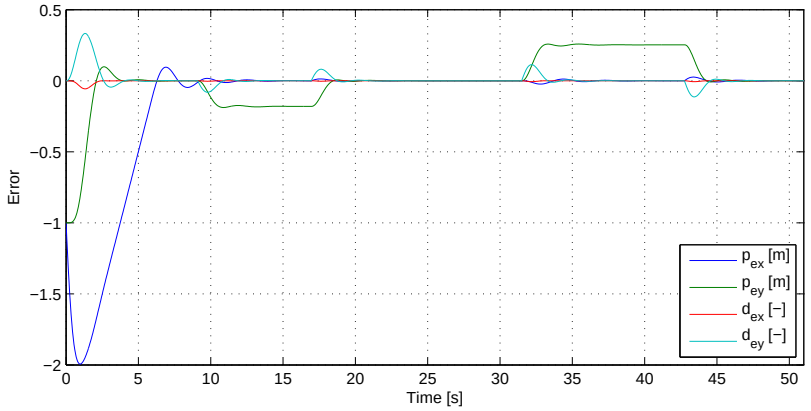


Figure 6.7. Errors during the simulation using the modified controller.

The stability of the system with feedback is analysed by calculating the eigenvalues. Firstly the system matrix for the system with feedback is calculated as:

$$A_{fb} = A - B\,K \tag{6.21}$$

Where A and B are the system matrices for the discrete system defined in equation (6.20) and $K = \begin{bmatrix} 1.00 & 3.10 & 0.90 & 0.00 \end{bmatrix}$. The eigenvalues of this matrix are calculated using the `eig()` command in MATLAB® to be:

$$0.9487 + j0.1186$$
$$0.9487 - j0.1186$$
$$0.9407 + j0.0360$$
$$0.9407 - j0.0360$$

Since the eigenvalues of the system with feedback are inside the unit circle the system is stable. Through the simulation of the system with feedback and from the analysis of the eigenvalues of the system with feedback it seems like a propulsion controller with gain $K_a = 2$ and a direction controller with gains $K = \begin{bmatrix} 1.00 & 3.10 & 0.90 & 0.00 \end{bmatrix}$ is a good choice for controlling the vehicle onto a trajectory. A good compromise between stability, small errors and smooth driving seems to be achieved in the simulations.

6.2 Controller Test

To evaluate the performance of the controller a test has been conducted. The used trajectory is the same as for the simulations in the controller design. During this test the sensor inputs are filtered as described in chapter 7.

In figure 6.8 the overall path of the vehicle is plotted along with the trajectory. It is seen that the vehicle is capable of following the trajectory without major deviation except at the end around position (-32,45). This large deviation is caused by the GPS that makes a jump of around 1 m which the controller cannot account for.

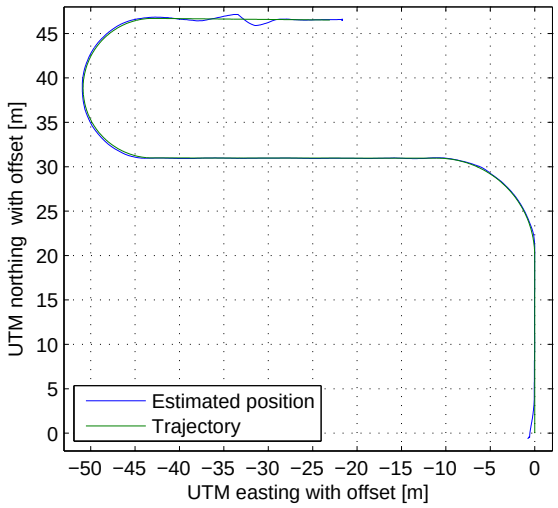


Figure 6.8. Estimated position and trajectory for a controller test.

During the test the controller errors were logged. These are plotted in figure 6.9. From the figure it is seen that the overall shape comply with the simulations where larger errors are observed when turning compared to driving straight. The errors while turning is however generally less than for the simulation while they are larger when driving straight.

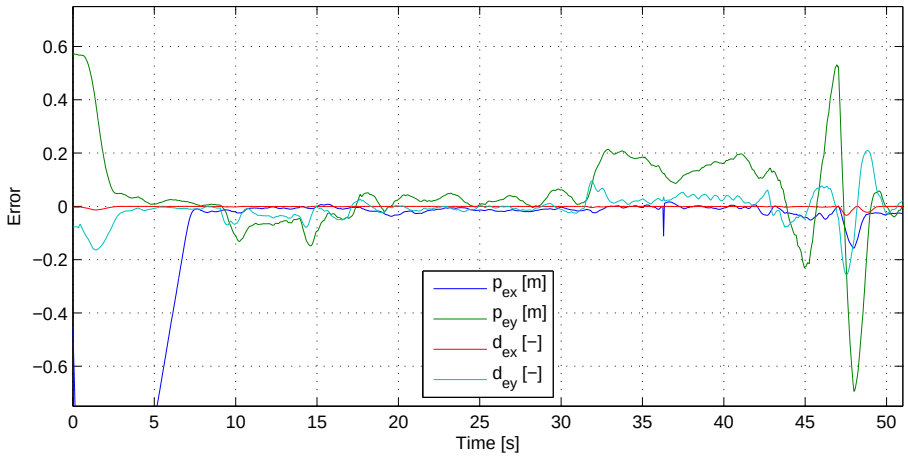


Figure 6.9. Controller errors logged in the test.

For the straight parts the error in the transverse direction is generally less than $\pm 0.05\,\mathrm{m}$. A similar test with the same controller shows better results when decreasing the linear velocity. It could therefore be beneficial to decrease the velocity when a very good precision is needed.

During this test a video was captured to show the performance. This video can be found on the CD `/videos/feed_fwd_test.mp4`

7 Sensors and Sensor Fusion

This chapter contains a description of how the sensor fusion is carried out. First an overview of the available sensors and what they can be used for is presented. Each sensor is hereafter discussed. The precision of each sensor is determined, and modifications made to the raw sensor input is described. At last the sensor fusion is described. Inputs from all the sensors are fused using an extended Kalman filter and the robot model found in chapter 5.

7.1 Sensors

As described in the introduction to the design, a variety of sensors are provided to be able to determine the position, heading and velocity of the vehicle. The composition of the provided hardware is described in appendix A. In table 7.1 the provided sensors are listed along with the system output they can measure. The measurements marked with parentheses indicates that the values can be measured but the measurement are considered too unreliable to be used (see section 7.1.4).

		Measurement			
Sensor type	**Sensor model**	**Position** (x, y)	**Heading** (θ)	**Linear vel** (v)	**Angular vel** (ω)
GPS	Ashtec MB100	X	X	X	
Magnetometer	OS5000		X		
Gyro	E•Core 2000				X
Vehicle Odometry	RobuROC4	(X)	(X)	X	X

Table 7.1. Sensors and their use.

In the following subsections the sensors are described one by one. This includes a precision analysis made for each sensor. The main purpose for the precision analysis is to examine how much to rely on each sensor in the sensor fusion (see section 7.2).

7.1.1 GPS

The overall position of the vehicle is determined by an Ashtech MB100 GPS /literature/hardware_manuals/Ashtec_MB100_Reference_Manual.pdf. This GPS uses RTK technology to determine the position of the vehicle with a precision down to 2 cm. In addition the GPS is capable of determining the velocity and heading of the vehicle. The precision of the heading measurement, however relies on the velocity of the vehicle since the heading is calculated in the GPS by comparing successive samples.

The GPS provides the position in latitude and longitude components in degrees and minutes. Since these are not directly usable in a Cartesian coordinate system the position

is converted into to the UTM coordinate system.

The heading from the GPS has 0° at north and positive direction clockwise. It is desired to have 0° at east and positive direction counter clockwise. The heading is therefore reversed by negating and the 0° point is moved by adding an offset of 90°. Lastly the heading is converted into radians:

$$\theta_{\text{GPS}} = (90^\circ - \tilde{\theta}_{\text{GPS}}) \frac{\pi}{180}$$

Where:

$\tilde{\theta}_{\text{GPS}}$ Is the heading measurement directly from the GPS

θ_{GPS} Is the modified heading used as sensor input

The last conversion made to the data from the GPS is that the velocity is converted from knots to m/s by multiplying by a factor of $0.514 \frac{\text{m/s}}{\text{knot}}$.

To determine the precision of the measurements in terms of variance, a test is conducted. A complete description of this test is given in appendix B. The variance of the position is calculated from the positions which are logged while the GPS is kept at a stand still. For the heading and velocity the vehicle moves on a straight line with constant velocity. The heading and velocity are logged and the variance is calculated. The determined variances are shown in table 7.2

Linear velocity $[m/s]$	$\sigma^2_{\text{pos}_{\text{GPS}}}$ [m^2]	$\sigma^2_{\theta_{\text{GPS}}}$ [rad^2]	$\sigma^2_{v_{\text{GPS}}}$ [m^2/s^2]
0.00	$3.253 \cdot 10^{-3}$		
0.25		$37.772 \cdot 10^{-3}$	$1.2387 \cdot 10^{-3}$
0.50		$23.791 \cdot 10^{-3}$	$2.3657 \cdot 10^{-3}$
0.75		$34.613 \cdot 10^{-3}$	$7.0995 \cdot 10^{-3}$
2.20		$2.721 \cdot 10^{-3}$	$3.6187 \cdot 10^{-3}$

Table 7.2. Variances for the GPS.

As seen in the table the variance of the heading measurements depends on the velocity. For high linear velocities the variance decreases to around $3 \cdot 10^{-3}\text{rad}^2$, whereas for velocities below 0.75 m/s the variance is a factor of about 10 larger. This is not surprising since the heading is calculated from the GPS by comparing successive samples and the higher the velocity is the longer the distance between two successive samples is. For the velocity measurement the variance seems quite stable. An interesting observation is that it increases when the linear velocity increases up to 0.75 m/s whereas for large velocities this variance is smaller again. The reason for the smaller variance at high velocities is most likely the same as for the heading measurement, since the distance between two successive samples increases. The reason for the increasing of the variance for small velocities could be caused by the increasing disturbances caused by vibrations, which could have a greater influence than the increasing distance between the samples.

7.1.2 Magnetometer

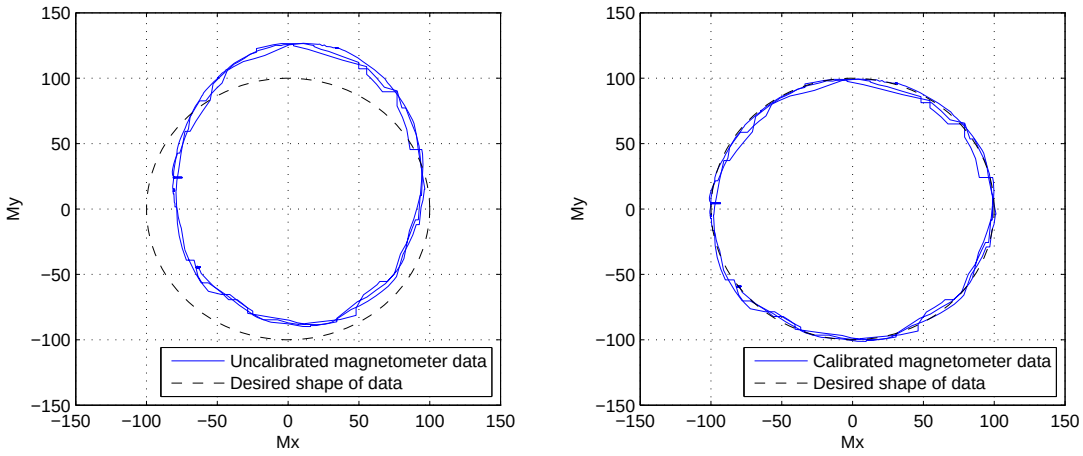
To measure the heading of the vehicle at all times a digital compass of type OS5000 is used /literature/hardware_manuals/OS5000_Compass_Manual.pdf. The compass can provide the heading directly by use of magnetometers and accelerometers. Some initial test did however show that the provided heading is very noisy. The noisy output is most likely caused by vibrations which affects some internal accelerometers. It is therefore chosen to calculate the heading directly from the magnetometer output only. When doing so, it is assumed that the magnetic field strength in z-direction of magnetometer is constant, hence the vehicle is not experiencing any significant pitch or roll.

The heading is calculated from the x and y magnetometer readings, called Mx and My respectively, by use of the arcustangent function:

$$\theta_{\text{magnetometer}} = \begin{cases} \arctan\left(\frac{My}{Mx}\right) & \text{for } Mx \geq 0 \wedge My \geq 0 \\ \arctan\left(\frac{My}{Mx}\right) + \pi & \text{for } Mx < 0 \\ \arctan\left(\frac{My}{Mx}\right) + 2\,\pi & \text{otherwise} \end{cases} \tag{7.1}$$

The direct output from the arcustangent function lies in the interval $[-\pi, \pi]$. This is converted into the interval $[0, 2\,\pi]$ by adding π and $2\,\pi$ as shown in equation (7.1).

Before the heading is calculated some modifications are made to the measurements to compensate for disturbances in the magnetic field. For a test run the raw sensor data is shown in figure 7.1(a). As seen in the figure the samples do not shape a perfect circle with centre at $(0, 0)$, which would be the case when the magnetic field is not disturbed. To compensate for this phenomenon a calibration is made. On a sample of data an ellipse is fitted and scaling factors and offsets are calculated to transform input data to be on a perfect circle. In figure 7.1(b) the data from figure 7.1(a) is shown after calibration.



(a) Uncalibrated data.

(b) Calibrated data.

Figure 7.1. Data sample from the magnetometer.

The heading calculated by use of equation (7.1) has 0 rad at north and positive direction clockwise. As with the GPS it is desired to have 0 rad at east and positive direction counterclockwise. In the variance tests described later it showed up that the magnetometer is turned slightly with respect to the vehicle, an offset of 0.05 rad is therefore subtracted. The calculated heading is therefore modified in the following way:

$$\theta_{\text{Magnetometer}} = \frac{\pi}{2} - \tilde{\theta}_{\text{Magnetometer}} - 0.05$$

Where:

$\tilde{\theta}_{\text{Magnetometer}}$ Is the heading calculated from the magnetometer readings

$\theta_{\text{Magnetomter}}$ Is the modified heading used as sensor input

To determine the precision of the measurements in terms of variance, a test has been conducted. A complete description of this test is given in appendix B. The vehicle is set to drive straight with constant linear velocity and zero angular velocity to obtain constant heading. The calculated heading is stored and the variance is calculated. The test was performed for four different linear velocities. The result is shown in table 7.3.

Linear velocity $[m/s]$	$\sigma^2_{\theta_{\text{Magnetometer}}}$ $[\text{rad}^2]$
0.25	$0.27722 \cdot 10^{-3}$
0.50	$0.98447 \cdot 10^{-3}$
0.75	$0.30863 \cdot 10^{-3}$
2.20	$0.68681 \cdot 10^{-3}$

Table 7.3. Variances for the Magnetometer.

As seen in the table the variance of the magnetometer measurements changes in the four tests with different linear velocity. These changes in the variance are not considered to have an effect on the changing linear velocity since it does not seem that the variance is either increasing or decreasing when the linear velocity does. It is therefore considered that the changes in the variances are caused by external disturbances in the tests.

7.1.3 Gyro

To determine the angular velocity of the vehicle a fibre optic gyro of model E•Core 2000 is used /literature/hardware_manuals/gyro/ecore2000.pdf. The gyro directly provides the angular velocity of the vehicle in [deg/s] which is converted into [rad/s] as the only modification to the sensor input.

To determine the precision of the angular velocity measurement, two tests are carried out. In the first test the vehicle is set to run with constant linear velocity and zero angular velocity. The data from the gyro is stored and the variance is calculated. In the second test the vehicle is set to have zero linear velocity and a constant angular velocity. The data from the gyro is again stored and the variance is calculated. The tests have been carried out for several different linear and angular velocities. The results are shown in table 7.4.

Linear velocity $[m/s]$	**Angular velocity** $[rad/s]$	$\sigma^2_{\omega_{\text{Gyro}}}$ $[\text{rad}^2/\text{s}^2]$
0.25	0	$12.986 \cdot 10^{-6}$
0.50	0	$15.173 \cdot 10^{-6}$
0.75	0	$8.996 \cdot 10^{-6}$
2.20	0	$21.705 \cdot 10^{-6}$
0	0.17	$25.198 \cdot 10^{-6}$
0	1.57	$68.810 \cdot 10^{-6}$

Table 7.4. Variances for the Gyro.

From the table it does not seem like that there is a connection between change in variance when the linear velocity changes. For the angular velocity the situation is different. For higher angular velocities the variance increases. The higher variance may be caused by the vibration caused by the skid between the wheels and the surface when turning on a spot.

7.1.4 Vehicle Odometry

From measurements on the wheels the RobuROC4 is capable of estimating the angular and linear velocities of the vehicle along with the heading and position relative to a defined start position /literature/hardware_manuals/Robusoft_RobuROC4_User_Manual_including_communication_protocol.pdf. As stated in table 7.1 the position and heading measurements are considered unreliable. The reason for this is that the heading and position are calculated solely based on wheel measurements and in presence of skid the heading will be faulty which will cause the position estimate to be faulty.

The current linear and angular velocities are directly read from the vehicle in m/s and rad/s respectively. The first modification made to these measurements is that they are multiplied by a constant scaling factor of 1.3 . This scaling factor is needed because the vehicle is ordered with larger wheels than standard which has not been corrected in the software. From simple tests it is shown that the vehicle converts the measurements on the wheels to linear and angular velocity of the vehicle in correspondence with the kinematic model found in chapter 5. As for the kinematic model found in the modelling chapter, the vehicle does not take any skid into account. To include the loss of rotational velocity due to skid a constant factor of 0.53 is multiplied to the angular velocity measurement. The factor is identified during the model validation in section 5.4.

To determine the precision in terms of variance of the linear and angular velocity measurements, a test has been performed. For a complete test description see appendix B. For the linear velocity measurement, the vehicle is set to run with a constant linear velocity. The velocity is logged and the variance is calculated. For the angular velocity measurement the vehicle is set to drive with a constant angular velocity. The angular velocity is saved and the variance is calculated. The results are shown in table 7.5.

As seen in the table the variance of the measurements for both the angular velocity and the linear velocity increases when the angular velocity or the linear velocity increases.

Linear velocity $[m/s]$	**Angular velocity** $[rad/s]$	$\sigma^2_{v_{\text{odometry}}}[\text{m}^2/\text{s}^2]$	$\sigma^2_{\omega_{\text{odometry}}}[\text{rad}^2/\text{s}^2]$
0.25	0	$18.989 \cdot 10^{-6}$	$7.618 \cdot 10^{-6}$
0.50	0	$36.540 \cdot 10^{-6}$	$13.112 \cdot 10^{-6}$
0.75	0	$33.552 \cdot 10^{-6}$	$13.458 \cdot 10^{-6}$
2.20	0	$41.389 \cdot 10^{-6}$	$52.646 \cdot 10^{-6}$
0	0.17	-	$37.459 \cdot 10^{-6}$
0	1.57	-	$1.092 \cdot 10^{-3}$

Table 7.5. Variances for the vehicle odometry measurements.

7.2 Sensor Fusion

To be able to use a controller that controls the vehicle to drive to a desired position or along a desired trajectory, the current position and heading of the vehicle must be known. As described in the previous section both the position and heading of the vehicle are directly measured. These measurements do however include disturbances from measurement noise. To get a more precise estimate of the current states of the vehicle a Kalman filter is used.

The main idea of the Kalman filter is that the states at the current time instant is estimated by using the model of the plant, inputs to the plant, and the previous state. Afterwards the estimate of the states are updated by using measurements. The ordinary Kalman filter requires the plant to be linear, which the model of the RobuROC4 is not. To overcome this limitation an extended Kalman filter is used instead. In the extended Kalman filter the nonlinear model of the plant is used directly to estimate the new states. In the update step the plant is linearised using the previous states as operating point. This linearised version of the plant is used to update the states including the covariance matrix describing the precision of the states.

In the Extended Kalman filter the new states and the outputs of the system are predicted by using the nonlinear model found in section 5.2:

$$\hat{q}_k^- = f_k(\hat{q}_{k-1}^+, u_{k-1}) \tag{7.2}$$

$$\hat{y}_k = h_k(\hat{q}_k^-) \tag{7.3}$$

Where:

$\hat{q}_k^-$ is the predicted states before update

$\hat{q}_{k-1}^+$ is the previous states after measurement update

u_{k-1} is the system input

$\hat{y}_k$ is the predicted output

f_k is the nonlinear system model

h_k is the nonlinear output model

The linearised system matrix (A) and linearised output matrix (C) are hereafter found by

calculating the Jacobian matrix and inserting the previous states as operating point.

$$A_{k-1} = \left.\frac{\partial f_{k-1}}{\partial q}\right|_{q=\hat{q}_{k-1}^{+}} \tag{7.4}$$

$$C_k = \left.\frac{\partial h_k}{\partial q}\right|_{q=\hat{q}_k^{-}} \tag{7.5}$$

The covariance is propagated by using the linearised system matrices:

$$P_k^- = A_{k-1} P_{k-1}^+ A_{k-1}^T + Q_{k-1} \tag{7.6}$$

Where:

P_k^- is the predicted covariance of the prediction of the current states

P_k^+ is the covariance of the previous states

Q_{k-1}^+ is a covariance matrix describing the precision of the model

At this point the system model has been used to the generate a prediction of the current state based on the previous states and the system inputs. In addition the covariance after the current prediction is calculated. The next step is to use the measurements to update the prediction. First the Kalman gain is calculated as:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{7.7}$$

Where:

R_k is covariance matrix describing the precision of the measurements

The states are hereafter updated using the calculated Kalman gains and the difference between the predicted outputs and the measured output:

$$\hat{q}_k^+ = \hat{q}_k^- + K_k(y_k - \hat{y}_k) \tag{7.8}$$

Where:

y_k is the current measurement vector

At last the covariance matrix is updated:

$$P_k^+ = (I - K_k H_k) P_k^- \tag{7.9}$$

In the design of the filter the design parameters are the initial states $\hat{q}_0^+$, initial covariance P_0^+, model covariance Q_k and measurement covariance R_k. In addition the sampling frequency must be determined. The initial state could be set in several ways either using measurements from the sensors directly and setting low variances or having an arbitrary initial state with very high variance such that the measurements will change the estimate of the states towards the measurements fast.

The variances defined in the covariance matrices R_k and Q_k defines how rapid the filter should respond to the measurement and how much the filter relies on the model. The variances in R_k describing the variance of the input measurements is found as described

above in the sensor section. The variances in Q_k are then used as tuning parameter for the filter.

Matrix R_k and Q_k can be assumed constant assuming the precision of the model and measurements is the same all time. In this case however the variances of R_k are changed while the filter is running. One of the reasons for this is that one of the sensors has a sampling time slower than the update frequency of the Kalman filter. In this case the variance for this sensor is increased in the R_k matrix when no new data has arrived, to indicate that the value in the measurement vector is invalid.

Before starting the design of the filter, the sampling frequency is settled. The update frequency is chosen to be 20 Hz since the controller needs a new estimate of the states of the system with this frequency (see section 6.1).

In the following sections the system matrices are restated and the output vector and matrix are defined. Furthermore the filter design is made in terms of choosing values for $\hat{x}_0^+$, P_0^+, R_k and Q_k.

7.2.1 System matrices

In section 5.2 a nonlinear discrete state space model was found in the following form:

$$q[k+1] = A(\theta)\, q[k] + B\, u[k]$$
$$y[k+1] = C\, q[k+1]$$

where:

$$q[k] = \begin{bmatrix} x[k] & y[k] & \theta[k] & \dot{\theta}[k] & q_1[k] & v[k] & q_2[k] \end{bmatrix}^T$$
$$u[k] = \begin{bmatrix} v_{ref}[k] & \omega_{ref}[k] \end{bmatrix}^T$$

$$A(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cos(\theta)\,Ts & 0 \\ 0 & 1 & 0 & 0 & 0 & \sin(\theta)\,Ts & 0 \\ 0 & 0 & 1 & Ts & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & Ts & 0 & 0 \\ 0 & 0 & 0 & -a2\,Ts & 1 - a1\,Ts & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & Ts \\ 0 & 0 & 0 & 0 & 0 & -c2\,Ts & 1 - c1\,Ts \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & b0\,Ts \\ 0 & (b1 - a1\,b0)\,Ts \\ d0\,Ts & 0 \\ (d1 - c1\,d0)\,Ts & 0 \end{bmatrix}$$

The output vector y is generated to match the sensor inputs (see table 7.1) by forming the output matrix C:

$$y[k] = \begin{bmatrix} x_G[k] & y_G[k] & \theta_G[k] & \theta_M[k] & \omega_{Gy} & \omega_R & v_G & v_R \end{bmatrix}^T$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Where:

$\{*\}_G$ denotes GPS measurements

$\{*\}_M$ denotes Magnetometer measurements

$\{*\}_{Gy}$ denotes Gyro measurements

$\{*\}_R$ denotes measurements from the RobuROC4

For the Extended Kalman filter to work the system matrices A and C must be linearised in the operating point (previous states). Since C is already linear:

$$C_k = \left.\frac{\partial h_k}{\partial q}\right|_{q=\hat{q}_k^-} = C$$

The A matrix is on the other hand nonlinear and is therefore linearised by calculating the Jacobian and inserting the operating point:

$$A_{k-1} = \left.\frac{\partial f_{k-1}}{\partial q}\right|_{q=\hat{q}_{k-1}^+}$$

$$= \begin{bmatrix} 1 & 0 & -\sin(\theta[k-1])\,v[k-1] & 0 & 0 & \cos(\theta[k-1])\,Ts & 0 \\ 0 & 1 & \cos(\theta[k-1])\,v[k-1] & 0 & 0 & \sin(\theta[k-1])\,Ts & 0 \\ 0 & 0 & 1 & Ts & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & Ts & 0 & 0 \\ 0 & 0 & 0 & -a2\,Ts & 1-a1\,Ts & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & Ts \\ 0 & 0 & 0 & 0 & 0 & -c2\,Ts & 1-c1\,Ts \end{bmatrix}$$

7.2.2 Initial state and covariance

The initial states $\hat{q}_0^+$ and covariance P_0^+ can generally be chosen in two different ways. Either pick measurements directly from the sensors and then choose a relatively low covariance, or choose the initial measurement arbitrary, and choose the covariance large.

For this filter it is chosen to set the initial state to zero and have a large covariance. This choice is made since measurements from all sensors are not necessary available at the initial state. The diagonal entries of the covariance matrix are set to 10^9 since this is considered large enough to make sensor inputs change the state fast. The initial states and covariance matrix does therefore have the following form:

$$\hat{q}_0^+ = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$
$$P_0^+ = I_{7\times 7} \cdot 10^9$$

7.2.3 Model and Measurement covariances

The model and measurement covariances are to be defined in two different ways. First both the covariance matrices will be defined for the ideal situation i.e. when it is assumed that the model predicts the states as good as it can and measurements from all sensors are available as correctly as they could be. Afterwards some modifications are made to the covariances to make the filter depend less on the measurement or parts of the model which are considered to be inaccurate. The covariance matrix of the measurements is defined from the variance of the sensor measurements found in section 7.1:

$$R_k = diag(\sigma^2_{x_G}, \sigma^2_{y_G}, \sigma^2_{\theta_G}, \sigma^2_{\theta_M}, \sigma^2_{\omega_{Gy}}, \sigma^2_{\omega_R}, \sigma^2_{v_G}, \sigma^2_{v_R})$$
$$= \begin{bmatrix} 0.003253 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.003253 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.002721 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.056400 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.000025 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.000053 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.036187 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.000041 \end{bmatrix}$$

The variance for the GPS position measurement is used directly from table 7.2. Since the desired operational linear velocity for the vehicle is close to 2.2 m/s the variance calculated for both heading and linear velocity when having this linear velocity is used.

The variance for the magnetometer heading measurement is chosen to be $0.0564 \cdot$ This is the mean of the calculated variances for the four tests specified in table 7.3 multiplied by 100. Tests showed that it is very hard to obtain consistent measurements between the GPS heading and the magnetometer heading mainly due to disturbances in the magnetic field next to cars, buildings and light poles. The variance is therefore multiplied by 100. By doing so the magnetometer heading will only influence the estimated heading slightly when the GPS reports headings, but when the magnetometer is the only heading measurement (the vehicle is standing still) the filter will converge towards this measurement and give an overall heading.

The variance of the gyro angular velocity measurement is chosen to be the mean of the variances found in the six tests. Even though the variance is higher for high angular

velocities, it is expected that the robot generally will have low angular velocities while driving on a trajectory. Choosing the variance too high will imply the estimate to depend too little on the gyro measurements when driving with low angular velocities which will be the case most of the time.

As seen in table 7.5 a larger linear velocity and angular velocity of the vehicle implies a larger variance for both the angular and linear velocity measurements from the vehicle odometry. The variance calculated when driving with a linear velocity of 2.2 m/s is therefore used. From table 7.5 it is seen that the variance when having an angular velocity of 1.57 rad/s is much higher than the one chosen for the filter. The variances are not chosen higher since the angular velocity will be close to 0 rad/s or at least far from 1.57 rad/s in most cases.

The covariance matrix for the model is constructed in the same way. The separate variances are selected such that the filter smoothens the data in a desired way for some test samples of data consisting of both system inputs and measurements:

$$Q_k = diag(\sigma_x^2, \sigma_y^2, \sigma_\theta^2, \sigma_{\dot{\theta}}^2, \sigma_{q_1}^2, \sigma_v^2, \sigma_{q_2}^2)$$

$$= \begin{bmatrix} 0.000010 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.000010 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.000010 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.000010 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.000010 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.000004 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.000001 \end{bmatrix}$$

Here the covariance matrices are defined for the ideal situation when the sensors measurements are valid and the vehicle model output is close to the behaviour of the vehicle. The remaining part of this section concerns deviations from the ideal situation, and what changes are made to the covariance matrices in the separate situations to avoid harmful impact on the estimate when the measurements or model is known to be incorrect.

Missing sensor data: Not all sensors give data with the same sample rate. In the case that no new data has arrived for a specific sensor the variance for this sensor is set to 10^9 to make the filter disregard the measurement.

Faulty heading measurement from the GPS: Since the GPS calculates the heading based on successive readings the precision of the measurement therefore depends on the linear velocity of the vehicle as shown in table 7.2. If the velocity is low the precision of the heading measurement is low and vice versa. The measurement variance of the GPS heading measurements is therefore increased when the linear velocity decreases. When the estimated linear velocity is below 2 m/s the measurement variance is set to $32.059 \cdot 10^{-3}$ rad^2. This is the mean of the measurement variances for the three low linear velocities shown in table 7.2. If the estimated linear velocity is below 0.25 m/s the

heading measurement from the GPS is completely disregarded by setting the measurement variance to 10^9 rad^2.

Due to the position of the GPS antenna (see appendix A) the heading measured by the GPS will be incorrect when turning since the GPS antenna will move in the transverse direction of the vehicle. The heading measurement from the GPS is therefore also disregarded when the estimated rotational velocity is above 0.3 rad/s.

Faulty angular velocity measurement from the gyro: The used gyro is capable of measuring angular velocities up to 100 °/s corresponding to 1.75 rad/s. Since this is below the maximum angular velocity of the vehicle the measurement from the gyro is disregarded, by setting the measurement variance to 10^9rad^2/s^2, if the measurement of the gyro is saturated i.e. the measurement is 100 °/s.

7.3 Filter Test

To show the performance of the designed filter, the filter is implemented on the robot as described in section 10.2, and a test run is performed. In this test run the vehicle is controlled by the implemented controller following a predefined trajectory.

For the model to be useful in terms of predicting the position of the vehicle the linear and angular velocities must be determined well. In figure 7.2 the estimated angular velocity is shown along with the measurements and the input. As seen in the figure the overall shape of measurements, inputs and estimate is the same. In the figure it seems that the input is offset in time. This is a consequence of the dynamics of the system and a little delay in the connection from the computer to the vehicle controller.

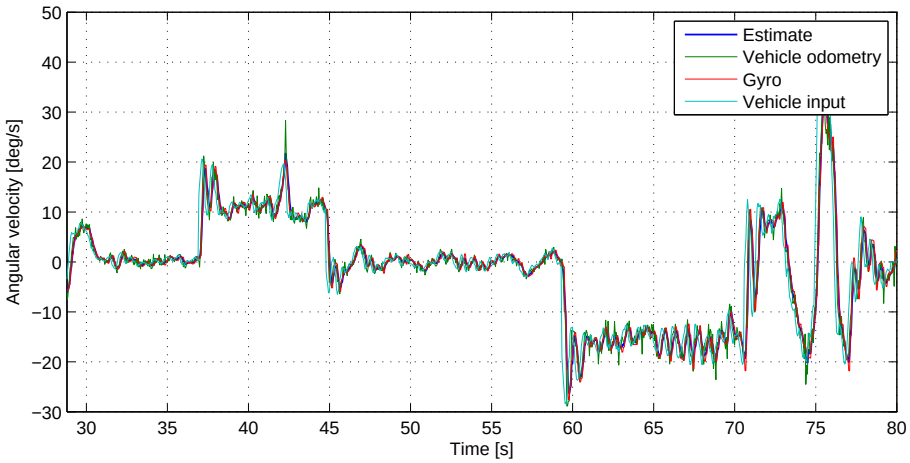


Figure 7.2. Estimated and measured angular velocity.

In figure 7.3 a zoom of 6 s is shown. Here it is seen that the estimate is smoother than the measurements. In the figure it seems that the gyro measures the same value for some

time. This is caused by the lower sampling rate of the gyro. The gyro samples at 10 Hz where the filter runs a 20 Hz. Around 29 s it is seen that the estimate is very close to the new gyro samples and the second sample with the same value from the gyro is not necessarily close to the estimate. This observation complies with the fact that the gyro measurements have a low variance and that the filter disregards the measurement when no new data has arrived.

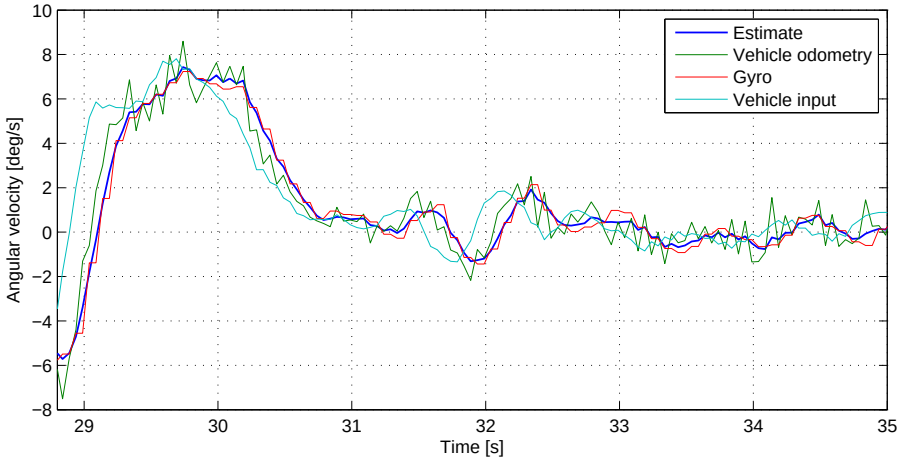


Figure 7.3. Estimated and measured angular velocity (zoomed).

In figure 7.4 the estimate of the linear velocity is shown along with the measurements and inputs. As for the angular velocity the overall shape of the measurements and estimate corresponds. As seen in the figure the measurement from the GPS is more noisy than the measurement from the vehicle. This observation complies with the variances found for the GPS sensor and the vehicle odometry in sections 7.1.1 and 7.1.4 respectively. Even though the GPS measurements are quite noisy the estimated linear velocity is very smooth.

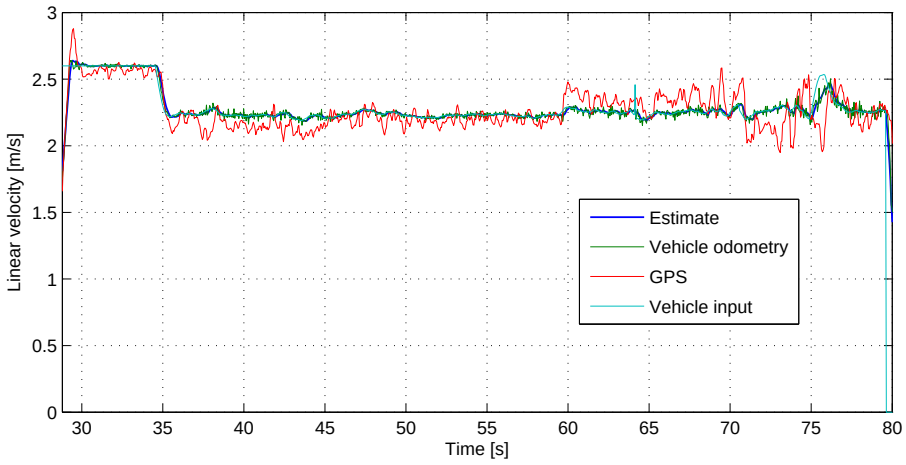


Figure 7.4. Estimated and measured linear velocity.

Another observation made from the figure is that the linear velocity measurement from the GPS is lower than the estimate in the time interval between 35 s and 45 s and to be higher than the estimate in the interval from 60 s to 70 s. These deviations from the estimate is caused by the position of the GPS antenna described in appendix A. Since the antenna is placed in the left side of the vehicle doing a left turn causes the GPS to move with a lower linear velocity that the centre of the vehicle and likewise doing a right turn implies that the GPS moves with a larger linear velocity than the centre of the vehicle.

The estimated heading and the heading inputs from the magnetometer and the GPS are shown in figure 7.5. As seen in the figure the overall shape of measurements and estimates complies. However, the magnetometer measurements seem to be slightly different from the GPS measurements and from the estimate. The reason for the estimate to be closer to the GPS measurements than to the magnetometer measurements is that the variance for the magnetometer is chosen lower than the variance for the GPS due to incorrect readings from the magnetometer.

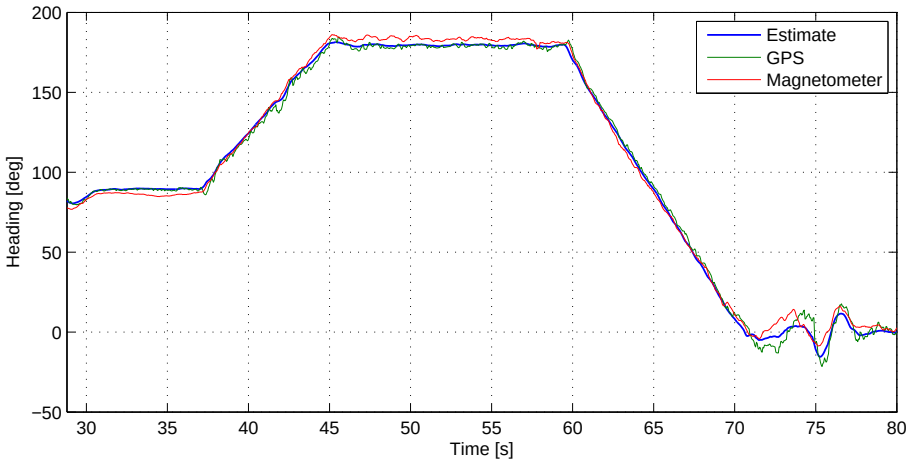


Figure 7.5. Estimated and measured heading.

These incorrect readings are also seen from figure 7.5. If the magnetometer was perfectly calibrated a misalignment of the magnetometer would cause an offset in the measurements, but as seen in the figure the magnetometer reading is sometimes larger than the GPS measurement (45 s to 55 s) and sometimes less than the GPS measurement (30 s to 35) and is therefore considered as a miscalibration rather than a misalignment. In the two periods where the heading is compared the vehicle is running straight which is why the GPS heading is considered to be correct.

The last part of the filter test to be shown is the estimate of the position of the vehicle. In figure 7.6 the estimate of the vehicle position is shown along with the measurement from the GPS. On the following figures an offset is subtracted such that the starting point is at coordinate (0,0). As seen in the figure the overall estimate follows the GPS data as expected.

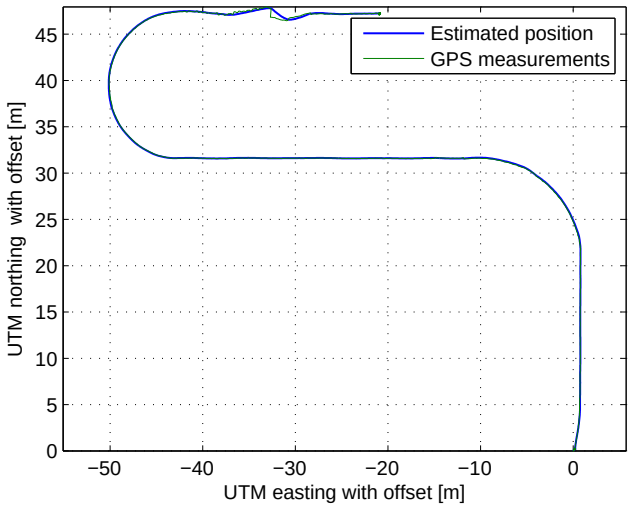


Figure 7.6. Estimated and measured position for the complete test.

In figure 7.7 the position estimate is shown in the left turn and just after it. As seen in the figure the estimate is smoother than the GPS measurements but the overall shape is the same. Even the jump in the GPS measurements around $-12\,\mathrm{m}$ easting does not seem to influence the estimate.

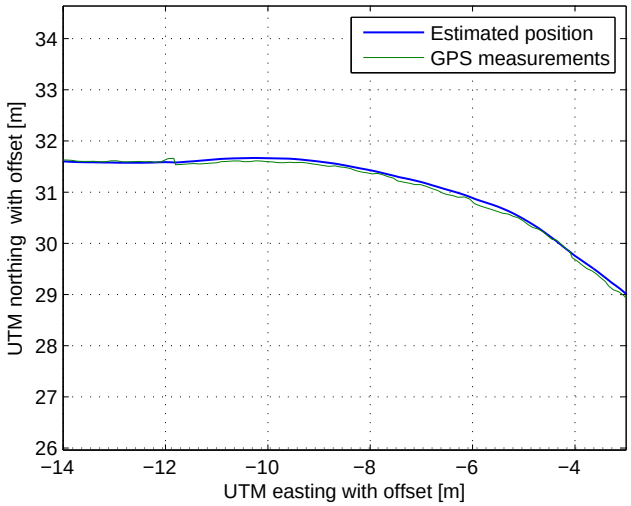


Figure 7.7. Estimated and measured position for the left turn.

In figure 7.8 the position estimate is shown around the right turn. As seen in the figure the measurements get more noisy whereas the estimate stays smooth. From the figure it seems that the GPS measurements drift to the north just before the large jump since the measurements are larger that the estimate until the sudden jump. From this it is seen that the filter gives a good estimate of the position of the vehicle when the GPS measurements are correct with small variance but drifting measurements and sudden large jumps in the measurements will cause the estimate to change rapidly since the

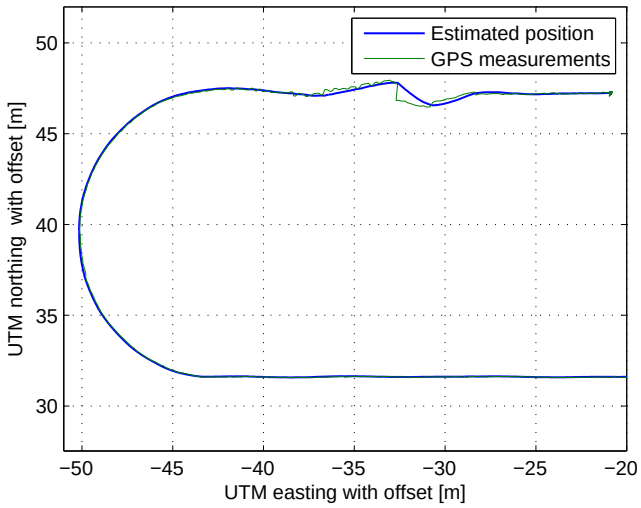


Figure 7.8. Estimated and measured position for the right turn.

GPS position measurements are considered very precise in the design of the filter.

Summary

During the test of the filter it is shown that the states of the vehicle are estimated well by using the extended Kalman filter, the sensor measurements and the model. It does however seem that some of the measurements from the sensors could be improved.

By calculating the heading from the magnetometer in a more advanced way (including the accelerometers for detecting pitch and roll) the filter could rely more on the measurements from the magnetometer and thereby the estimate could be more correct when the heading from the GPS is incorrect i.e. when the vehicle is standing still or turning rapidly.

The position measurement from the GPS could possibly also be improved by examining the phenomenon when the position suddenly jumps. In the datasheet for the GPS the precision is stated to be down to 2 cm which is not the case since the jumps in the position is much greater than 2 cm. To examine this problem a greater analysis of the setup of the GPS must be carried out.

8 Road detection

This chapter contains a description of how the road edge is determined from a visual input. First the overall procedure is introduced and a short description of the parts needed in the road detection is made. Afterwards the parts are described in details.

The overall idea to the procedure used to detect roads on images is inspired from [13].

The overall road detection process is shown in figure 8.1. First an image is acquired using a camera mounted on the vehicle (see appendix A).

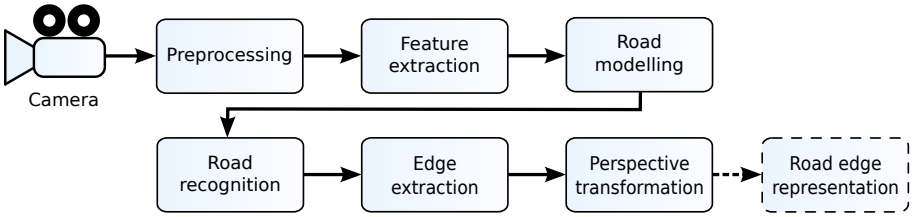


Figure 8.1. Overall road detection process.

The acquired image is preprocessed making the image comply with the requirements given by the rest of the road detection process. Next the preprocessed image is divided into smaller parts called clusters, and for each cluster some features are extracted. Both the clustering and feature extraction takes place in the feature extraction part.

Based on the knowledge of how the vehicle is located on the road, a model of the road is generated by combining the extracted features for the clusters that is known to be the road. Road-like clusters are hereafter recognised by comparing all clusters in the image with the generated model of the road. The right edge of the road is then extracted and a perspective transformation is conducted to obtain a representation of the road edge in the robot frame.

In the following sections the functionality of each block in the block diagram shown in figure 8.1 will be presented from the first block containing the preprocessing to the last block containing the perspective transformation where the output is a representation of the right road edge. At the end of the chapter some assessments will be made on the proposed algorithm.

Through the following sections the different operations performed will be illustrated using the road image shown in figure 8.2, if nothing else is mentioned. This image is chosen since it has some shadows which the road detection algorithm must be able to deal with.

Figure 8.2. Image used for illustration in the different steps of the road detection algorithm.

8.1 Preprocessing

The input image received from the camera is turned upside down due to the mounting of the camera described in appendix A. The image is therefore rotated 180° as the first step in the preprocessing. As described in section 8.3 the image is divided into clusters of size X times X pixels later in the road detection algorithm for which reason the image is cropped to match an integral number of clusters in both directions of the image. This is illustrated in figure 8.3.

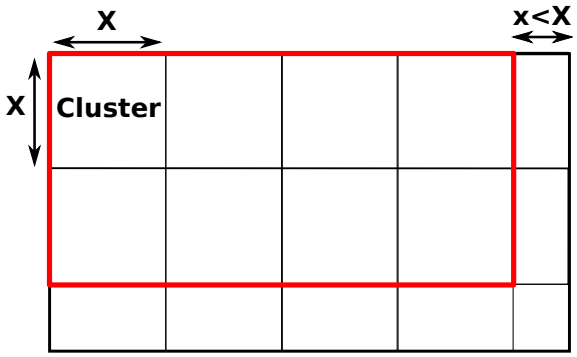


Figure 8.3. Cropping process. In red the cropped image.

The input image from the camera has a resolution of 640x480 pixels and as described in section 8.2 the clusters are chosen to have a size of 15x15 pixels for which reason the image is cropped to a size of 630x480 pixels corresponding to the red rectangle shown in figure 8.3. Due to the mounting angle of the camera it captures lots of information far from the vehicle. Since the information in the far distance from the camera is not that precise it is chosen to remove the upper 210 pixels of the image to improve performance in terms of required computational power. The images shown in the rest of this chapter do however include the top of the images to illustrate better how the algorithm works.

Once the cropping is performed the input image is transformed into the HSV colour space in which colours are split into the three channels Hue, Saturation and Value. For the test image the three different layers are shown in figure 8.4.

Figure 8.4. Hue, Saturation, and Value channels on a shadowed road (from left to right).

8.2 Feature Extraction

Once the preprocessing of the image is performed the image is divided into clusters of 15x15 pixels. Then the image is transformed into a feature space. It has been chosen to use the mean of the Saturation and the mean of the Hue of the pixels within a cluster to constitute the features of this cluster. Theses two features are stored in a vector $x = \begin{bmatrix} S & H \end{bmatrix}^T$ for each cluster. The initial choice of features is based on the images of the three layers of the HSV image shown in figure 8.4. It is seen from the figure that the Hue and Saturation channels show good contrast between the road and the non-road pixels whereas the Value channel does not show much contrast and is very sensitive to shadows. Due to this the Saturation and the Hue are considered useful for separating road and non-road parts of the image.

In the figure 8.5 the features of each cluster extracted from the image shown in figure 8.2 are plotted using their actual RGB colour. The clusters belonging to the road are circled in red, the one belonging to the sky in blue and the one from the grass in green. Even though the sky clusters are quite close to the clusters containing the road this will not have any influence on the result since the sky is removed in the cropping part of the preprocessing step. Once the sky clusters are removed it can be seen that the road and grass clusters are clearly separated. The outlier clusters represent other colours present in the image such as buildings, threes or even dark spots on the road. For example the orange clusters located on the bottom left of the figure are part of the brick wall present in the original image.

8.3 Road Modelling

To be able to detect which clusters in the clustered image contains the road, a model of the road is used. To find such a model a prototype of what is assumed to be the road beforehand is used. Since the vehicle is assumed to be driving on the right side of the road, the bottom left of the image is considered as being the road, and a minimum prototype is defined in this region as shown on the figure 8.6.

The shape of this prototype has been chosen such that only the roads clusters are included in the prototype even if the vehicle is not perfectly aligned with the road at the time where the image is captured. The model of the road is hereafter generated as the

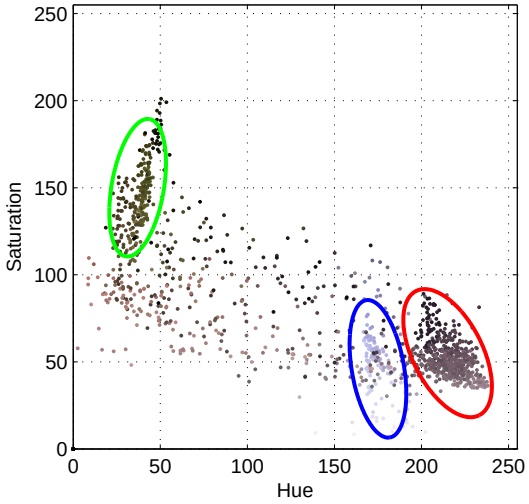


Figure 8.5. Hue and saturation value of each cluster in real colour with grass circled in green, sky in blue and road in red. Computed from the input image 8.2.

Figure 8.6. Minimum prototype used to find the road model.

mean of the separate features for the clusters that belong to the prototype $\mathcal{P}$ of the road:

$$\mu = \frac{1}{N} \sum_{x \in \mathcal{P}} x$$

Where:

μ is the model of the road

N is the number of clusters covered by the prototype of the road

To generate a better model of the road, the road found on the previous image is used to update the prototype. By doing so the prototype will include a larger part of the road, and the model will therefore describe the road better.

In figure 8.7 the prototype generated from the road detected on the previous image is presented. The edge of the road computed in section 8.4 is plotted in red in the image and is used to generate the road prototype plotted in blue. To generate the prototype three

points are extracted from the road edge: the two points where the found road edges crosses the edge of the image and the middle of the two top corners. As the image is cropped in the preprocessing part the edge of the road appears truncated in this figure. Thanks to the final triangle like shape of the prototype shown in figure 8.7 non-road clusters will not be included in the image if the vehicle turns between two successive images.

Figure 8.7. Prototype generation, in blue, from the detected in the previous image.

To make the final prototype, the minimum prototype shown in figure 8.6 and the new prototype based on the road detected shown in figure 8.7 are merged. If the road is not detected properly on the previous image, (see section 8.4) only the minimum prototype will be used. If the road is not detected in the previous image it is most likely because the prototype used in the previous image contains something else than the road. By resetting the prototype to the minimum, the road model should not contain other clusters than the road.

8.4 Road Recognition

To determine which clusters belong to the road, and thereby detect the road, a similarity measure between the features of each cluster in the image and the model of the road must be defined. In figure 8.8 the features of each cluster of the image 8.2 are plotted with their actual RGB colour except from the red ones which belong to the road prototype.

From the red points in figure 8.8 it can be seen that there seems to be a small correlation between the two features. For this reason it is chosen to use the Mahalanobis distance to calculate the similarity between the model and the clusters in the image. The Mahalanobis distance between the clusters in the input image and the model of the road

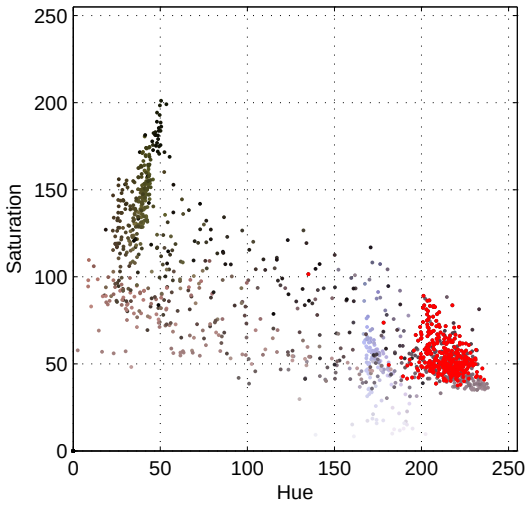


Figure 8.8. Hue and saturation value of each cluster in real colour. The red ones belongs to the road prototype.

is calculated as:

$$D(x) = \sqrt{(x-\mu)^T S (x-\mu)} \tag{8.1}$$

Where:

x is the features for cluster

μ is the model of the road

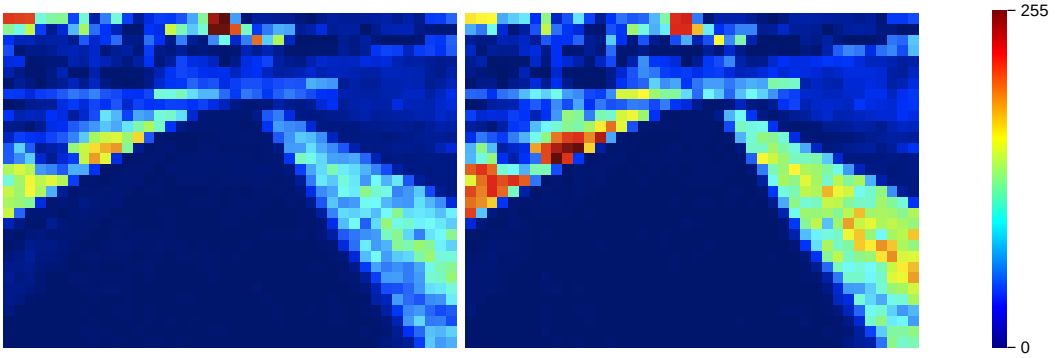
S is the covariance matrix calculated from the samples used to generate the road model

To compare the performance of the Mahalanobis distance to the Euclidean distance[1] the distance from the model to each cluster in image 8.2 is calculated using both similarity measures. In figure 8.9 the calculated distances are represented in the colour map presented in figure 8.9(c) where red areas indicate large distances and blue areas indicate small distances.

From the larger contrast seen in figure 8.9(b) compared to figure 8.9(a), it is seen that using the Mahalanobis distance, the similarity between the model of the road and the non-road clusters are generally less than when the Euclidean distance is used. For three different images the Mahalanobis distance is calculated, and both the original image and the distance image are shown in figure 8.10. In the distance map the intensity directly give the distance i.e. dark areas are short distances and bright areas are large distances.

From this figure it is seen that the contrast in the distance images is smaller when shadows are present. On figures 8.10(a) and 8.10(c) where no shadows are present a large contrast in the distance image is observed whereas the distance image for the image with shadows in figure 8.10(b) shows less contrast and thereby less distance from the model to the non-road elements on the image.

[1] Same as Mahalanobis but with identity covariance matrix.



(a) Euclidean distance. (b) Mahalanobis distance. (c)

Figure 8.9. Distances between the model and the clusters of the image in figure 8.2 using either the Euclidean or Mahalanobis distance.

(a) On a straight road. (b) On a road with shadows. (c) On a curved road.

Figure 8.10. Mahalanobis distance for different roads types and conditions.

Once the Mahalanobis distance is computed a fixed threshold is applied to obtain a binary image. This threshold value should allow an easy recognition of the road without including too many other clusters. In figure 8.11 the histogram of the figure 8.9(b) is shown and the threshold applied is plotted in red. It can be seen that the threshold value could be decreased. However, choosing it like shown in figure 8.11 allows the algorithm to include more clusters having a small distance from the road model like the shadow and the green line shown in figures 8.10(a) and 8.10(c). Once this threshold is applied several separated regions can be identified as shown on figure 8.12.

To determine which of theses regions are actually the road an erode morphological operation using a square of size 2x2 pixels is first applied to break the connection between isolated clusters that could be connected to bigger regions. Then all the areas are compared and the largest one is considered as being the road and will be used during the rest of the road detection.

A maximum possible area of the road region is defined as being 70 % of the image area, in order to detect cases where non road clusters are detected as the road. In case

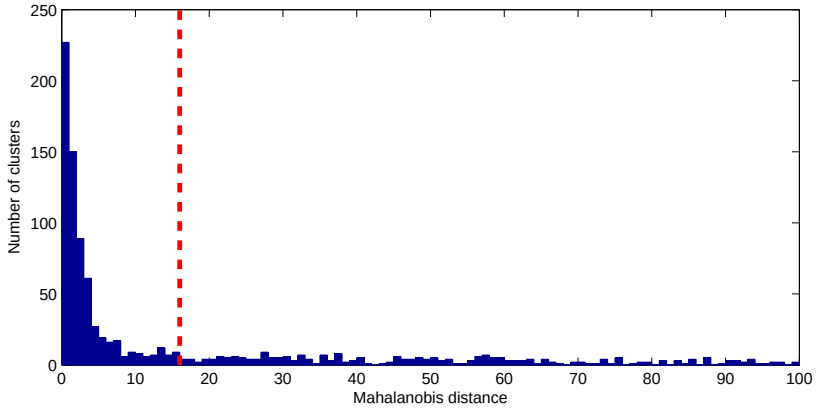


Figure 8.11. Histogram of the distance map calculated using the Mahalanobis distance. The red line corresponds to the threshold value used.

(a) On a straight road. (b) On a road with shadows. (c) On a curved road.

Figure 8.12. Right edge detection for the road's blob in different road configurations.

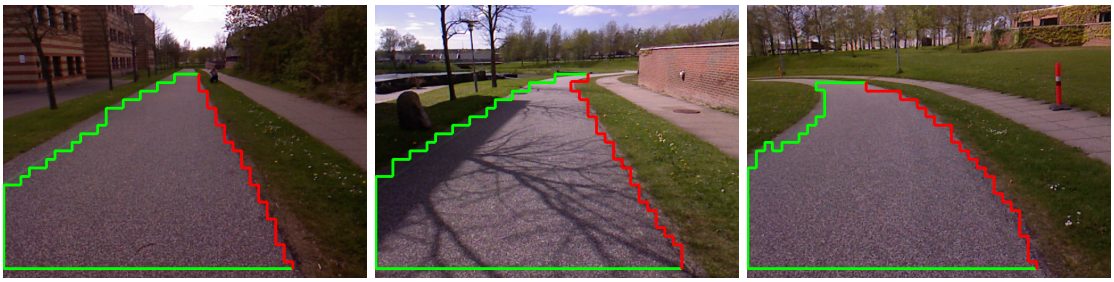
the detected road area become greater than the maximum allowed the prototype that will be used to process the next image is re-set to the minimum one. Then the computation of this image is stopped to avoid sending corrupted data and the algorithm starts again from the next image received. As the calculation of the next road model will not take in account this error the algorithm is actually able to recover from its error and will not send any wrong data.

8.5 Edge Detection

From the detected road region the edges of the road can be extracted as shown in green in figures 8.13. As our focus is to detect the right border of the road the next step is to extract the actual right border from this road's contour. To do so only the right of the road edge is extracted as shown in red in figures 8.13.

8.6 Perspective Transformation

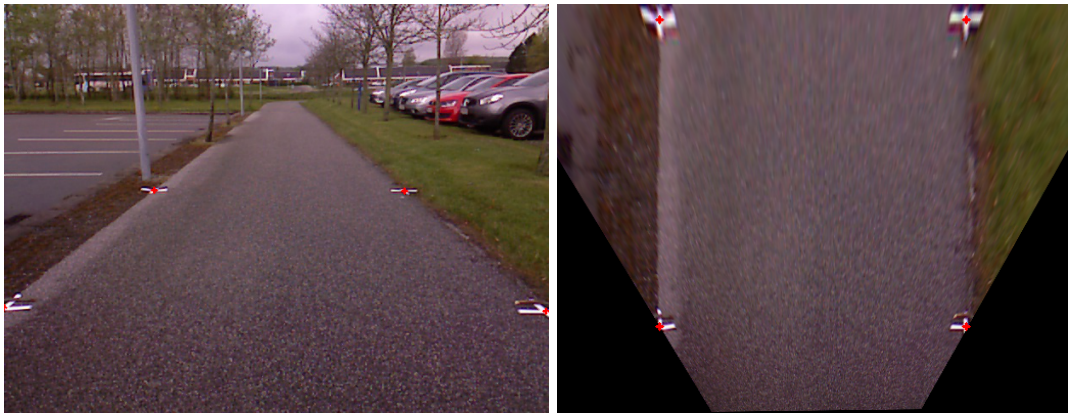
When the right edge data is calculated, the coordinates are expressed with respect to their position on the captured image. In order to evaluate the distance of these points, they must be expressed in a Cartesian coordinate system. In this way, the perspective



(a) On a straight road. (b) On a road with shadows. (c) On a curved road.

Figure 8.13. Right edge detection for the road's region in different road configurations.

is not visible and the coordinates are expressed as if they were seen from the top. A transformation is generated by locating four points on the original image and assigning a position to them in the transformed image. To do the transformation in a proper way, it is chosen to form a square with a side length of 3 m on the ground by placing markers (see figure 8.14(a)). They are then located on the input image from the camera and assigned to their new position on the transformed image such that they form a square with a side of 300 pixels as shown on figure 8.14(b).

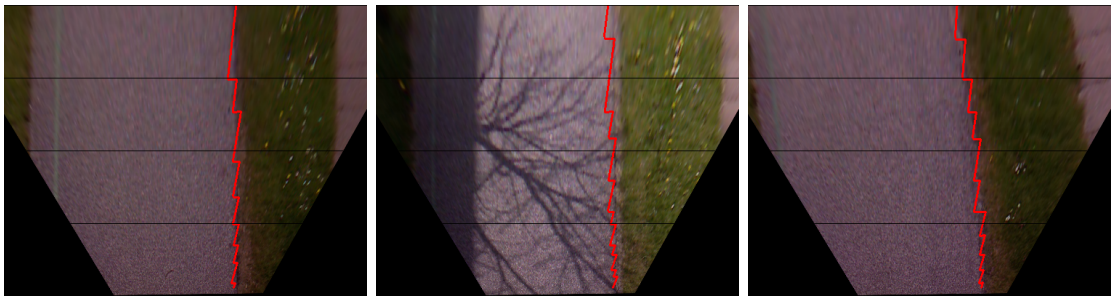


(a) Original Image. (b) Transformed Image.

Figure 8.14. Reference pixels (in red) are placed in both images to generate the transformation matrix.

The transformation matrix is then calculated through a perspective transform algorithm and results in a transformation matrix which can be applied to any point in order to retrieve its coordinates in the new orthogonal coordinate system. Thus, each point of the right edge is transformed resulting in a new representation of it with right proportions as shown on figures 8.15.

Since the accuracy of the perspective transformation decrease quickly when the distance increase, the detection of the road does not go further than 6 m from the camera to avoid sending inaccurate data. As presented in appendix A the camera setting does not allow the robot to see the closer part of the road up to 2 m. Therefore for each successful road detection the transformed right edge is computed from 2 m to 6 m ahead of the centre of the vehicle.



(a) On a straight road. (b) On a road with shadows. (c) On a curved road.

Figure 8.15. Right edge transformation in different road configurations. The horizontal black lines are placed in 1 m intervals.

As the computation of the upper part of the image corresponding to further than 6 m in the top view is pointless, this part is cropped as presented in section 8.1 to decrease computation time.

Assessments On the Method

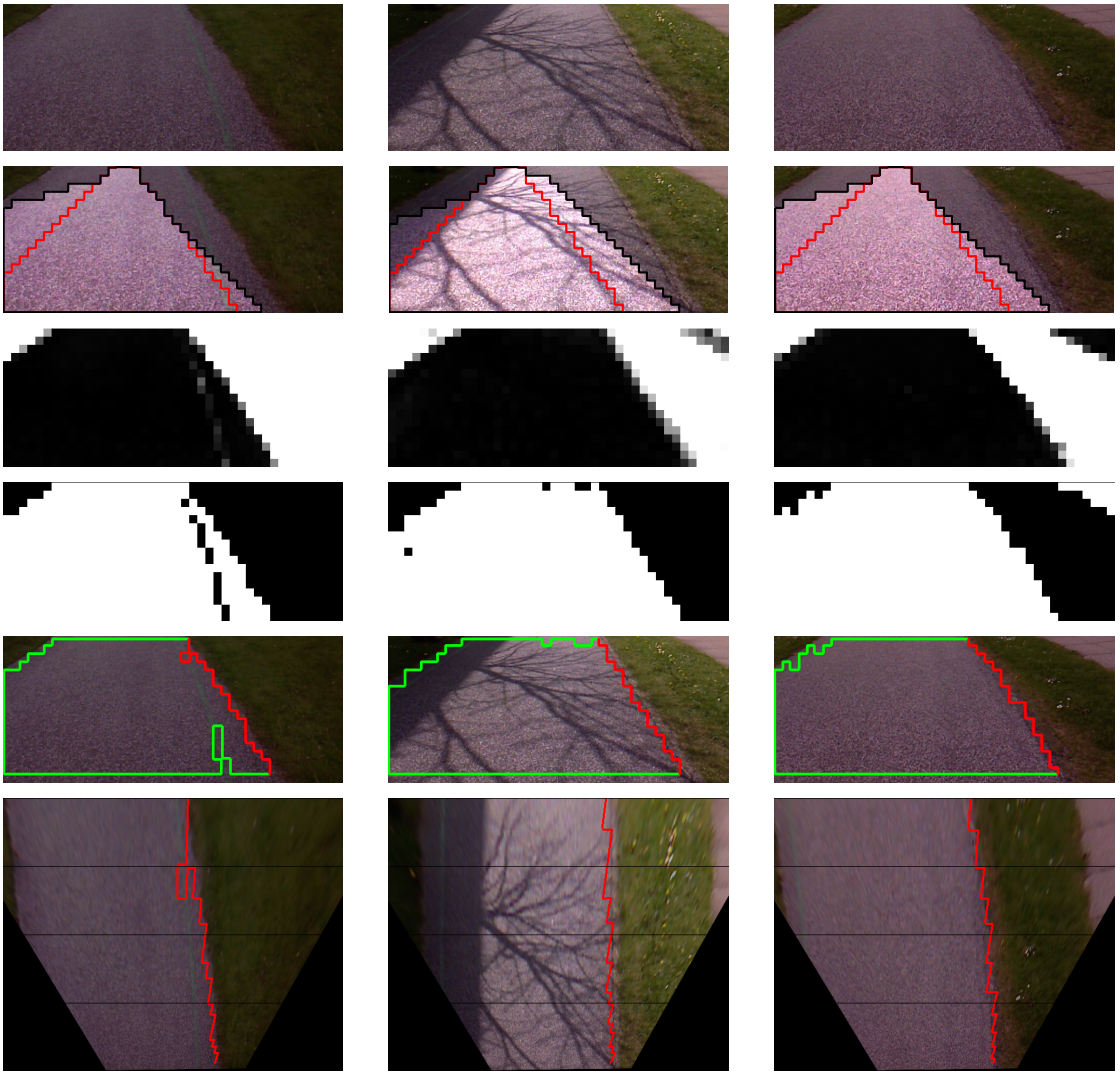
This section presents the intermediate images of the major steps of the algorithm previously described in three different road configurations. These images are shown in figure 8.16. From top are presented: the original image, the prototype used (and the minimum marked by red), the grey scale Mahalanobis distance image, the binary image, the right edge found, and finally the image without perspective effect.

From theses three sets of images presented in figure 8.16 it can be seen that the right edge of the road is well detected even in hard lightening conditions. In the set of pictures seen on figure 8.16(a) it is seen that there is a green line painted on the right edge of the road, and this line is not detected as the road after the threshold. Nevertheless the extraction of the right edge of the road does not take this line into account and the edge detection is not affected except for one cluster on top of the image.

In the figure 8.16(b) the lighting conditions are different as there are shadows on the road both from a building and a three. It can be seen that without the knowledge of the previous image to create the road prototype the shadow from the building would not be included in the road model so the distance map would present less contrast and therefore the road detection would not be that accurate.

In figure 8.16(c) it can be seen that even though the road is turning left the grass is not included in the road prototype and the right edge is correctly extracted.

From theses three examples it can be assessed that the algorithm is robust enough to detect the right edge of the road for both various road condition and lightening condition. The weakest edge detection occur for the set 8.16(a) where the green line is considered as not being a part of the road. However this case should not appear in real road striping condition.



(a) On a weakly defined edge. (b) On a road with shadows. (c) On a curved road.

Figure 8.16. Major steps of the road recognition plotted for three different road configurations.

9 Trajectory Generation

This chapter contains two different methods of generating trajectories. The first half of this chapter describes the generation of the trajectory that is sent to the controller from the right border coordinates calculated in the road detection part. The second half of the chapter contains a description of how the virtual pre-markings are created from a larger set of road edge data which is available beforehand.

9.1 Local Trajectory Generation

This section contains the description of how the local trajectories are generated directly from the road edge extracted in the road detection described in chapter 8. In the beginning it is explained how the road edge is approximated for each image and then converted into the global coordinate system. It is then described how the different trajectories calculated for each image are combined to form a new trajectory which is the one that will be sent to the controller.

For each image in the section, the y coordinates are on the vertical axis going from the top to the bottom and the x coordinates are on the horizontal axis going from the left to the right.

9.1.1 Edge approximation

From the extracted right edge in the road detection the best possible approximation for the real edge of the road must be made. The edge approximation will then be used to generate the trajectory that the robot must follow. The approximation must therefore be as smooth as possible and must stick to the edge of the road. A third order polynomial fit has shown to give the best compromise between smoothness and accuracy.

Making a polynomial approximation directly on the data points that have been transformed does not give satisfying results. This can be explained by the fact that the points close to the robot are much more numerous than the one that are far away since the equivalent distance in the real world between two pixels exponentially increases with the perspective. Therefore, the very close part of the road has too much influence on the approximation. To avoid this problem, a linear interpolation is made on the data points for each pixel in the y axis coordinate going from 0 to 400. Before this interpolation, the data is thinned such that there are not multiple points for the same y coordinate. This is necessary in order to allow the interpolation algorithm to work. In this way one point represents one centimetre of the edge, which results in an even weight independent of the distance from the camera.

The linear interpolation is illustrated in figure 9.1. On the figure, the blue points show the edge coordinates as they are received and the red points show the interpolated values all along the edge. For display convenience, the interpolation on the figure is made every 5 pixels in the y axis instead of every 1 pixel.

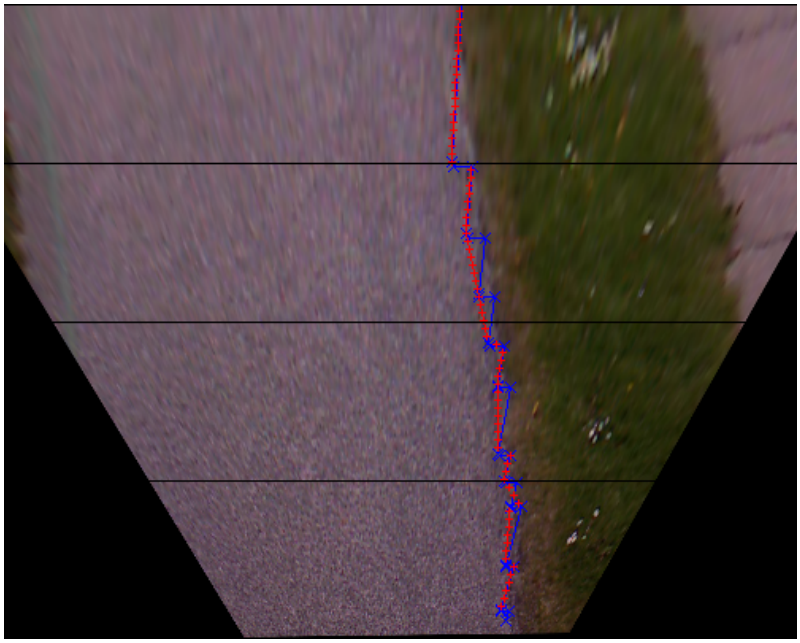
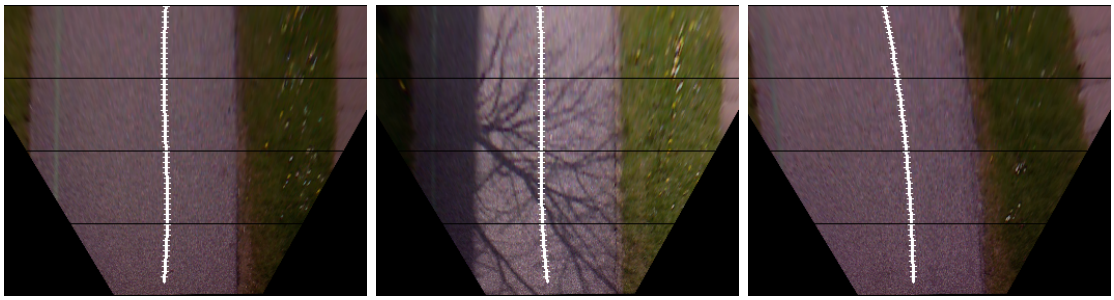


Figure 9.1. Linear Interpolation of the right edge.

A least-square third order polynomial approximation is performed on the interpolated data along the y axis. This polynomial is evaluated for y values with a step $\Delta y = v \times T_{controller}$ where v is the desired velocity of the vehicle and $T_{controller}$ is the sampling period of the controller. This step is chosen because it is the distance between consecutive points of a trajectory which is required by the controller for running at this certain speed. This polynomial approximation is the final representation of the road edge, an offset is added to this approximation along the x axis, in order to represent the trajectory that will be followed and will then be used to generate the final trajectory seen in figure 9.2. The offset value is equal to the sum of the desired distance between the edge and the robot, and half the width of the robot.



(a) On a straight road. (b) On a road with shadows. (c) On a curved road.

Figure 9.2. Trajectory generation evaluated from the right edge polynomial approximation.

9.1.2 Conversion to global coordinates

The trajectory approximation calculated for each image is expressed in a coordinate system which is relative to the robot and must be converted to the UTM coordinate system. This section explains how the transformation is made taking into account the position of the robot relatively to the image and the position of the robot in the global frame.

To calculate the position of the robot relatively to the image some distances are measured beforehand. These measurements allows us to represent the location of the robot on the transformed image calculated before and to locate the trajectory relatively to the robot (see figure 9.3, where the image is captured when the vehicle is positioned parallel to a straight road). The coordinates of the vehicle (x, y) on the transformed image can be expressed as:

$$x = x_{rightedge} - \Delta x$$

$$y = y_{proj} + \Delta y + \frac{R_{lenght}}{2}$$

where:

$x_{rightedge}$ is the x coordinate of the right edge of the road on the image

Δx is the distance between the right edge of the robot and the right edge of the road

Δy is the distance between the front edge of the robot and the bottom of the image

R_{length} is the length of the robot

y_{proj} is the y coordinate of the bottom of the image

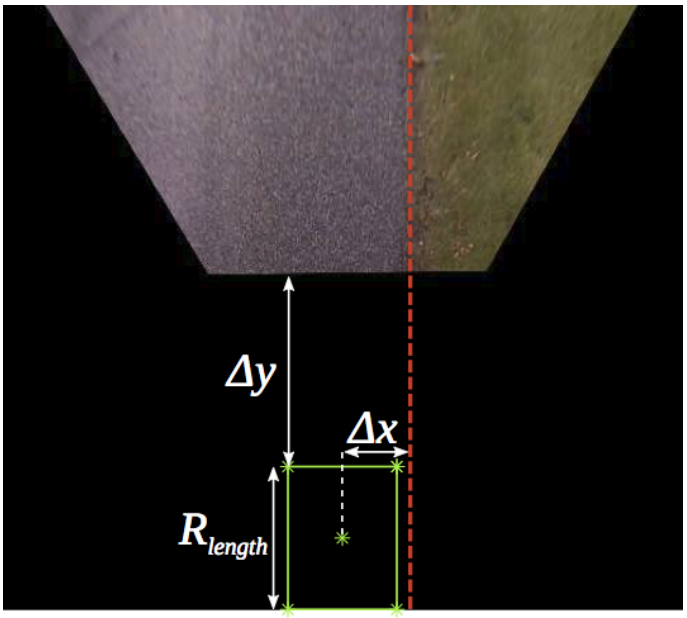


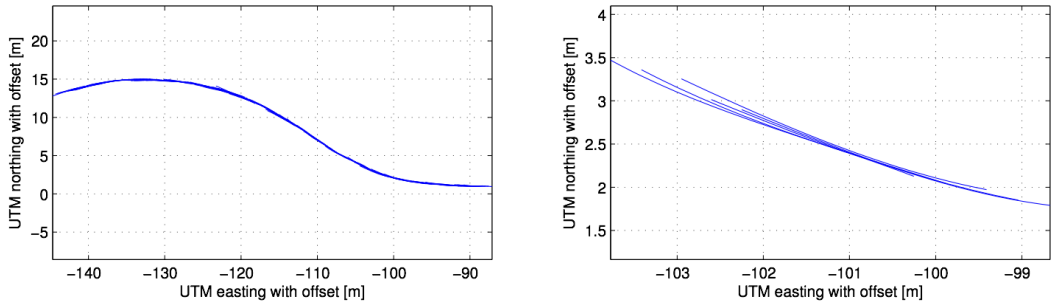
Figure 9.3. Localisation of the robot on the image.

When the coordinates of the robot have been expressed, it is possible to generate the transformation matrix that will allow to transfer these coordinates into the UTM coordinate system. To perform this task, four transformations are applied.

The first transformation performs a translation of the robot such that its centre is moved to the origin of the global frame. The second transformation mirrors the coordinates in the x-axis. By doing so coordinates are converted from the image coordinate having positive direction of the y-axis downwards to the global coordinate system having the positive direction of the y-axis upwards.

A rotation transformation is performed around the origin with an angle $\theta_2 = \theta - \frac{\pi}{2}$, where θ is the angle of the robot given by the estimated states. $\frac{\pi}{2}$ is subtracted from the angle of the vehicle because the robot is facing upwards before the rotation, and an angle of 0 should make the robot face east. Finally a translation transformation is performed to move the robot from the origin to its real position given by the estimate of the position states.

Figure 9.4 shows trajectories generated from several successive images captured on a test run and transformed to the global coordinate system. In the figures the trajectories are generated for the vehicle driving from east to west hence the rightmost point on each trajectory is the closest to the vehicle.



(a) In a large scale.

(b) In a small scale.

Figure 9.4. Generated trajectories from a test run.

In this figure, it can be seen that consecutive trajectories are superposed together meaning that the newly found trajectory on each image is coherent with the previous one. However, these trajectories are not exactly superposed and a little offset is always present between two trajectories. Since the controller has to receive a non-overlapping trajectory without such offsets, a way to combine and smoothen these trajectories must be found.

9.1.3 Combining Trajectories

When a new trajectory is calculated from an image, the robot is already following another one. To start following the new trajectory, the robot must slowly converge to it and then continue on it. The way to do this is to create a trajectory which starts at the position of the vehicle, sticks to the old trajectory until the beginning of the new trajectory is reached, and then sticks to the new trajectory.

To perform this, the distance of each point of the old trajectory to the first point of the new trajectory is calculated. At the beginning of the old trajectory, this distance decreases

as the points are getting closer to the new trajectory. When the distance calculated for one point is longer than the one calculated for the previous point the closest point is found. All the points that are placed ahead are deleted from the old trajectory. Also, since the trajectory must start from the position of the robot, the same process is applied to the points of the old trajectories, but their distance is calculated from the position of the robot which is accessible by the states of the robot. The result of this deleting of old data can be seen in figure 9.5.

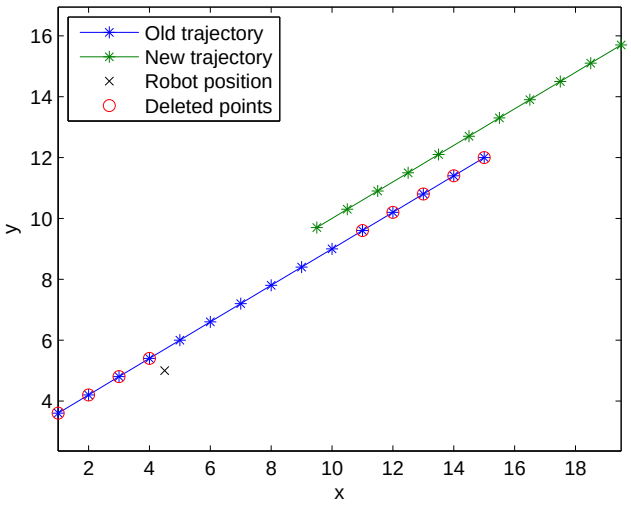


Figure 9.5. Deletion of the points that are not useful anymore from the old trajectory.

When this operation has been performed, a sudden jump is left in the trajectory. This is not desired since it would cause the controller to make large corrections which could cause the robot to get off course. To overcome this issue the n last points of the remaining part of the old trajectory is removed. These points are then recalculated to form a linear trajectory of $n-1$ points with a constant distance between consecutive points, going from the last remaining point in the old trajectory to the first point of the new trajectory. This replacement is illustrated on figure 9.6. The number of points n that are removed has been chosen to be 10 in the final implementation, since this creates a satisfactorily smooth result. This way of updating the trajectories allows the intersections between segments generated to be smoothened.

Before the trajectory can be sent to the controller the heading, linear, and angular velocities must be calculated for each point on the trajectory. These parameters are calculated from the coordinates of two points on the trajectory. These points are chosen to be 10 points ahead and 10 points behind the point for which the parameters are calculated. For a trajectory of n points, those parameter are calculated from the differences on both axis between the coordinates of the two chosen points and from the sampling period of

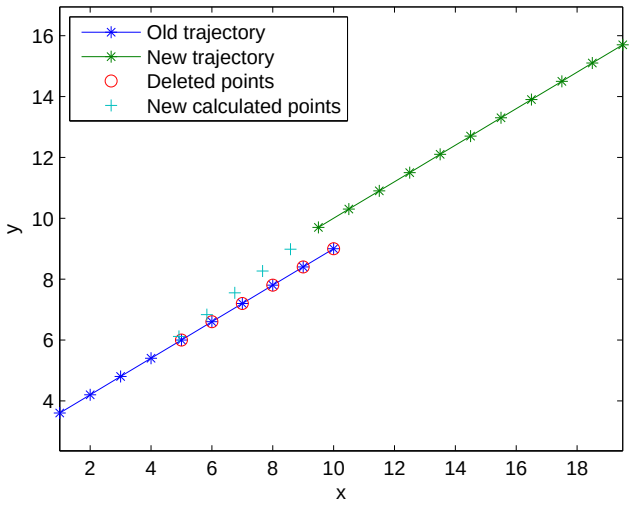


Figure 9.6. Smoothening of the trajectory with a number of replaced points $n = 6$

the controller $T_{controller}$ as:

$$\Delta x = \begin{cases} (x_{20} - x_0)/(20 \times T_{controller}) & \text{for the 10 first points} \\ (x_{n-21} - x_{n-1})/(20 \times T_{controller}) & \text{for the 10 last points} \\ (x_{i+10} - x_{i-10})/(20 \times T_{controller}) & \text{otherwise} \end{cases} \tag{9.1}$$

$$\Delta y = \begin{cases} (y_{20} - y_0)/(20 \times T_{controller}) & \text{for the 10 first points} \\ (y_{n-21} - y_{n-1})/(20 \times T_{controller}) & \text{for the 10 last points} \\ (y_{i+10} - y_{i-10})/(20 \times T_{controller}) & \text{otherwise} \end{cases} \tag{9.2}$$

The heading θ is calculated as:

$$\theta = \begin{cases} \arctan\left(\frac{\Delta y}{\Delta x}\right) & \text{for } \Delta x \geq 0 \wedge \Delta y \geq 0 \\ \arctan\left(\frac{\Delta y}{\Delta x}\right) + \pi & \text{for } \Delta x < 0 \\ \arctan\left(\frac{\Delta y}{\Delta x}\right) + 2\,\pi & \text{otherwise} \end{cases} \tag{9.3}$$

The linear and angular velocities can be calculated directly from the points on the trajectory by calculating the distance between successive points and comparing the angle difference between successive points respectively. It is however not chosen to do like this, since the combination of the trajectories does not makes perfectly smooth transitions, which would result in large angular velocities. Since the trajectory velocities is directly fed forward in the controller, velocities with such spikes will therefore create rapid movements of the robot, which is not desired.

Instead the linear velocity is set to the desired mapping velocity of 2.2 m/s. The angular velocity is set to zero. By doing so the feed forward from the trajectory is not used, therefore the vehicle will not be able to follow the trajectory as well as if the angular velocity had been defined. It is however considered more important to have a smooth drive compared to be precisely on the trajectory as long as the vehicle stays on the road.

To show the performance of the road detection and trajectory generation parts a video was captured. This is found on the CD `/videos/follow_road.mp4`. It should

however be stressed that the video does not show the final performance, since it was captured before final adjustments to the algorithms were made.

9.2 Pre-marking

The following section describes how the final pre-marking is performed. This part of the project has been given a lower priority compared to the other functionalities of the robot. The algorithm for creating the pre-marking of the road is therefore not in a final version. At this time the algorithm does only consider layout of one stripe at the time. This means that no precaution is taken to make sure that the two pre-markings in each side of the road are parallel and it can not design the layout of the centre line.

The first action performed in the pre-marking is to determine the placement of the road from the raw road edge data found in the road detection algorithm. Since these data have not been modified in any way they are generally edged in nature as seen plotted in figure 9.1. Examples of the raw road edge data can be seen in figure 9.7. Note that the data has been recorded from east to west in the figure. Due to the fact that the road detection algorithm runs at 5 Hz the raw road edge data is overlapping. In this figure all the data in between the four shown pieces have been removed to enhance visibility.

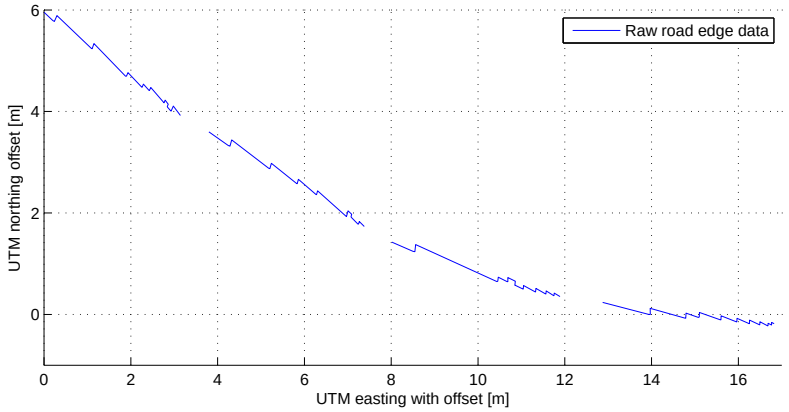


Figure 9.7. Example of raw road edge data directly from the road detection algorithm.

Since the raw road edge data is assumed most accurate closest to the vehicle, it is chosen only to work with the first sample in each road edge data piece and extract the overall road edge from this. The result of this thinning of data is plotted in figure 9.8. The extracted road edge data in this figure is what will be referred to in the rest of this section as *raw road edge*.

The angle and curvature of the road needs to be extracted from these road edge data. In order to perform this action without ending up with a result that is too affected by variations from sample to sample, the raw road edge data is filtered using a moving average. The next step that is performed is to divide this filtered road edge data in segments

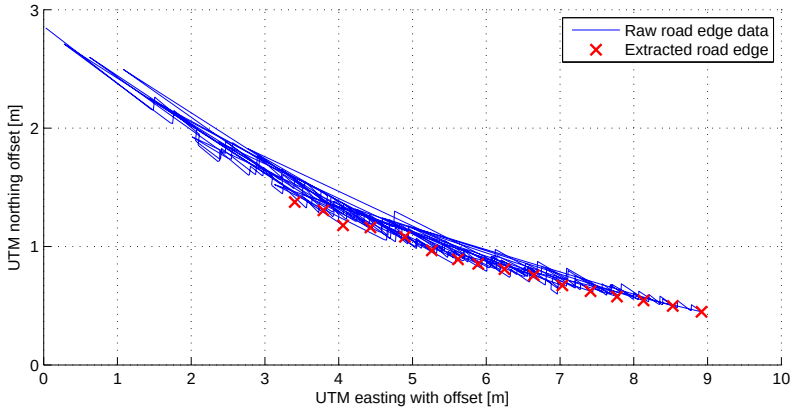


Figure 9.8. Extracted road edge data from the raw road edge overlapping pieces.

of approximately 4 m. For each segment the angle difference between the current and previous segment is calculated to determine if the road is curving or not. The information about whether the segment curves or not is saved along with each segment.

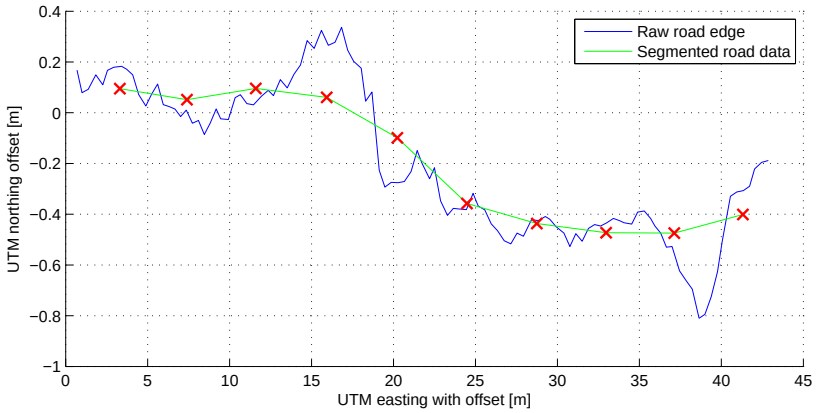


Figure 9.9. Segmented road data with a length of approximately 4 m.

The information about curve of the segment is used to put together new larger pieces of straight segments. In all cases where a straight segment is followed another straight segment these two segments are put together. This procedure is repeated with the new larger piece if the next segment is also straight and so on. All the road edge data is now projected to the inside of the road as seen in figure 9.10. Note that the offset road data shown in the figure has been thinned to enhance visibility. The angle at which this is performed is calculated from the new large pieces for all the straight parts. For the curved parts the angle between each road edge data point is calculated and used.

To generate the final smooth pre-marking, polynomial fitting is now used to represent the road stripe. Again the previous defined segments are used to find the approximation. The individual straight segments are combined as a first order polynomial with next straight segment if the mean square error between the polynomial and offset road edge data is sufficiently small. If it mean square error is too large the segments are split into

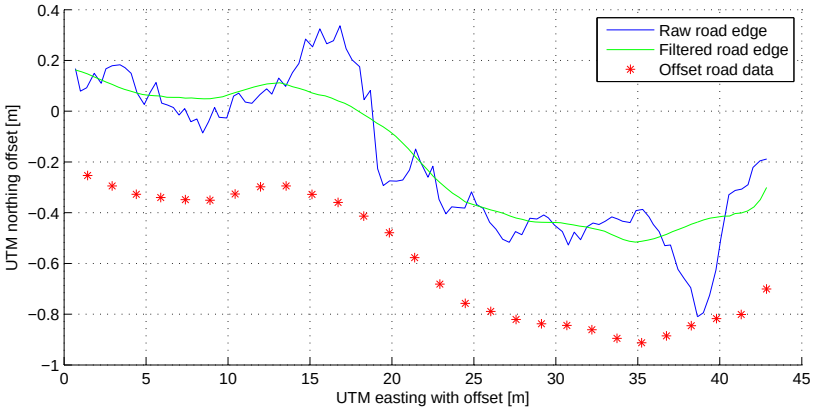
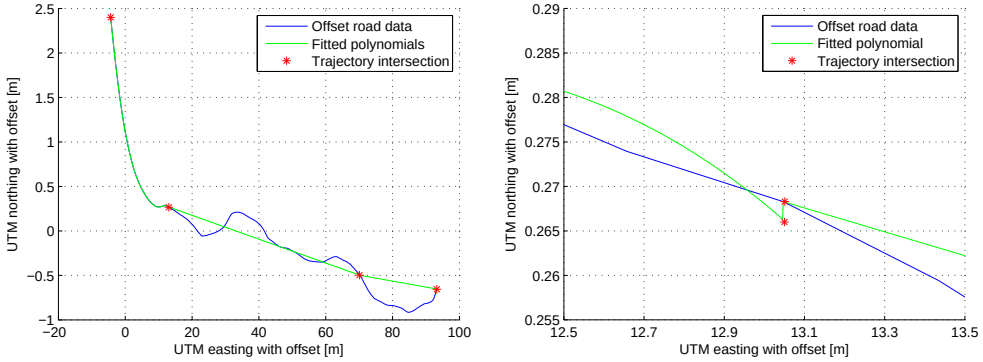


Figure 9.10. Orthogonally projected road data towards the inside of the road.

two individual first order polynomials. For the curved parts a fourth order polynomial is fitted to easting and northing offset road edge coordinates individually on the two axis. In figure 9.11 the fitted polynomials are shown along with the offset road data.



(a) Combined polynomials.

(b) Polynomial transitions.

Figure 9.11. Fitting polynomials to the offset road edge data.

When the whole offset road edge data set has been approximated it is sampled in pieces of length $v \cdot Ts$, where v is the desired velocity and Ts is the sampling time in the controller. For the generation of the pre-markings it is chosen to use a velocity of 1.3 m/s rather than 2.2 m/s to improve precision when following pre-markings. To smoothen the transitions between each polynomial, shown in figure 9.11(b), a moving average of the whole pre-marking is performed.

The last operations performed on the pre-marking data are to create a trajectory for the robot to follow. This implies calculating the $[x\ y\ \theta\ \omega\ v]$ vector.

In figure 9.12 it can be seen where the final pre-marking ends up with respect to the raw road edge data. As it can be seen the overall shape of the pre-marking seems to be a smooth representation of raw road edge data. However due to the moving average performed twice in the algorithm, the distance from the road edge to the pre-marking does not remain constant in bends as it should. Instead it will be smaller or larger depending

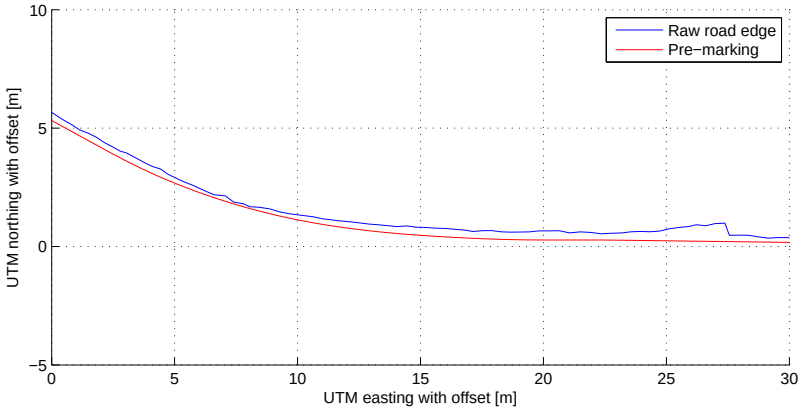


Figure 9.12. The final pre-marking plotted together with the raw road edge.

on which way the road curves. This effect can reach unacceptable limits in sharp turns and is therefore not good enough for the final product. At this point no compensation is done in order to enhance the performance of this algorithm.

10 Implementation

This chapter contains a description of the implementation of the designed system. An overview of the implementation is first presented after which more specific implementation details are presented

The software used to control the vehicle is based on the ROS framework [21]. The main benefit in using ROS is that multiprocess systems are easily implemented since all interprocess communication is handled by ROS. In the ROS framework processes are named nodes and interprocess communications are carried out by using either topics or services. Services are like ordinary function calls where one node calls a "service" on another node and a result is returned to the calling node. Topic communication is one-way communication where a node publishes some information without knowing if any node is using the published information. If a node needs the information from a topic it simply subscribes to this topic and thereby gets the data when it is published by the sender.

ROS provides an API for programming the nodes in both `c++` and Python and all nodes do not need to be programmed in the same language. In the implementation all nodes are programmed in `c++`. The source code for the nodes is found on the CD `/software/robotic_road_striper/src`.

In figure 10.1 an overview of the nodes and the inter node communication is sketched. As seen in the figure no services are used in the system. It is chosen to use topics rather than services partly because topic communication makes a looser binding between the nodes and partly because development is easier using topics, since it is easy to monitor what is send on a topic just by subscribing to it.

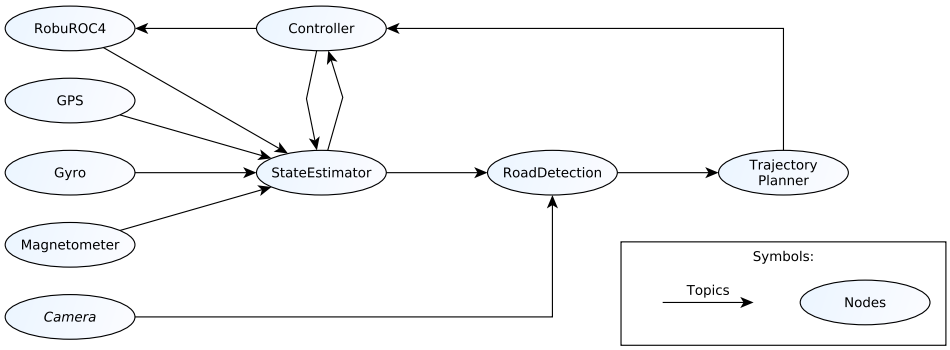


Figure 10.1. Overview of the software structure.

In table 10.1 the responsibilities for each node is described. The reason for the camera node to be marked with italic is that it is a part of the OpenNI Kinect driver from ROS [22], and is not designed or implemented by the project group.

Node name	Node responsibility
RobuROC4	Command the vehicle to drive at a certain linear and angular velocity. Measures the current angular and linear velocity.
GPS	Extract the data (position, heading and linear velocity) from the Ashtec MB 100 GPS and make it available for the other nodes.
Gyro	Extract angular velocity from the E•Core 2000 Fiber Optic Gyro and make it available for the other nodes.
Magnetometer	Calculate the heading of the vehicle from measurements from the OS5000 digital compass and make it available for the other nodes.
Camera	Handle the connection to the X-BOX 360 Kinect camera and makes images available for the other nodes *(This is a collection of nodes provided by the OpenNI Kinect driver [22])*
StateEstimator	Estimate the states of the vehicle using the extended Kalman filter, measurements from sensors, input to the vehicle, and a model of the vehicle.
RoadDetection	Detect the road edge from camera input.
TrajectoryPlanner	Generate trajectory either directly from the detected edge of the road or from previous knowledge about the position of the road.
Controller	Generate control signal to be send to the vehicle calculated from the states of the vehicle and a trajectory.

Table 10.1. Nodes and their responsibilities in the software solution.

In table 10.2 the interface between the nodes is described in terms of the content in the topics. The "Topic name" is the name of the topic generated by the sender node. This name is used by other nodes to access the information on the topic. The message type defines what type of information is included in the topic as an example the geometry_msgs::Pose2D includes three doubles: x, y, and theta. Other message types could contain fields of other message types or even vectors of other message types. The aim has been to use predefined message types rather than designing new message types to improve reusability. Message types prefixed by geometry_msgs:: or sensor_msgs:: are from predefined packages (see ROS wiki [21]). Message types prefixed by robotic_road_striper:: are special messages defined during this project. The definition of those message types are found on the CD /software/robotic_road_striper/msg.

To start up the system the launch functionality of ROS is used. Launch files are created and specifies which nodes to launch at system start up. The launch files for the system are found on the CD /software/robotic_road_striper/launch/. The system is launched by calling either of the two the following commands:

```
roslaunch robotic_road_striper road_mapping.launch
roslaunch robotic_road_striper road_striping.launch
```

The first launch file starts the system with all the needed nodes for following the road. After a launch using this file the robot is ready to follow the road when the controller is turned on (see section 10.5). The second launch file does not start up the vision part and is used for following trajectories from other sources than vision input. Before the controller is turned on a trajectory must be sent to the controller since no node, started by the launch file, publishes trajectories for the controller.

In the following sections the implementation of each node is described in terms of functionality and non trivial implementation issues.

Sender node	Topic name	Message type	Topic content
RobuROC4	/roc4/curVel	geometry_msgs::Twist	Linear and angular velocity.
GPS	/GPS/position	sensor_msgs::NavSatFix	Longitude and Latitude of the vehicle in degrees.
	/GPS/orientation	geometry_msgs::Pose2D	Heading of the vehicle.
	/GPS/velocity	geometry_msgs::Twist	Linear velocity of the vehicle.
Gyro	/Gyro/velocity	geometry_msgs::Twist	Angular velocity of the vehicle.
Magnetometer	/Magnetometer/orientation	geometry_msgs::Pose2D	Heading of the vehicle.
Camera	/camera/rgb/image_color	sensor_msgs::ImageConstPtr	Image from the Kinect camera.
StateEstimator	/states	robotic_road_striper::States	Estimated states of the vehicle.
RoadDetection	/roadEdge	robotic_road_striper::RoadEdge	Description of the edge of the road with respect to the vehicle and the states of the vehicle logged at the time where the image was captured.
Trajectory-Planner	/trajectory	robotic_road_striper::Trajectory	List of points to which the vehicle must drive.
Controller	/cmdVel	geometry_msgs::Twist	Desired angular and linear velocity.

Table 10.2. Topics published from nodes in the system.

10.1 Sensor and Actuator Nodes

For each sensor described in section 7.1 a node is created. These nodes handle the communication with the sensors and perform preprocessing on the input data by performing the modifications mentioned in sections 7.1.1-7.1.4. The sensor nodes are implemented in separate *.cpp files according to table 10.3.

Node	Source file
RobuROC4	/software/robotic_road_striper/src/roburoc4Node.cpp
GPS	/software/robotic_road_striper/src/gpsNode.cpp
Gyro	/software/robotic_road_striper/src/gyroNode.cpp
Magnetometer	/software/robotic_road_striper/src/magnetometerNode.cpp

Table 10.3. Implementation files for the sensor nodes.

The GPS, Gyro, and Magnetometer nodes communicate to their respective hardware component through a serial connection. To perform this communication a SerialCommunicator class has been implemented that allows either to get a single character from the serial device or get a whole line. The implementation of this SerialCommunicator class is found on the CD /software/helper_classes/serialCommunicator/serialCommunicator.cpp.

The data received from the GPS and Magnetometer is coded in NMEA ASCII strings where the data fields are separated by commas. The data is therefore extracted by separating the strings received from the strings by the commas. The data from the Gyro is for each message 8 Byte of binary data where the angular velocity is extracted by making bitwise manipulations to the binary data. For more information

about the communication protocol the specific sensor refer to the datasheets on the CD /literature/hardware_manuals.

To make the calibration of the magnetometer described in section 7.1.2 a calibration mode is implemented in the magnetometer node. The calibration mode is activated in the magnetometer node when a message is received on the `\calibMagnetometer` topic. When the calibration mode is activated the node stops publishing the heading to the heading topic, and starts to log the raw data from the sensor. The user is asked to turn the vehicle, and the magnetometer node stores the received data if the new sample differs enough from the old to be sure that data from all directions are stored. The magnetometer node stops collecting data when 100 samples of data has been received, and the ellipse fitting is conducted on those 100 samples. From the ellipse fitting the calibration parameters are extracted. These parameters are used to modify the calibration data and the calibrated data are shown (as in figure 7.1(b). The user is asked if he could approve the calibration. If this is the case the calibration parameters are stored and used for modifying the data used to calculate the heading. If the user cannot approve the calibration the calibration parameters are discarded and the old parameters are used. The magnetometer does then automatically leave the calibration mode.

The RobuROC4 node communicates with the vehicle via an Ethernet connection using User Datagram Protocol (UDP). The communication protocol for communication with the RobuROC4 is defined in the manual /literature/hardware_manuals/Robusoft_RobuROC4_User_Manual_including_communication_protocol.pdf. The data is sent in packages comprised by serialized binary data. To serialize and de-serialize the data the Boost serialization library is used since it does exactly what is needed [23]. The different telegrams are defined as objects and those objects are (de)serialized by use of the Boost serialization library. The implementation of the telegram classes are found on the CD /software/pure_protocol. To send and receive data from a socket the boost asio library is used [24].

10.2 State Estimation Node

The state estimation node implements the extended Kalman filter described in section 7.2. The source code for the node is found on the CD /software/robotic_road_striper/src/stateEstimationNode.cpp. The filter is setup to run with update frequency of 20 Hz using the ROS Rate class that ensures a node to wake up with the desired rate. When the node is woken up it listens for new information on the topics. From the new measurements from the topics, the measurement vector and covariance matrix for the measurements are updated and the filter equations described in chapter 7 are used to update the estimate of the states. Lastly the new found estimate of the system states are published to the states topic.

To make the matrix computations needed in the Kalman filter a third party library

called armadillo is used [25]. This library makes matrix definition and matrix manipulation in a MATLAB® like syntax.

10.3 Road Detection Node

This node implements the road detection functionality described in chapter 8. The source code for the node is found on the CD /software/robotic_road_striper/src/roadDetectionNode.cpp.

The node is woken up at a rate of 5 Hz using the ROS Rate class as both the road detection node and the trajectory generation node are able to process an image in 40 ms. At the beginning of each cycle, the node listens to the image topic to get a new image. In addition it listens to the topic containing the estimates of the states of the vehicle, to have the states of the vehicle at the time where the image is captured. Hereafter the road detection is carried out. When the road edge has been extracted from the input image it is published, along with the captured states, to the `/roadEdge` topic.

To make the image manipulations needed, the OpenCV Library is used [26].

10.4 Trajectory Planner Node

The Trajectory planner node implements the local trajectory generation described in chapter 9. The source code for this node is found on the CD /software/robotic_road_striper/src/trajectoryPlannerNode.cpp.

Since the trajectory planner node uses the data of the right edge of the road, it listens to the `/roadEdge` topic. The states of the robot when the image was captured is also read from this topic. Hereafter, the trajectory generation is performed. When the final trajectory has been calculated, it is published to the `/trajectory` topic.

To manipulate matrices, the node uses the OpenCV Library [26]. To make the linear interpolation, and the polynomial approximation, the GNU Scientific Library (GSL) is used [27].

The generation of the pre-markings described in section 9.2 should ideally be implemented in this node but due the lower priority given to this and time limits a MATLAB® implementation is made. The MATLAB® implementation can be found on the CD /matlab_code/trajectory_generation/pre-marking_gen.m.

10.5 Controller Node

The controller node implements the controller described in chapter 6. The source code for the controller node is found on the CD /software/robotic_road_striper/src/controllerNode.cpp. Since the controller must follow a trajectory the controller must include functionality to receive a trajectory from the `/trajectory` topic.

The node is woken up at a frequency of 20 Hz using the ROS Rate class. The first thing carried out in the beginning of each cycle is that the node listens for a new estimate of the states on the `/states` topic and it listens for a new trajectory on the `/trajectory` topic.

When a new trajectory is received the starting point is not guaranteed to be ahead of the vehicle and could equally well be behind. It is therefore chosen to select the starting point on the trajectory as the closest point from the vehicle. This starting point is found by iterating from the first point in the trajectory and calculating the distance to the position of the vehicle given by the current estimate of the states. When the distance starts to increase the iteration stops and the point is used as the first point the controller must follow.

After the node has finished listening for new states and trajectory, the generation of actuation signals is carried out. The error signals are calculated from the current point on the trajectory and the states of the vehicle and the control signals are calculated from equations (6.16) and (6.19). These control signals are hereafter published to the `/cmdVel` topic. The last thing happening in a control cycle is that the point to follow in the next cycle is found.

To be able to disable the controller two control modes are implemented (on and off). If the controller is off the actual control is not carried out and a velocity of zero is published to the `/cmdVel` topic for both angular and linear velocities. At each cycle the controller listens to the `/controller/controlMode` topic. If the Boolean value true is received the controller starts after 5 s if false is received the controller stops immediately.

10.6 Misc Nodes

In addition to the previously mentioned nodes four extra nodes have been implemented to aid in the development of the Robot. In this section the functionality of each of the four nodes will be described.

Menu node: This node is implemented to ease the startup of the controller and calibrate the magnetometer. Instead of manually publishing to the `/controller/controlMode` topic to start the controller or publishing to the `/calibMagnetomeret` topic to calibrate the magnetometer, this node is implemented to do this just by activating a menu item. The menu node is implemented using the Ncurses library [28].

Speed step node: This node is implemented to be able to set the input velocity to the vehicle to a given value. The node takes three arguments when it is launched: linear velocity, angular velocity, and time. The speed step node publishes the desired velocities to the `/cmdVel` topic to command it to drive with this velocity. Due to a security feature of the robot, the vehicle will only drive for one second if only one input is received, the

node will therefore continue sending the velocities until the desired time is reached. When the desired time is reached a zero velocity message is published to the `/cmdVel` topic to bring the vehicle to halt.

PlayStation® 3 Controller node: This node is implemented to be able to control the vehicle manually. Even though the vehicle is shipped with an X-Box controller which is fine for transportation uses it cannot be used for driving the vehicle when we need to know what input is given to the vehicle (for instance when the state estimate filter is running). The node depends on the joystick drivers for ROS which are capable of converting the input from the PlayStation® 3 controller to a `/joy` topic. In this topic the position of all buttons and joysticks can be extracted, and the input velocities to the vehicle can be calculated. The velocities are calculated when new information is received from the `/joy` topic. The calculated velocity is published to the `/cmdVel` topic at a rate of 20 Hz.

Trajectory publisher node: To be able to generate trajectories externally and thereafter let the robot follow them this node is implemented. As argument it takes the path to the file in which the trajectory is defined. The input file should be a `*.csv` file and each line should have the following format:

```
x,y,theta,angular_velocity,linear_velocity
```

Where the x and y coordinates are in the UTM coordinates, assuming that the vehicle is in the '32 V' zone. Theta is the desired heading (zero is east and positive direction counter clockwise). Angular and linear velocity are the velocities of the reference robot following the trajectory.

When the node is started up it reads the input file line by line and generates a trajectory message which is published to the `/trajectory` topic. When the trajectory is published the node shuts down again.

Part III

Conclusion & Perspectives

11 Acceptance Test

This chapter contains a description of the acceptance test used to evaluate if the designed and implemented solution can comply with the requirements specified in the requirement specification in chapter 3. In addition the results from the acceptance test will be presented and analysed.

To show the performance of the system, the vehicle should ideally be placed on a road and with a single command first start to map the road and thereafter make stripe on the road. The striping should be done following auto generated virtual pre-markings from a set of design rules, including both the number of stripes and where these stripes must be placed in the transverse direction of the road. After this process the painted stripes should be analysed to see if they comply with the requirement specification. This acceptance test will however diverge some from the ideal test for several reasons.

Instead of doing one large test, the acceptance test is divided into two tests. One test for each of the two major tasks in the road striping: mapping and striping. The reason for this division into parts is that the link between these two parts (the task for making the virtual pre-markings) has not been implemented on the vehicle and the algorithm still needs human hand to make adjustments for each data set to obtain the desired result.

Since no striping mechanism is included in the product, the actual stripes will not be painted on the road. The most accurate way available to measure the actual position of the vehicle and where the vehicle has driven is the logged GPS data. These data will be used for evaluating the precision of the striping process. By evaluating in this way, the capabilities of the vehicle in terms of following a virtual pre-marking will be evaluated, but no evaluation can be made of how the actual stripe will be placed on the road.

In the following sections the separate tests will first be described, and the results from these tests will then be presented and analysed.

11.1 Test Description

As described earlier it is chosen to divide the acceptance test into two separate tests. In the first test the road is mapped by letting the vehicle follow the road from the visual input. In the second test the vehicle is commanded to follow a virtual pre-marking.

In appendix C the measurement journal for the acceptance test is found. This includes the precise test procedure for the two separate tests. In the following sections a short description of the tests is given.

Test 1: Mapping the road

To determine the precision of the mapping of the road, the vehicle is commanded to follow the side of the road using visual input. The coordinates of the found road side are stored. These can then be used to determine how well the road side is determined by comparing coordinates of the road side to a map of the road. This test will also show the capabilities in terms of following the road since the vehicle must follow the road to be able to stripe it.

Test 2: Following virtual pre-markings

From the data collected in test 1, a virtual pre-marking is generated using a MATLAB® script implementing the functionality described in section 9.2. This script is found on the CD /matlab_code/trajectory_generation/pre-marking_gen.m. The vehicle is commanded to follow the generated pre-markings. This test will show how well the Autonomous Road Striper follows pre-markings and the precision of the pre-markings.

11.2 Test Results

To evaluate how well the system performs the results from the two separate tests will be discussed in the next two sections. In the last section the results are compared with the specific requirements in the requirement specification.

Road mapping capabilities

In figure 11.1 the mapping of the bike road is shown. The mapping consists of the road edge extracted in the vision part and hereafter converted into the global coordinate system. From the figure it is seen that most of the road is mapped by several representations of the road edge. This is caused by the high update rate of the road data compared to the velocity of the vehicle. Before making assessments on the precision of the mapping process, it is concluded that the vehicle is capable of following the road from visual input since a mapping of the road exists. Videos showing the mapping of the road from the robots perspective are placed on the CD /videos/road_detection_mapping_1.m4v and /videos/road_detection_mapping_2.m4v.

In figure 11.1 it is seen that the overall shape of the mapping forms a road with a width of approximately 3 m which complies with the fact that the mapped road is a bicycle road with a width of exactly 3 m.

In the straight part of the road when the easting is from −100 m to 0 m the road is not mapped perfectly in either of the two mapping runs. This could have several reasons. When the test was conducted some very sharp shadows were present which could cause the road detection algorithm to detect those shadows as the road edge and the detected road edge will turn into the road as seen several places. A second reason for *mapping 1* to have some deviations in this straight part is that some parts of the road edge consists

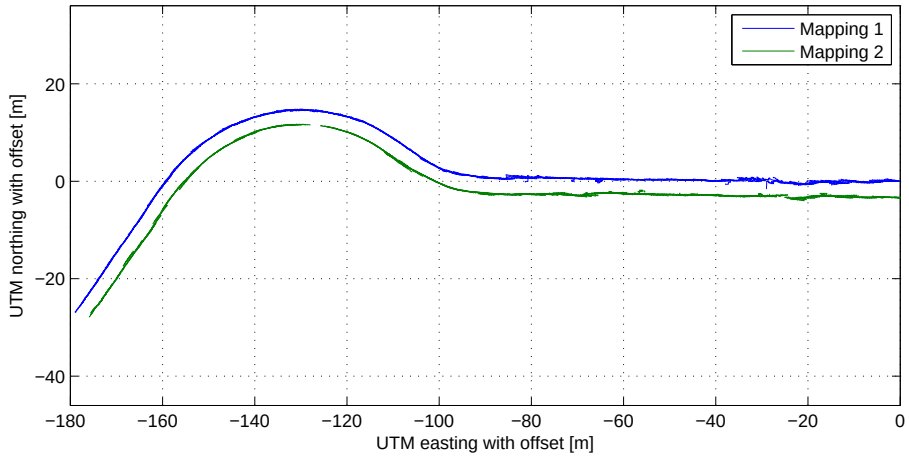


Figure 11.1. The two sides of the road mapped with the Autonomous Road Striper.

of gravel and not grass. This decreases the contrast between the road and the verge, and makes the separation harder.

A third and probably major reason for deviations is the position measured by the GPS. By inspection of the raw data from the GPS, it is seen that some sudden jumps in the position are present in this part of the road. These jumps will cause the raw data to be offset and could be the reason for these sudden jumps. In addition those jumps will cause an invalid heading measurement from the GPS which will imply that the detected road edge to be converted into the global coordinate system incorrectly. A potential reason for the GPS to have those jumps is that buildings are present on both sides of the road in this part.

In the area shown within easting from $-180\,\mathrm{m}$ to $-100\,\mathrm{m}$ those mentioned sources to imprecise measurements were not present. In figure 11.2 a zoom of a part of this is shown. In the *mapping 1* it is seen that the overall shape of all the detected road edges comply with each other i.e. there is no major jumps or some incorrectly transformed road edges caused by heading error.

The *mapping 2* part does however have some deviations. It looks like the transformation is okay since the direction of all the mapped parts comply. The sideways jumps are assumed to be caused by a green stripe painted on the road. In the mapping process it was observed that the vehicle suddenly started to drive with a constant offset from the green line on the road rather than with a constant offset from the road edge.

As seen in figure 11.2 has *mapping 2* a gap. This is assumed to be caused by the turn of the road and at this exact part the image contains too much of the road. The road detection algorithm will therefore disregard the image since it is assumed that other things that road are detected as road, when a too large part of the image is detected as road. When a lesser part of the image contains the road, the edge is again identified and extracted. This is verified by looking into the video from the vehicle perspective when mapping the second side of the road /videos/road_detection_mapping_2.m4v. Around time 44 s it is seen that the road is not detected for each image.

To see how well the actual mapping fits the road, the data shown in figure 11.2 is

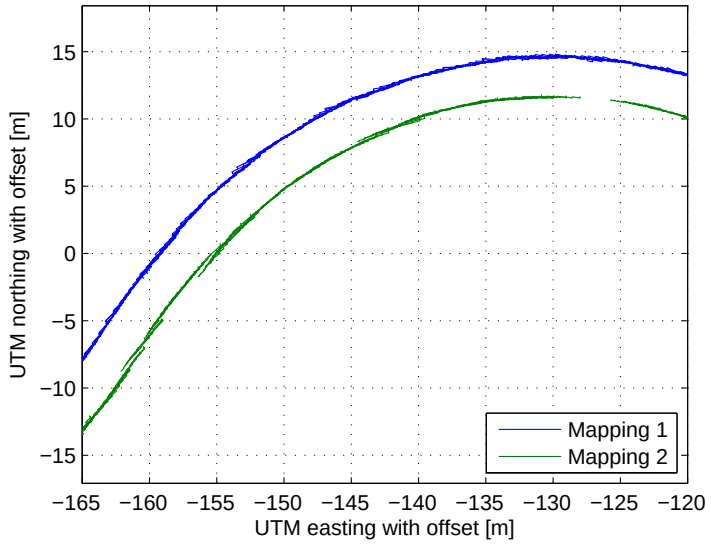


Figure 11.2. The two sides of the road mapped with the Autonomous Road Striper (zoomed).

plotted in figure 11.3 on an aerial photo. Since the precision of the map is not known an offset is added to the mapping to make the mapping and aerial photo fit.

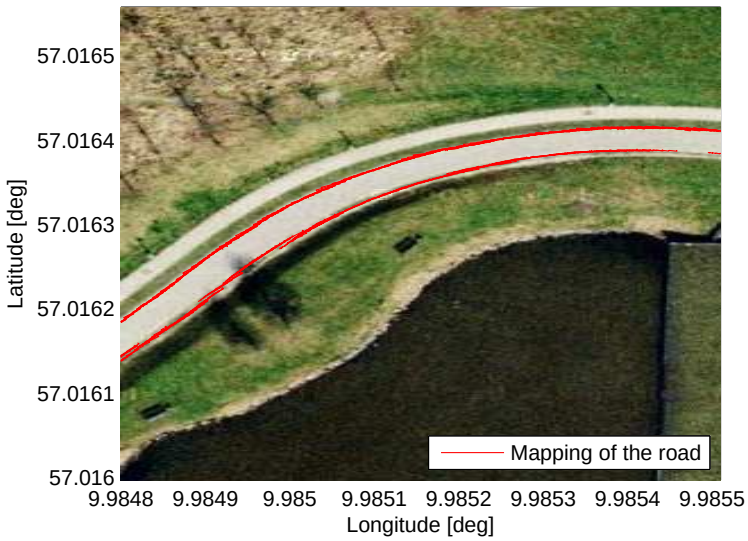


Figure 11.3. The two sides of the road mapped with the Autonomous Road Striper plotted on an aerial photo. ©2012 Google - Imagery ©2012 COWI A/S, DDO - Map Data ©2012 Google

Since the precision of the aerial photo is unknown it is not possible to say much about of the precision of the mapping. It is however seen that the mapping seems to fit the curvature seen on the image.

From this test it is seen that the Autonomous Road Striper is capable of following and mapping a road fairly well under ideal conditions. It does however have some problems when shadows are present on the road or the road edge is not clearly marked i.e. the verge

of the road has almost the same colour as the road. In addition bad GPS measurements seem to cause problems when mapping the road.

To shown the performance of the vehicle in terms of following the road, videos were captured during the to mappings of the road. These videos are found on the CD /videos/mapping_1.MOV and /videos/mapping_2.MOV.

Capability of following virtual pre-markings

In figure 11.3 the estimated position of the vehicle is plotted for the test where the vehicle is commanded to follow a pre-marking in form of a trajectory. This pre-marking is generated from the road edge detected in the mapping process. As seen in the figure, the robot is capable of following the trajectory with some deviation.

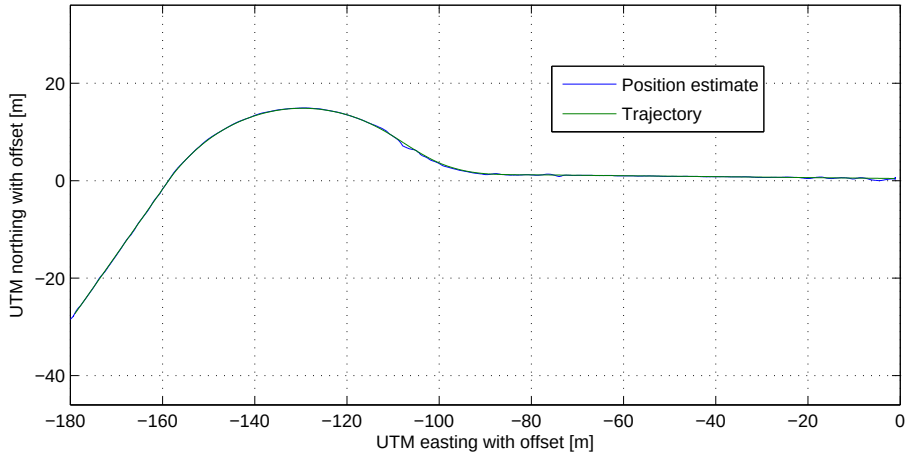


Figure 11.4. The estimated position of the vehicle and the trajectory to follow during the striping process.

To show the distance between the estimated position and the trajectory, the errors used in the controller are used. It is chosen to use those since they are considered as the most precise measurement at hand. The errors of the controller during the striping process shown in figure 11.4 are shown in figure 11.5

In the figure all errors of the controller are presented. The most interesting is however the error in the transverse direction of the trajectory p_{ey}. From the figure it is seen that this error is within $\pm 0.2\,\mathrm{m}$ but is larger in the beginning before the vehicle has converged to the trajectory and while the vehicle turns which happens between time 50 s and 90 s. The largest error is observed at time 80 s where the distance to the trajectory is up to 0.6 m. When the trajectory is straight, the error stays less than $\pm 0.1\,\mathrm{m}$ for longer periods of time. This is seen in figure 11.5 between time 20 s to 50 s and again from 90 s to 180 s. The main reason for the larger error in turns is caused by a logical error in the pre-marking generation, which is identified after completion of the acceptance test. The rotational velocity part of the trajectory was defined with the opposite turning direction,

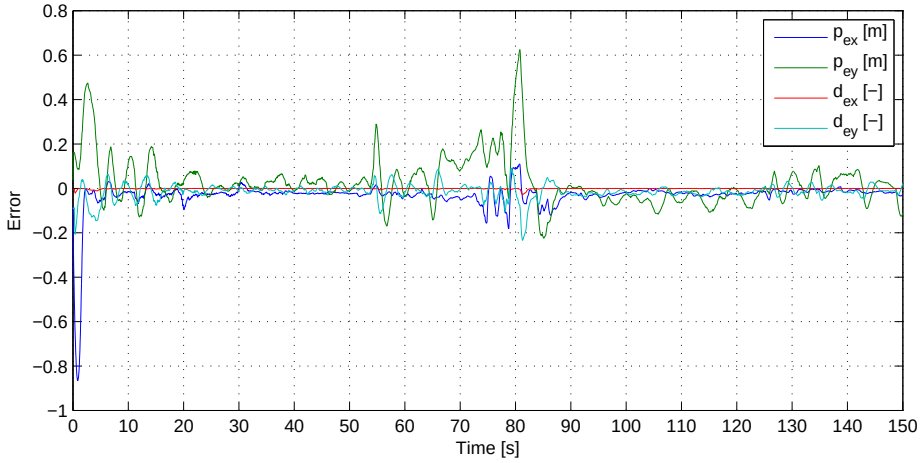


Figure 11.5. Controller errors measured during the striping process.

such that when the trajectory is turning right, the feed forward from the trajectory makes the vehicle turn left and a larger error is therefore obtained.

From this test it is shown that the Autonomous Road Striper is capable of following pre-markings with a precision down to 0.1 m for the straight parts while the much less precision in the turns due to a logical error in the pre-marking generation algorithm.

Assessments on the specific requirements

As shown in the two previous sections the Autonomous Road Striper is capable of both mapping and virtually striping a piece of road. The precision is however not sufficient to comply with requirement **r.1** stating that the stripes must be painted with a precision of ± 0.05 m.

During the test it was observed that the GPS drifts. This is not visible from the figures in the mapping part since the mapping of the second side was carried immediately after the mapping of the first side. Between the mapping and striping the pre-markings were generated and approximately 40 min went by from the mapping to the striping. In this period the GPS had drifted such that instead of driving on the road, the vehicle drives in the grass in the side of the road approximately 1 m offset from the desired path. From the specifications of the GPS, it should give a global measured position within 2 cm in RTK mode. This drifting should therefore not be possible, and some problems with this RTK system must be present.

Due to both of the above mentioned problems the precision requirement **r.1** is not fulfilled. In the design it was chosen to decrease the velocity while striping to 1.3 m/s to improve precision. This however implies that the average velocity of the vehicle will be 5.6 km/h and the striping velocity will be $\frac{5.6\,\text{km/h}}{5} = 1.12\,\text{km/h}$ without taking account for the time it takes to turn. Since requirement **r.2** states that the average velocity must be at least 1.4 m/s when 3 stripes is to be painted this requirement is also not fulfilled.

12 Conclusion

This project has treated the topic of painting road stripes on public highways and motorways autonomously. The project has been focusing on how the process of road striping can be automated using an autonomous mobile robot without having precise whereabouts of the road beforehand.

Existing solutions for road striping have been analysed. The current way of striping roads is to pre-mark the road using a machine and then follow the pre-marks to stripe the road using another machine. The road striping regulations have been investigated in order to determine how road stripes must be placed on the road to comply with existing rules. It was suggested to adopt the existing approach of pre-marking the road before striping, by creating a map of the road before striping. To create this map autonomously a visual detection of the road must be created and for this reason different approaches of detecting a road from an image has been examined.

Based on the analysis, a requirement specification has been set up. Two requirements were defined which the autonomous robot had to comply with in order to be useful for road striping. These two requirements regard the minimum precision and velocity.

The precision requirement states that the robot must not diverge more than $\pm 5\,\mathrm{cm}$ from the desired position in the transverse direction of the road. The velocity requirement has been set to be as fast as possible but at the same time respect the hardware limitations of the robot used. While the maximum velocity of the vehicle is $9\,\mathrm{km/h}$ it has been chosen to define the minimum striping velocity such that the average velocity of the vehicle is $7\,\mathrm{km/h}$ giving some headroom for turning in the ends etc. Besides these two specific requirements, the robot movement should be made as smooth as possible to reduce wobbling of the stripes.

A model of the robot has been deduced including both the kinematic and the dynamic behaviour of the robot. To model the behaviour of the robot its, unknown internal controller was taken into consideration using a grey-box model. A controller has then been designed based on the model to allow the robot to track a predefined trajectory. To be able to perform well, the controller needs to have precise information about the states of the robot. These states are obtained from different sensors which are filtered using an extended Kalman filter.

From a camera mounted on the robot, pictures of the road were taken and the edge of the road was detected in a road detection algorithm. The detected edge is transformed into a global coordinate system. This edge representation is used to create the trajectories that the robot must follow in both the mapping and the striping part.

The solution has been implemented in `C++` using the Robotic Operating System (ROS) where each part has been implemented as different processes called nodes.

To be able to verify if the robot is able to fulfil the requirements stated, an acceptance test has been designed. This acceptance test has been split into two distinct parts; mapping the road using visual input and following the pre-markings generated from the map.

The robot was able to map the two edges of a road with both curves and shadows present. When the detected road edges are superimposed over an aerial image of the road the overall curvature of the extracted road edges fits the road in the image. However when the map of the road is studied in greater detail, the edge can be observed to diverge slightly in some places. These deviations are caused by sudden jumps in the recorded GPS position.

In the second part of the acceptance test the performance in terms of following a pre-marking is determined by letting the robot follow the pre-markings generated from the map determined in the mapping part. The robot was able to follow a pre-marking with a precision better than $\pm 10\,\text{cm}$ while driving straight but with deviations up to 60 cm while turning. It should however be noted that due to drift in the GPS, the pre-markings had moved approximately 1 m with respect to the road. During this test the average velocity of the vehicle was determined to be $5.6\,\text{km/h}$ if three stripes had to be painted on the road.

From these test results it is seen that the designed system is not capable of complying with the two requirements stated in the requirements specification. One of the main reasons for this is that there were problems with the position measured by the GPS. The test of the controller does however show that the robot is able to follow a predefined trajectory with a precision better than $\pm 6\,\text{cm}$ for straight parts and better than $\pm 20\,\text{cm}$ while turning, even though the turns are sharper than the ones in the acceptance test. During this test the GPS generally showed better precision.

All in all the acceptance test shows that the Autonomous Road Striper (ARS) is capable of mapping a road using visual input. It also shows that the robot is able to follow pre-markings, just not with the desired precision. Since the lack of accuracy while following the pre-markings is caused by GPS problems, the first place to improve the performance would be to look into this problem.

13 Perspectives

Over the last few decades automation of processes have been implemented everywhere in our surroundings. Some of the major motivating forces are reducing cost, production time, and achieve a repeatable high quality. While advanced technology has been become increasingly cheaper and computational power has grown, more and more complex tasks can nowadays be automated by use of robots. From this context it is only natural to look into new areas that have not yet been automated.

In terms of striping roads, the main advantage of creating an autonomous robot is to be able to reduce the cost. With the current road striping methods workers must be operating the machines pre-marking and striping the road all the time. An autonomous road striper should only require attention for recharging the battery and refilling the paint. For this reason one operator could be given responsibility of servicing multiple robots.

As mentioned in the conclusion, the Autonomous Road Striper (ARS) seems to be able to perform decent when the GPS is accurate. Consequently, the first improvement that could be brought to the ARS would be to make a deep analysis of these GPS problems to minimise or remove these. Even during the best runs, it does however seem like the drive is generally not smooth enough for mounting a paint gun directly on the robot. To enhance this performance it could be a possibility to add a fast moving painting gun on the robot. The objective of this arm would then be to make small fast adjustments in the transversal direction of the road to achieve a smoother striping.

The robot used during this project uses a four wheeled skid steering principle in order to steer. However such a principle is not optimal for achieving a smooth drive, since turning in general moves both ends of the vehicle in sudden movement. Additionally it is observed that turning the vehicle this way propagates some vibrations through the robot which in the end will affect the painting gun. It seems natural to overcome these problems by choosing a front wheel steering vehicle. This would generally allow a smoother driving resulting in straighter stripes.

In terms of speed the ARS can not compete with existing solutions mounted on trucks. The principles used in the product does however not contain any constraints limiting the velocity. If a faster robot had been available the task could have been performed faster. Generally the most critical aspect in terms of speed is the road recognition which takes some time to compute. With a faster computer nothing would prevent the frame rate from being increased and therefore the mapping speed could be increased.

Acronyms

API Application Programming Interface

IMU Inertial Measurement Unit

ROS Robotic Operating System

UTM Universal Transverse Mercator

RTK Real Time Kinematic

UDP User Datagram Protocol

NMEA National Marine Electronics Association

GSL GNU Scientific Library

ARS Autonomous Road Striper

Bibliography

[1] The Danish Road Directorate. Accessed: Feb 2012. [Online]. Available: http://www.vejdirektoratet.dk/roaddirectorate.asp?page=dept&objno=1024

[2] Zhengzhou Dayu Machinery Co., Ltd. Accessed: Feb 2012. [Online]. Available: http://www.road-marking-machine.com/products/auxiliary-equipment/hand-push-pre-marker.html

[3] Linetech Design & Mfg Ltd. Accessed: Feb 2012. [Online]. Available: http://www.linetechdesign.com/linetech/index.php?option=com_content&task=view&id=23&Itemid=44

[4] Roadlazer line stripers. Graco. Accessed: Feb 2012. [Online]. Available: http://wwwd.graco.com/Distributors/DLibrary.nsf/Files/300430/$file/300430I.pdf

[5] Graco Inc. Accessed : Feb 2012. [Online]. Available: http://www.graco.com/Internet/T_PDB.nsf/ProdGroups/ContractorStriping

[6] J.D. Honigberg International, Inc. Accessed : Feb 2012. [Online]. Available: http://www.jdhpowerequipment.com/kelly.html

[7] (2006, jul) Afmærkning på kørebanen. hæfte 0. generelt. The Danish Road Directorate. Accessed: Feb 2012. [Online]. Available: http://webapp.vd.dk/vejregler/pdf/VR05_G_Afm_koerebanen_H0_Generelt_V2_061004_HCD.pdf

[8] (2006, jul) Afmærkning på kørebanen. hæfte 1. længdeafmærkning. The Danish Road Directorate. Accessed: Feb 2012. [Online]. Available: http://webapp.vd.dk/vejregler/pdf/VR05_G_Afm_koerebanen_H1_Laengdeafm_V2_061004_HCD.pdf

[9] (2006, jul) Afmærkning på kørebanen hæfte 6. dimensioner. The Danish Road Directorate. Accessed: Mar 2012. [Online]. Available: http://webapp.vd.dk/vejregler/pdf/VR05_G_Afm_koerebanen_H6_Dimensioner_V2_061008_HCD.pdf

[10] T. Lei, Y. Fan, and L. Huang, "Fast lane recognition based on morphological multi-structure element model," *Optoelectronics Letters*, vol. 5, no. 4, pp. 304–308, 2009.

[11] M. Montemerlo, S. Thrun, H. Dahlkamp, and D. Stavens, "Winning the darpa grand challenge with an ai robot," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*, 2006, pp. 17–20.

[12] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, vol. 3. Spie Bellingham, WA, 1997, p. 26.

[13] R. Aufrère, R. Chapuis, and F. Chausse, "A dynamic vision algorithm to locate a vehicle on a nonstructured road," *The International Journal of Robotics Research*, vol. 19, no. 5, pp. 411–423, 2000.

[14] J. Álvarez and A. Lopez, "Road detection based on illuminant invariance," *Intelligent Transportation Systems, IEEE Transactions on*, no. 99, pp. 1–10, 2010.

[15] G. Cook, *Mobile Robots: Navigation, Control and Remote Sensing.* John Wiley & Sons, 2011.

[16] 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.

[17] robusoft, "roburoc 4 the powerful outdoor mobile robot for civil and military applications," 2012.

[18] M. Knudsen. Experimental modelling of dynamic systems. Accessed : May 2012. [Online]. Available: http://www.control.auc.dk/~mk/ExpMod/Publicdoc/LectureNote02pdf.pdf

[19] H. Schioler and T. Ngo, "Trophallaxis in robotic swarms-beyond energy autonomy," in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on.* IEEE, 2008, pp. 1526–1533.

[20] G. F. Franklin, A. Emami-Naeini, and J. D. Powell, *Feedback Control of Dynamic Systems*, 6th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2010.

[21] ROS.org, "Robot operating system," May 2012. [Online]. Available: http://www.ros.org/wiki/

[22] ——, "Ros - openni kinect driver," May 2012. [Online]. Available: http://www.ros.org/wiki/openni_kinect

[23] boost, "boost - serialization," May 2012. [Online]. Available: http://www.boost.org/doc/libs/1_49_0/libs/serialization/doc/index.html

[24] ——, "boost - asio," May 2012. [Online]. Available: http://www.boost.org/doc/libs/1_49_0/doc/html/boost_asio.html

[25] NICTA, "Armadillo - c++ linear algebra library," May 2012. [Online]. Available: http://arma.sourceforge.net/

[26] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[27] M. G. et al, *GNU Scientific Library Reference Manual (3rd Ed.)*, no. ISBN 0954612078. [Online]. Available: http://www.gnu.org/software/gsl/

[28] Ncurses. Accessed: May 2012. [Online]. Available: http://www.gnu.org/software/ncurses/

Appendix

A Provided Hardware

In this appendix the provided hardware for solving the task of striping roads is described. First the provided hardware is presented after which the composition of the hardware parts are described.

In table A.1 the provided hardware is listed. For specific information about the individual devices, refer to the datasheets on the CD /literature/hardware_manuals.

Devise type	Devise model
Vehicle	RobuROC4
GPS	Ashtec MB100
Digital compass	Ocean Server 5000
Fibre optical gyro	KVH E•Core 2000
Laptop	Toshiba TECRA M11-134
Camera	Xbox 360 Kinect

Table A.1. The provided hardware.

The base for the composition is the vehicle on which an antenna mast and a camera mount is attached (see figure A.1, A.2, and A.3). On the antenna mast the digital compass is mounted as high as possible to reduce the disturbances in the magnetic field caused by the vehicle. The GPS antenna is also mounted on the antenna mast whereas the GPS device is placed directly on the vehicle. The fibre optical gyro is also mounted directly to the top of the vehicle. The placement of these sensors is shown on the image in figure A.5.

As seen on the image in figure A.5 the control laptop is mounted directly on the vehicle. Even though the communication to all the sensors is via serial RS232 connections they are connected to the vehicle via RS232 to USB converters. The Camera is connected to the laptop directly via USB connection.

The camera is mounted on the antenna mount as indicated in figure A.1 and A.2 and is turned upside down to make the mount more firm since the original camera stand is not capable of keeping the camera still while running. In figure A.4 the mount is shown. As seen in the figure a wedge is inserted to angle the camera to be able to capture more of the road closer to the vehicle and decrease the perspective effect.

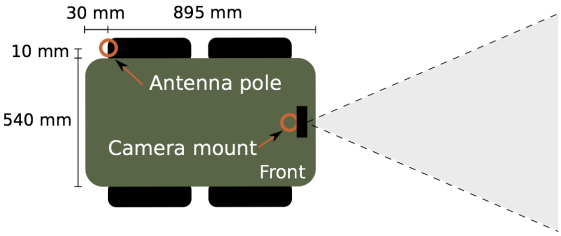


Figure A.1. Vehicle top view.

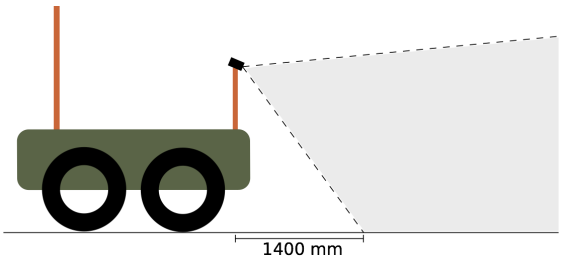


Figure A.2. Vehicle side view.

Figure A.3. The composition of the provided hardware.

Figure A.4. Camera mount.

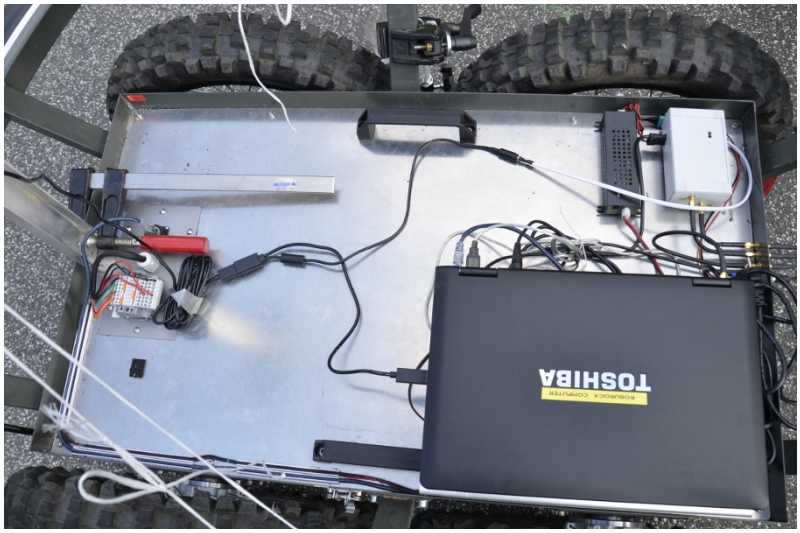


Figure A.5. Placement of hardware on the vehicle.

B Model Verification and Sensor Variance Determination

B.1 Formalities

This measurement journal describes the test conducted to verify the model of the RobuROC4 and to determine the variance of the measurements from the sensors.

Participants:

Niels H. Andersen

Rune Madsen

B.2 Test Location

The tests are performed at two different locations:

1. Parking lot in front of Fredrik Bajers Vej 7, 9220 Aalborg Ø.
2. Vicon motion tracking in room C2-111.

In the motion tracking room, a Vicon motion tracking system allows an object mounted with reflective markers to be detected with very high precision. This system is used to capture the motion of the RobuROC4. The room however has a limited amount of space and for that reason only the rotational tests are be conducted there. The rest of the experiments are performed outside on a parking lot since they require more space.

B.3 Logging of Data

In all tests raw sensor data from the sensors are logged at 20 Hz. The raw data from the sensors is saved in `*.csv` format with a filename containing the respective sensor name postfixed by the date and time at which the data was recorded. For tests performed in the Vicon motion tracking room, the raw data is save to data to a file directly from the MATLAB® workspace as `*.mat` files with a sampling frequency of 100 Hz.

During the experiments different inputs must be applied to the robot. This is done in two different ways. For all of the tests except one, a speed step node written in `C++` is used (for more information about notes etc. see the Implementation chapter 10). This node simply commands the robot to drive with certain linear or rotational velocities for a specified amount of time. For the last test a PlayStation® 3 controller connected to the onboard computer via Bluetooth connection. The onboard computer receives the Bluetooth signal, saves in to a file and passes the commands on to the RobuROC4.

B.4 Test Procedure

The following section describes the procedure of each test. In all the following tests the vehicle starts out from a standstill and an input is then applied. Note that in some of the tests multiple runs are be conducted. All the recorded data from the tests can be found in `/matlab_files/test_data/`. The input for each of the five tests is stated in table B.1. This table also states the filename postfix's of the recorded sensor data. In test number two the data recorded by the motion tracking system is saved in individual files while the `*.csv` file contains data from all the tests.

Test no.	Use	Input method	Input	date postfix
1	Model verification	Speed step node:	1.3 m/s and 0 rad/s for 5 s 0 m/s and 0 rad/s for 1 s	2012_05_11_09_23
2	Model verification and sensor variance	Speed step node:	0 m/s and 2.96 rad/s for 5 s 0 m/s and 0 rad/s for 1 s	2012_05_14_16_24 ang_2-96_10sec.mat
		Speed step node:	0 m/s and 0.33 rad/s for 5 s 0 m/s and 0 rad/s for 1 s	ang_0-33_10sec.mat
3	Model verification	Speed step node:	1 m/s and 0 rad/s for 2 s 1 m/s and 1 rad/s for 1.5 s 1 m/s and 0 rad/s for 2 s 1 m/s and −1 rad/s for 1.5 s 1 m/s and 0 rad/s for 2 s 0 m/s and 0 rad/s for 1 s	2012_05_11_09_44
4	Model verification	PS3 controller:	Smooth drive around parking lot	2012_05_11_09_36
5	Sensor variance	Speed step node:	2.2 m/s and 0 rad/s for 18 s 0 m/s and 0 rad/s for 1 s	2012_05_11_09_24
		Speed step node:	0.75 m/s and 0 rad/s for 52 s 0 m/s and 0 rad/s for 1 s	
		Speed step node:	0.5 m/s and 0 rad/s for 80 s 0 m/s and 0 rad/s for 1 s	
		Speed step node:	0.25 m/s and 0 rad/s for 52 s 0 m/s and 0 rad/s for 1 s	
6	Sensor variance (GPS)	Nothing keeping the vehicle standing still logging the position		2012_04_19_13_45

Table B.1. Test procedures

B.5 Remarks

During the analysis of the recorded data it was observed that the time each signal was applied to the robot was not exactly as specified in table B.1. Since the speed step node is just implemented with the `sleep()` C-function, the real time performance is pore. This does however not affect the validity of the tests as long as the real input is used for comparison.

C Acceptance Test Measurement Journal

C.1 Formalities

This measurement journal describes the test conducted in connection with the acceptance test of the Autonomous Road Striper.

Date: 27th of May, 2012

Participants:

Niels H. Andersen

Rune Madsen

Maxime Nollet

Alexandre Slove

C.2 Test Location and setup

The tests are performed on a bicycle road close to Fredrik Bajers Vej 7, 9220 Aalborg Ø. The setup consists of the device tested device (the Autonomous Road Striper) including the base station for the RTK GPS. The test setup also includes a Xbox controller used to manually operate the RobuROC4.

Data from the Autonomous Road Striper is logged to files while driving. This includes internal controller variables and the extracted road in the global coordinate system. The data is stored in `*.csv` format with an expressive file name for the content of the file postfixed by the date and time.

C.3 Test Procedure

The test is divided into two parts. A mapping part and a striping part. In the test one piece of bicycle road is virtually striped i.e. the robot must follow the two sides first to map the road where after it must run in both sides to stripe the road.

Test 1: Mapping the road

For the mapping part, the test procedure is as follows:

1. The vehicle is placed close to the edge of the road by using the Xbox controller.

2. The system is started up calling: `roslaunch robotic_road_striper road_mapping.launch` in a terminal window.
3. The menu node is started calling: `rosrun robotic_road_striper menu_node` in another terminal window.
4. The controller is turned on by selecting the *Controller ON* menu item.
5. When the vehicle reaches the end of the road, the controller signals is overruled by pressing the A-button button on the Xbox controller and the vehicle halts. The controller is turned off by selecting the *Controller OFF* menu item.
6. The menu is closed by choosing the *Exit* menu item, and the rest of the system is terminated by pressing `ctrl+c` in the first terminal window.

The vehicle is turned around and the other side of the road is mapped using the exact same procedure.

The road data logged in the mapping of the road is converted to trajectories using the MATLAB® script implementing the pre-marking algorithm. The script for generating the pre-markings is found on the CD /.... The output of this script is a `*.csv` file containing a trajectory that the vehicle must follow. The generated output trajectories are placed on the CD:

1. /matlab_code/trajectory_generation/generated_pre-marking/trajec_1.csv
2. /matlab_code/trajectory_generation/generated_pre-marking/trajec_2.csv

Test 2: Striping

For the striping part, the test procedure is as follows:

1. The vehicle is placed close to the starting point of the desired position of the stripe by using the Xbox controller.
2. The system is started up calling: `roslaunch robotic_road_striper road_striping.launch` in a terminal window.
3. The menu node is started calling: `rosrun robotic_road_striper menu_node` in another terminal window.
4. The trajectory is loaded into the controller by calling `rosrun robotic_road_striper trajectory_publisher_node trajec_1.csv` in a third terminal window.
5. The controller is turned on by selecting the *Controller ON* menu item.
6. When the vehicle reaches the end of the trajectory the controller is turned off by selecting the *Controller OFF* menu item.
7. The menu is closed by choosing the *Exit* menu item and the rest of the system is terminated by pressing `ctrl+c` in the first terminal window.

The vehicle is turned around and the other side of the road is striped using the same procedure with the only difference that the `trajec_2.csv` is used instead of `trajec_1.csv` in step 4.

C.4 Test Results

All the logged data is stored for the performed runs. During the test the GPS gave some problems, therefore only striping of the first side is performed. All the logged data is placed on the CD /matlab_files/test_data/2012_05_27_acceptance_test/. The date postfix for the performed runs are stated in table .

Test	Date postfix
Mapping 1	2012_05_27_14_44
Mapping 2	2012_05_27_14_47
Striping 1	2012_05_27_15_22

Table C.1. Filename poxtfixes for logged data in the acceptance test.