NTNU – Trondheim
Norwegian University of
Science and Technology

Autonomous Navigation And Row Detection in Crop Fields Using Computer Vision

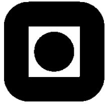
Jarle Dørum

Master of Science in Cybernetics and Robotics
Submission date: May 2015
Supervisor: Jan Tommy Gravdahl, ITK
Co-supervisor: Trygve Utstumo, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Adigo mekatronikk

Norges teknisk-naturvitenskaplige Universitet
IME-fakultetet
Institutt for Kybernetikk og robotikk



Thesis Assignment TTK4900

Student: Jarle Dørum
Program: M.Sc. Engineering Cybernetics

Title: **Autonomous Navigation and Row Detection in Crop Fields Using Computer Vision**

Adigo is developing an autonomous robot for weed control in row crops, named *Asterix*. A vision system classifies weeds and crops and subsequently targets each weed leaf with an individual droplet of herbicide. The system requires a highly accurate position and movement estimate to accurately target individual leaves. The robot is equipped with an RTK GPS, IMU, wheel encoders and a camera system which can be utilized for visual odometry.

Asterix is a differentially steered mobile platform with two actuated front wheels and a passive off-center rear caster wheel. The robot will drive autonomously in the tractor wheel tracks and treat the row crop in between the wheels. While *Asterix* is a differentially steered robot with an off-center rear caster wheel, a similar but much smaller robot with a centered rear caster wheel, named *Dogmatix*, will be used for most of the testing.

The overall task of this thesis is to develop a system that is able to identify row crops using computer vision and follow the rows.

Assignments:
- Research relevant theory about computer vision.
- Develop a method of identifying row crops from images and estimate the position and orientation.
- Develop a controller that is able to follow the identified row crops.
- Find a way to handle robot behavior when reaching end of a row, i.e. recognize the end, turn around, and identify the next row.
- Develop a simulator that simulates the entire system, with realistic robot and fields.
- Perform experiments with both the Dogmatix and Asterix robots. Compare the results with the same experiments conducted in the simulator.
- Write and submit a paper based on last semester's results to a suited conference.

Advisor: Jan Tommy Gravdahl
Professor, Dept. of Engineering Cybernetics

External advisor: Trygve Utstumo
Ph.D. candidate, Dept. of Engineering Cybernetics
MSc. Engineering Cybernetics, Adigo AS

Project assigned: January 5, 2015
To be handed in by: June 1, 2015

Trondheim, January 2015

____

Jan Tommy Gravdahl
Professor, Dept. Engineering Cybernetics

ABSTRACT

This thesis documents the development of an autonomous row crop guidance system for a differentially wheeled agricultural robot. Computer vision is used to identify rows and estimate their position and orientation. The estimates are used as measurements in a Kalman filter, before being supplied to a row controller which attempts to make the robot follow the row closely. A simulator with a realistic robot and fields have been developed, and all methods were tested on real robots. Tests have been done on an indoor recreation of a field and outdoors on a real carrot field. The result is a system which is able to autonomously traverse an entire field in simulations and recreated fields.

SAMMENDRAG

Denne avhandlingen dokumenterer utviklingen av et autonomt radfølgingssystem for en differensielt styrt jordbruksrobot. Maskinsyn er brukt til å identifisere jorderader og estimere radenes posisjon og retning. Estimatene brukes som målinger i et Kalman filter, som videre blir brukt i en radkontroller som forsøker å få roboten til å følge raden så nært som mulig. En simulator med en realistisk robot og jorder har blitt utviklet, og alle metoder ble testet på ekte roboter. Tester har blitt utført på et innendørs gjenskapt jorde og utendørs på et ekte gulrotjorde. Resultatet er et system som er i stand til å autonomt traversere et helt jorde med simuleringer og gjenskapte jorder.

PREFACE

This thesis concludes my Master of Science degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology. The project has been done in collaboration with Adigo AS.

I would like to thank my supervisors Jan Tommy Gravdahl and Trygve Utstumo for their guidance and help throughout the project. I would also like to thank Adigo for providing the interesting challenge, supplying with robots and making field trips possible.

Jarle Dørum
Trondheim, May 28, 2015

CONTENTS

1 INTRODUCTION 7
- 1.1 Problem Statement 8
- 1.2 Literature Study 8
- 1.3 The Robots 10
- 1.4 Report Outline 10

2 COMPUTER VISION THEORY 11
- 2.1 Digital Images 11
  - 2.1.1 Grayscale Images 11
  - 2.1.2 Color Images 12
  - 2.1.3 Color Spaces 13
- 2.2 Pinhole Camera Model 15
  - 2.2.1 Intrinsic Parameters 18
  - 2.2.2 Extrinsic Parameters 19
  - 2.2.3 Full Pinhole Camera Model 20
- 2.3 Geometric Primitives and Perspective Transformations 20
  - 2.3.1 Homogeneous Coordinates 20
  - 2.3.2 Transformations 22
  - 2.3.3 Euclidean 22
  - 2.3.4 Similarity 23
  - 2.3.5 Affine 23
  - 2.3.6 Projective 23
- 2.4 Basic Image Processing 24
  - 2.4.1 Point Operators 24
  - 2.4.2 Group Operators 25
  - 2.4.3 Filters 25
  - 2.4.4 Morphological Operations 26
- 2.5 Histogram Backprojection 27
- 2.6 Estimating Straight Lines in Images 28
  - 2.6.1 Hough Transform 28
  - 2.6.2 Line Fitting 29

3 CONTROLLER DESIGN 33
- 3.1 Kinematic Model of Robot 33
- 3.2 Line Following Controller 34

3.3 Calculating Angle and Distance to Line 37
3.4 Row Estimation with Kalman Filter 38

4 HARDWARE & SOFTWARE 41
4.1 Robot Operating System 41
4.1.1 Nodes 41
4.1.2 Topics 43
4.2 OpenCV 43
4.3 Motor Controller 44
4.3.1 Motor Controller Driver 45
4.4 Wheel Encoders 46
4.4.1 Tuning 47
4.4.2 Relationship Between Wheel Speeds and Robot Velocities 47
4.5 IMU & GPS 48
4.6 On-board Computer 48
4.7 Camera 49

5 DEVELOPMENT 51
5.1 Simulator 51
5.1.1 Robot Design 52
5.1.2 Simulated Field 53
5.2 Overview of Coordinate Systems 54
5.3 Mapping From Image Coordinates to Robot Coordinates 55
5.3.1 Finding Homography Matrix 57
5.3.2 Mapping from Transformed Picture to Robot Coordinates 58
5.4 Find Row Crops in Images 60
5.4.1 Calculating Reference Histogram 60
5.4.2 Histogram Backprojection 61
5.4.3 Projective Transformation 62
5.5 Estimating Row Crop Pose 62
5.5.1 Combining Hough Lines and Line Fitting 65
5.6 End of Row and Turning Strategy 67
5.6.1 Detecting End of Row 67
5.6.2 Turning Controller 69
5.7 Entire System 69
5.7.1 Implementation in ROS 71

6 SIMULATIONS 73
6.1 Line Following Controller 73
6.2 Identifying and Following a Single Line 75
6.2.1 Estimating Line Pose 75

6.2.2 Following a Single Line with Camera 76
6.3 Identifying and Following Row Crops 77
6.3.1 Method 1 77
6.3.2 Method 2 77
6.4 Row Estimation with Kalman Filter 78
6.4.1 Method 1 79
6.4.2 Method 2 80
6.5 Full System 80

7 LAB TESTS 83
7.1 Line Following Controller 83
7.2 Identifying and Following a Single Line 83
7.2.1 Estimating Line Pose 84
7.2.2 Following a Single Line with Camera 85
7.3 Identifying and Following Row Crops 86
7.3.1 Method 1 86
7.3.2 Method 2 86
7.4 Row Estimation with Kalman Filter 87
7.4.1 Method 1 88
7.4.2 Method 2 88
7.5 Full System 89

8 FIELD TESTS 91
8.1 Image Analysis 91
8.2 Turning At End of Row 92
8.3 Recreating Field in Simulator 93

9 CONCLUDING REMARKS 97

A VIDEOS 99

B PAPER 101

BIBLIOGRAPHY 111

LIST OF FIGURES

Figure 1.1	Illustration of the navigation problem.	8
Figure 1.2	Picture of the Asterix robot.	9
Figure 1.3	Picture of the Dogmatix robot.	9
Figure 2.1	Example of grayscale picture.	12
Figure 2.2	Additive and subtractive color spaces.	14
Figure 2.3	RGB and HSV color spaces.	14
Figure 2.4	Example picture comparing RGB and HSV.	16
Figure 2.5	Histograms of image color channels.	16
Figure 2.6	HSV threshold example.	17
Figure 2.7	Pinhole camera.	17
Figure 2.8	Coordinate systems for the pinhole camera model	18
Figure 2.9	Normalized pinhole camera model.	19
Figure 2.10	Geometric transformations.	22
Figure 2.11	Kernel convolution.	26
Figure 2.12	Example of morphological operations.	27
Figure 2.13	Points on a line in Hough space.	29
Figure 2.14	Example of least squares and RANSAC line fitting.	30
Figure 3.1	Illustrational drawing of Asterix robot.	34
Figure 3.2	Illustration of line following control problem.	35
Figure 3.3	Calculation of distance to line.	38
Figure 4.1	Data flow diagram of robot system.	42
Figure 4.2	Picture of a Roboteq MDC22XX motor controller	45
Figure 4.3	Dogmatix computer and camera.	49
Figure 5.1	Screenshot of Blender.	53
Figure 5.2	Graph diagram generated from URDF file.	53
Figure 5.3	Screenshot of Gazebo simulator.	54
Figure 5.4	Simulated field.	54
Figure 5.5	Overview of coordinate systems.	56
Figure 5.6	Projective transformation illustration	57
Figure 5.7	Projective transformation of a rectangle.	59
Figure 5.8	Estimate homography matrix with chessboards.	59
Figure 5.9	Mapping from image to robot coordinates	60
Figure 5.10	Screenshot of program for histogram creation.	61
Figure 5.11	Backprojection example in simulator.	62
Figure 5.12	Projective transformation in simulator	62

Figure 5.13 Method 1 applied in simulator. 65
Figure 5.14 Method 2 applied in simulator. 67
Figure 5.15 Illustration of a turnaround maneuver. 68
Figure 5.16 Lenghts before turning at end of row. 69
Figure 5.17 State diagram. 70
Figure 5.18 Figure of ROS nodes and topics. 72
Figure 6.1 Simulation of row controller. 74
Figure 6.2 Simulation of row controller. 74
Figure 6.3 Picture from line mapping to global coordinates. 76
Figure 6.4 Simulation, following line with camera. 76
Figure 6.5 Picture from simulated row crops. 77
Figure 6.6 Simulation of method 1. 78
Figure 6.7 Simulation of method 2. 78
Figure 6.8 Simulation of method 1 with Kalman filter. 79
Figure 6.9 Simulation of method 2 with Kalman filter. 80
Figure 6.10 Simulation, full system, method 2 with Kalman filter. . 81
Figure 7.1 Dogmatix, row controller. 84
Figure 7.2 Picture of Dogmatix in line mapping test. 85
Figure 7.3 Dogmatix, following single line with camera. 85
Figure 7.4 Picture of rows used for lab tests. 86
Figure 7.5 Dogmatix, method 1. 87
Figure 7.6 Dogmatix, method 2. 87
Figure 7.7 Dogmatix, method 1 with Kalman filter. 88
Figure 7.8 Dogmatix, method 2 with Kalman filter. 89
Figure 7.9 Full system test with Dogmatix. 90
Figure 7.10 Dogmatix, full system, method 2 with Kalman filter. . . 90
Figure 8.1 Picture of Asterix in field. 91
Figure 8.2 Closeup image of plants in a row. 92
Figure 8.3 Attempted estimation of rows in field. 93
Figure 8.4 Test of turning at end of row with Asterix. 94
Figure 8.5 Picture of recreated field in simulator. 94
Figure 8.6 Row detection in simulated field. 96
Figure 8.7 Simulation of full system with recreated field. 96

1 INTRODUCTION

As the world's population continue to grow there is an ever increasing need to produce more food. By 2042 the world population is estimated to reach 9 billion people [1]. To cover the future needs of food production, more advanced farming techniques are required. Precision agriculture is a concept of estimating variability in crops to allow optimized management for individual parts of the field. E.g. by doing frequent measurements all over the field, the measurements can be used to optimize the amount of fertilizer and herbicide that is being applied to every part of the field to maximize the crop yield.

Applying herbicide to the crops is in most cases necessary to avoid unwanted plants and improve crop yields. The unwanted plants, weeds, can be harmful to the crops. Usually though, weeds only cover a very small fraction of the field. In [2] 12 fields were studied, and as much as 70% of the sample area was found to be free of grass weeds and 30% free of broadleaf weeds. By identifying and precision spraying individual weed leafs the savings in applied herbicides can be massive.

Adigo AS is currently developing an autonomous agricultural robot for detection and precision spraying of individual weed leafs, named Asterix. The robot will follow the tractor wheel tracks in the field and treat the rows with herbicide while navigating autonomously. Previous research on the project includes development of a precision drop-on-demand nozzle for herbicide application [3], a model predictive row controller to minimize potential crop damage during operation[4] and attitude estimation in agricultural robotics [5].

The nozzle array presented in [3] is intended to only be slightly wider than the row crops, meaning that the robot has to follow the row crops precisely. A small offset could mean that the weed is out of reach for the nozzles, leaving the weed untreated. Even worse, a large offset will cause the robot to damage crops by running over them. This motivates the research to develop robust row following methods that can achieve high precision. In this thesis, various methods for detecting and following row crops for a differentially wheeled mobile robot using computer vision have been developed.

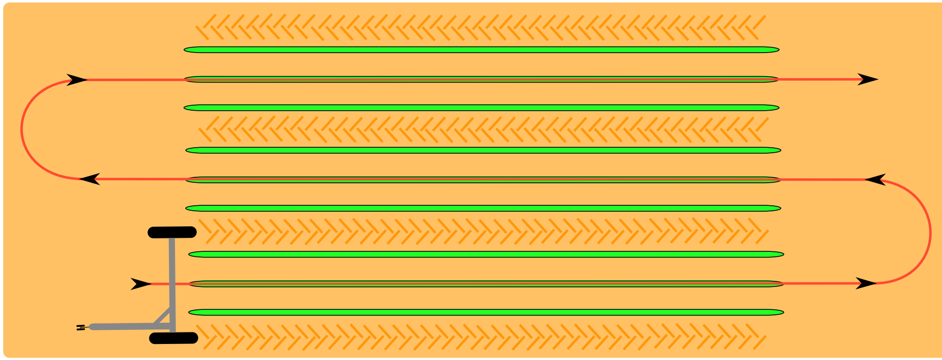


FIGURE 1.1: Illustration of the navigation problem. The robot starts at lower left position, follows the middle row until the end is reached, turns around and starts following the next row until end of field is reached.

1.1 PROBLEM STATEMENT

The field crop that has been considered in this thesis is that used for growing carrots. The field has established tractor wheel tracks that the robot should follow, and in between the wheel tracks there are three rows of crops, as seen in Figure 1.2. The approach used here for navigation is to identify the middle row and attempt to follow it closely. Once the end of a row is reached, the robot should turn around and follow the next row, as illustrated in Figure 1.1

1.2 LITERATURE STUDY

The research field of agricultural robotics is growing rapidly with many conferences taking place every year, so this literature study will only cover a few interesting projects. Detection and mapping of rows is an important part of autonomous navigation and has received a lot of research interest. However, there are so many different kinds of crops that every detection technique needs to be adapted to the specific crop. In addition, the crops will typically look completely different from early to late stages of the growing process. In some cases, the plants will be 20 centimeters tall, while in other cases less than a centimeter.

In [6] methods for detection and mapping of wide row crops are presented. These rows are similar to the rows considered here, but all analysis is done offline on a recorded video and no real-time implementation with autonomous

FIGURE 1.2: Picture of the Asterix robot during field trials on a carrot field in Rygge.

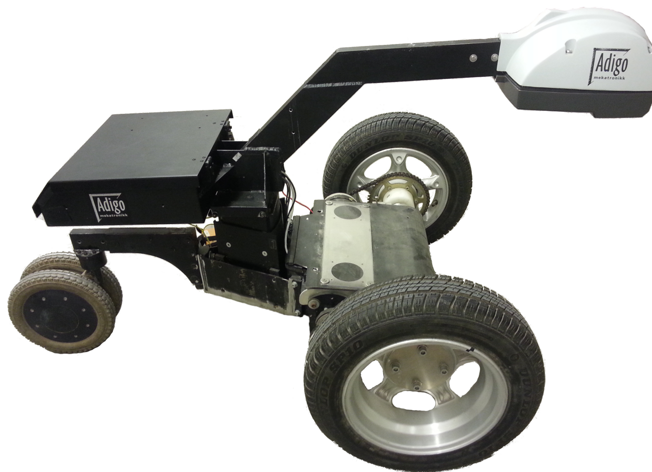


FIGURE 1.3: Picture of the Dogmatix robot, used for all indoor testing. The "head" sticking out in the front contains the computer and camera.

operation is presented. Some of the techniques in this paper are similar to those used here.

A project similar to the Asterix project is a project called BoniRob [7]. It is an autonomous mobile robot platform that can perform lots of different tasks on a field depending on needs. In [7] the navigational system is briefly introduced. The biggest difference from Asterix is that they are using a three dimensional lidar to detect rows instead of a camera, while the Asterix project only relies on computer vision so far. Lidar is a robust technique, but it also means that the plants need to be tall to be detected. For the maize fields used in [7] this is ok, but for a flat field with small plants a lidar alone will probably not be sufficient. Also, lidars are much more expensive than a camera.

1.3 THE ROBOTS

Two different robots have been used for testing. Both robots are differentially wheeled robots with rear caster wheels. A prototype robot in the Asterix project, from here on referred to as just Asterix, is shown in Figure 1.2. The rear caster wheel is mounted off-center to ensure that no plants are damaged during operation. Its width is the same as a tractor to be able to follow the tractor wheel tracks in the field.

The other robot, nicknamed Dogmatix, has been used for indoor testing. A picture of it is included in Figure 1.3. Like its bigger brother it is differentially wheeled, but it has a more conventional design with a rear centered caster wheel.

Both robots are setup so that the same software and code can be used on both robots. This means that software developed on Dogmatix can be directly transferred to Asterix without making any changes, and vice versa.

1.4 REPORT OUTLINE

This thesis is divided into nine chapters and two appendices. Chapter 1 introduces the problem and the robots. Chapter 2 covers the theory behind some computer vision techniques. In Chapter 3 the row controller is derived. Chapter 4 presents some of the hardware and software of the robots. The main parts of the actual development of the methods are documented in Chapter 5. Chapter 6 contains all simulation results, while Chapter 7 presents the same tests performed on Dogmatix. Chapter 8 documents the field trip that was done. A short discussion and concluding remarks are found in Chapter 9. Appendix A contains links to videos of the system in operation. A paper that was written and submitted while writing the thesis is included in Appendix B.

2 COMPUTER VISION THEORY

This chapter covers some basic computer vision theory. The author had no previous experience with computer vision or image processing, so a large part of the thesis was devoted to becoming familiar with basic computer vision.

The chapter opens with a brief introduction to how images and color spaces are represented in computers. Next, there is a fairly comprehensive study of how three-dimensional real world objects are projected onto a two-dimensional image. This is an important basis for finding real world position and orientation of a row based on an image. Next, some basic image processing techniques such as filters and thresholding is introduced. After that follows an important backprojection technique which is used to find plants in images based on colors. Finally, methods for finding and estimating lines in images are covered.

2.1 DIGITAL IMAGES

A digital image is usually represented as a matrix. Each element of the matrix represents a *pixel*, the smallest addressable element of an image. The position of a pixel can be represented in a two-dimensional coordinate system $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T = (x, y)$. In *OpenCV*, a computer vision library which will be covered in more detail later, the origin of the image coordinate system is in the upper left corner as shown in Figure 2.1.

This section will cover the basics of digital images, color spaces and some simple image manipulation methods. First, let us start with simple grayscale images.

2.1.1 *Grayscale Images*

Grayscale images are images that only contain brightness values and are more commonly referred to as black-and-white images [8]. Each element of the matrix (pixel) contains a positive integer that represents the brightness value. Zero

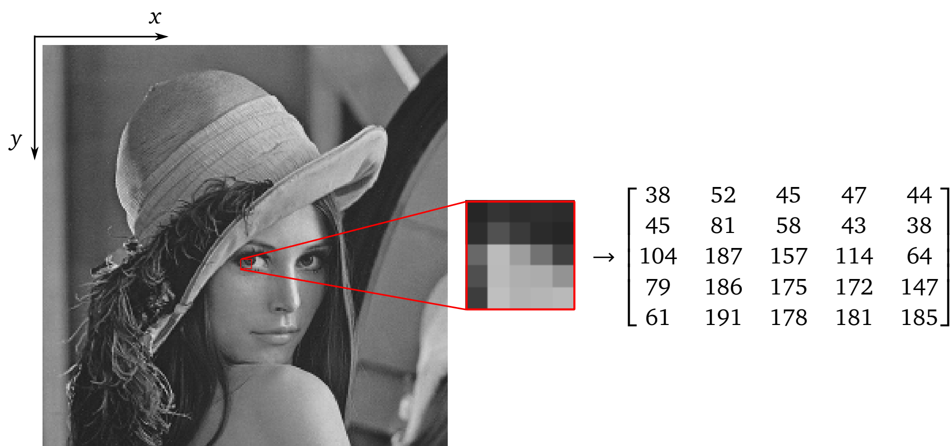


FIGURE 2.1: Example of a grayscale picture and how it is represented in matrix form. This is a 8-bit grayscale picture, i.e. brightness values range from 0 to 255 where 0 is black and 255 is white.

represents black and the maximum value represents white, while all numbers in between are shades of gray. E.g. for a 8-bit grayscale image the range would be from 0 (black) to $2^8 - 1 = 255$ (white). An example 8-bit grayscale picture is shown in Figure 2.1.

2.1.2 *Color Images*

To understand how digital color images are represented, it is useful to briefly cover how the human eye senses light and colors. The human eye has approximately 100 million sensors at the retina. [9, p. 6] The largest portion of these sensors are so called rods, used for black-and-white vision, while the remaining sensors are called cones, used for perceiving colors. There are three kinds of cones that sense different parts of the visible light spectrum [9, p. 6]:

- **S**: Senses short wavelength - blue.
- **M**: Senses medium wavelength - green.
- **L**: Senses long wavelength - red.

The color we perceive is a summation of what the cones sense. The combination of these three colors can cover almost the entire visual spectrum, which is why the *RGB* (Red Green Blue) color space is extensively used in representing digital images [10, p. 46]. RGB is an additive color space, which means that red, green

and blue combined add up to white. Figure 2.2 shows a comparison with the subtractive *CMY* (Cyan Magenta Yellow) color space.

A digital color image with RGB color model can be represented as an extension of the grayscale image where each element of the matrix consists of a triplet instead of a single element. The triplet has one value for each of the colors red, green and blue, often referred to as channels, which combined make up the color for the pixel. E.g. for a 24-bit image each channel has a range from 0 to $2^8 - 1 = 255$, meaning that the triplet $(255, 0, 0)$ is red, $(255, 255, 0)$ is yellow, $(0, 0, 0)$ is black and $(255, 255, 255)$ is white. A 24-bit image can represent up to $2^{24} = 16,777,216$ different color combinations.

2.1.3 *Color Spaces*

The RGB color model is useful for hardware implementation, e.g. a single pixel on a computer monitor often consists of two green, one red and one blue light source. However, RGB is not very descriptive for humans, as we usually do not express colors as mixtures of red, green and blue, and the RGB value will change with different light conditions. From a human point of view it is more natural to describe a color as i.e. "green" and how bright or saturated the color appears. As it turns out, this representation is also very useful for interpretation by a computer. One such color model is known as *HSV*.

The HSV color model describes colors using *hue*, *saturation* and *value*. While RGB can be represented in three-dimensional Cartesian coordinates as a cube, HSV can be represented as a three-dimensional cone, as illustrated in Figure 2.3. The three different channels represent the following [11]:

- **Hue (H)**: A measure of the spectral composition of a color, i.e. the wavelength of the color. It is measured in degrees around the vertical axis, usually between 0 and 180 or 0 and 360 degrees.
- **Saturation (S)**: Indicates how "pure" the color is, i.e. how far from gray of equal brightness it is. Represented as the radial component of the cone.
- **Value (V)**: A measure of the relative brightness. Represented as the height of the cone.

Note that at the point $V = 0$ hue and saturation is undefined, and wherever $S = 0$ hue is undefined. There is also a singularity in the hue channel after completing a whole rotation at 360 and 0 degrees. There are no specified

standards for the ranges of H, S and V, which means that they will vary in different computer programs.

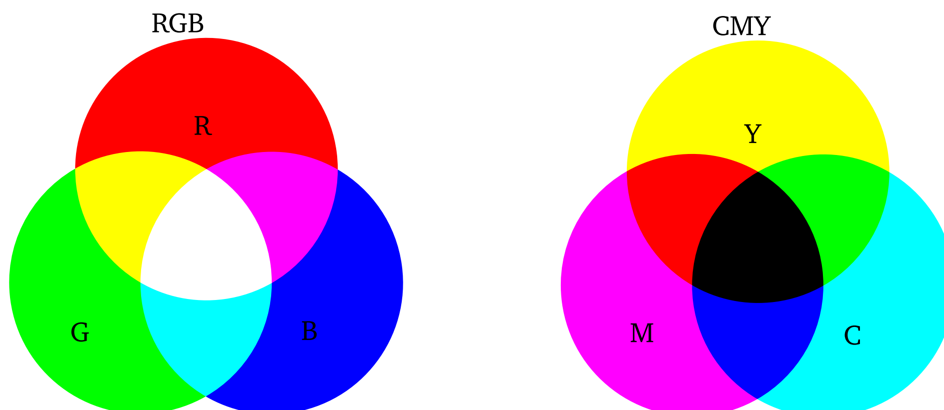
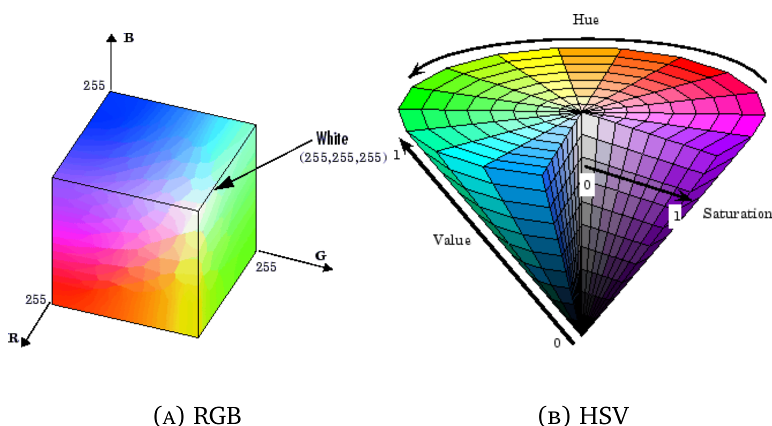


FIGURE 2.2: RGB is an additive color model, so red, green and blue are additive colors making up white when all are added together. CMY is a subtractive color model, so cyan, magenta and yellow specify which colors are removed from white light, adding up to black. RGB is useful for emitting light sources, such as a computer monitor, while CMY is used for reflective light sources, such as printing on white paper.



(A) RGB (B) HSV

FIGURE 2.3: Comparison of RGB and HSV color spaces.

The greatest advantage of using HSV in computer vision will be demonstrated using a simple example. Assume that we want to pick all regions containing grass in Figure 2.4. The picture contains a bright blue sky, clouds and grass under different lighting conditions. Intuitively it might make sense to threshold the areas that have high G values using RGB. Inspecting the brightly lit grass it can be verified that it does indeed have high G values. However, by inspecting parts of the sky it can be seen that it actually contains more green than the grass!

Furthermore, the parts of the grass that lie in the shadow of clouds contain very little green. Clearly, simply thresholding high values of the G channel of RGB is not a viable solution for finding grass regions.

If we instead look at the HSV values we see that the H channel is very similar for all parts of the grass, while especially V varies quite a lot depending on lighting. The sky has a very different H value, so thresholding using H channel appears to be a valid option. Indeed, by calculating histograms of each color channel as shown in Figure 2.5, it can be seen that there are two spikes in the hue histogram: One at around 70 degrees (green) and another around 210 degrees (blue). The other color channels do not appear to have any particular features usable for extracting a region. The result of a thresholding based on hue channel is shown in Figure 2.6

This does not mean that RGB does not have a place in computer vision. A simple technique sometimes used is comparing the difference between the color channels. E.g. in the case of separating grass and sky, it is very possible that the difference between R and G is fairly consistent, i.e. $R - G$ is close to constant. However, for the rest of this thesis the HSV color space has been used for all color images.

2.2 PINHOLE CAMERA MODEL

A pinhole camera is one of the simplest cameras. It consists of a closed chamber with a very small hole (pinhole) in the front with no lens [12]. The idea is that light rays from objects in the world pass through the pinhole and form an upside-down image in the back of the chamber. The distance from the pinhole to the image plane is known as the *focal length*. To describe this mathematically, it's easier to think of a virtual image at a distance equal to the focal length in front of the camera. This is illustrated in Figure 2.7.

The pinhole camera model is an ideal and simplified model of a camera, but in many cases it provides a good approximation. We would like to derive a mapping from world coordinates to image coordinates. The world coordinate system's origin is placed at the *optical center* (pinhole) and is represented as

$$\boldsymbol{w} = \begin{bmatrix} u & v & w \end{bmatrix}^T \tag{2.1}$$

while image coordinates are represented as

$$\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T \tag{2.2}$$

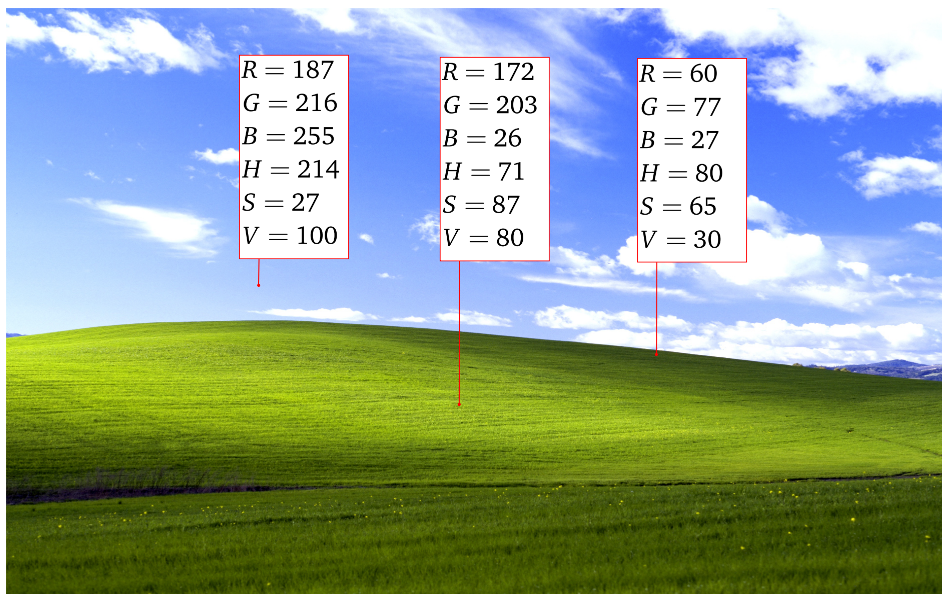


FIGURE 2.4: Example picture with color values represented with RGB and HSV for different areas. The range used are from $(0, 0, 0)$ to $(255, 255, 255)$ for RGB and $(0, 0, 0)$ to $(360, 100, 100)$ for HSV.

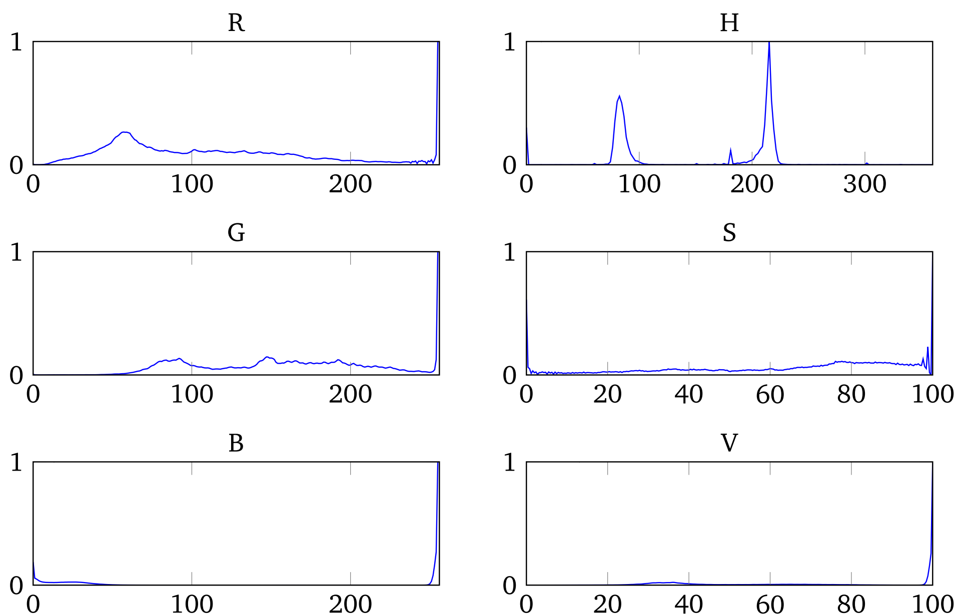


FIGURE 2.5: Normalized histograms of the color channels of the picture in Figure 2.4.

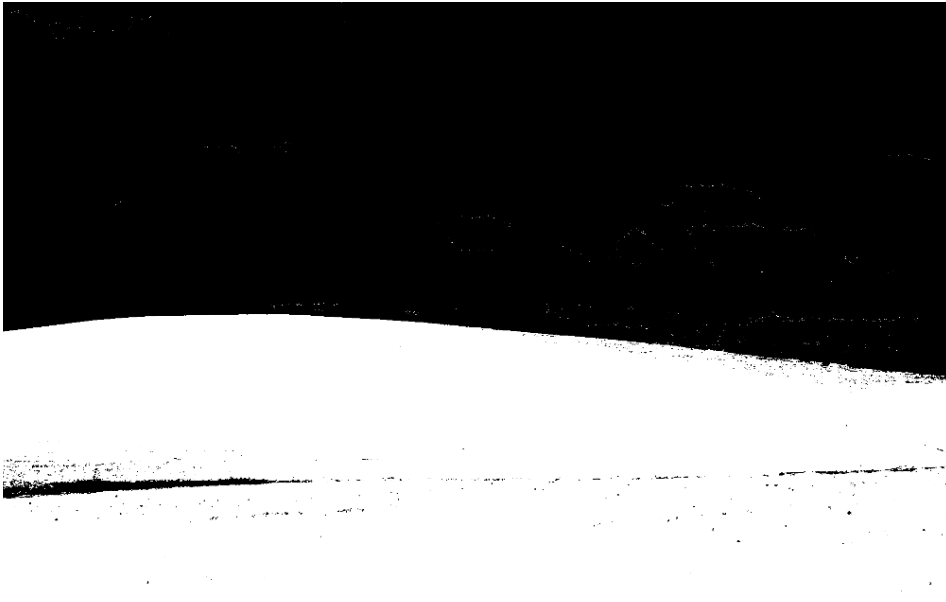


FIGURE 2.6: A mask showing the result of a threshold on the hue channel of the picture in Figure 2.4 from $H = 63$ to $H = 85$. The white area shows the selected region.

The position and orientation of world and image coordinate systems are shown in Figure 2.8. In the next sections we will derive a model for *perspective projection*, i.e. the mapping of 3D points $\boldsymbol{w}$ to image points $\boldsymbol{x}$.

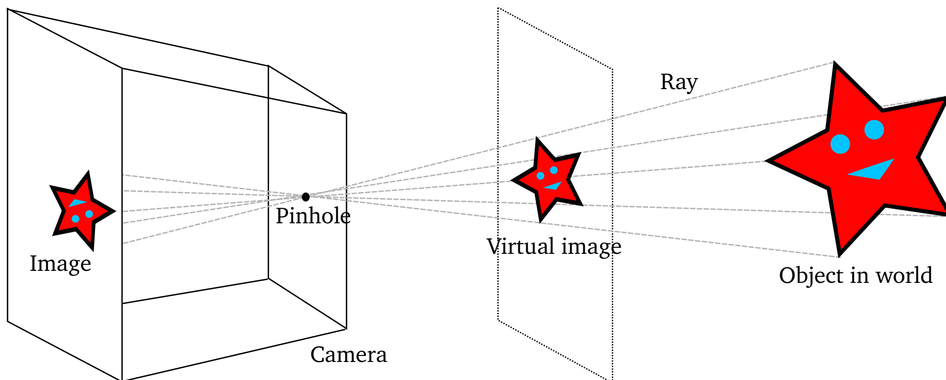


FIGURE 2.7: Illustration of pinhole camera. Rays from the world form an upside-down image on the back of the camera. A virtual image in front of the camera makes the image the right way round to make it easier to think about. (Based on figure 14.2 from *Computer Vision: Models, Learning, and Inference* [12])

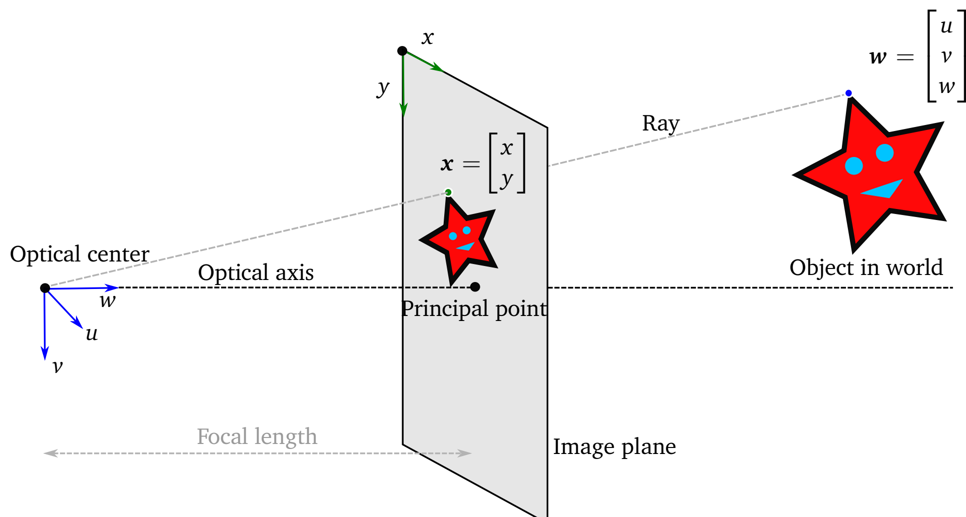


FIGURE 2.8: Illustration of coordinate systems for the pinhole camera model. (Based on figure 14.3 from *Computer Vision: Models, Learning, and Inference* [12])

2.2.1 *Intrinsic Parameters*

Intrinsic parameters or *camera parameters* are parameters that describe the camera itself. These remain unchanged unless the camera configuration changes, e.g. by zooming or replacing the lens. First, let's consider the idealized normalized camera.

The normalized camera is a pinhole camera with focal length equal to one and image coordinate system centered at the principal point [12, p.361]. This is illustrated in Figure 2.9. It can easily been seen by use of similar triangles that the image coordinates can be calculated using

$$x = \frac{u}{w}, \quad y = \frac{v}{w} \tag{2.3}$$

where all lengths are measured in length units, e.g. meters, and not in pixels. This model has some faults. First of all image position is usually given in pixels instead of lengths, and there's no reason why focal length should be exactly one. This can be compensated by multiplying (2.3) with scaling factors ϕ_x and ϕ_y. Secondly, the image point $\boldsymbol{x} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ will rarely be at the principal point, e.g. in OpenCV the origin is in the top left corner. To compensate for this offsets δ_x and δ_y are added to (2.3). Finally, a skew term γ is added to the x position which is multiplied with the real world v value. This term does not

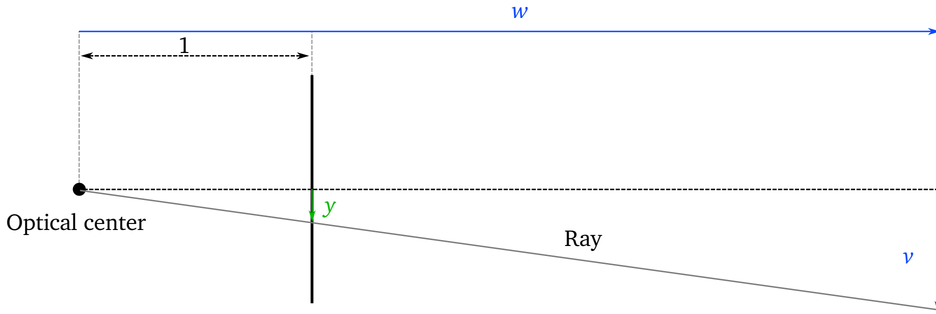


FIGURE 2.9: Normalized pinhole camera model. (Based on figure 14.4 from *Computer Vision: Models, Learning, and Inference* [12])

have a simple physical explanation, but it is useful in practice. The resulting model with the added corrections for intrinsic parameters is [12, p.364]

$$x = \frac{\phi_x u + \gamma v}{w} + \delta_x, \quad y = \frac{\phi_y v}{w} + \delta_y \tag{2.4}$$

2.2.2 *Extrinsic Parameters*

Extrinsic parameters describe the position and orientation of the camera. This is done by converting world coordinates $\boldsymbol{w}$ using a rotation matrix $\boldsymbol{\Omega}$ and a translational vector $\boldsymbol{\tau}$. The transformed point $\boldsymbol{w}'$ is expressed as [12, p.364]

$$\boldsymbol{w}' = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{31} & \omega_{32} & \omega_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \boldsymbol{\Omega}\boldsymbol{w} + \boldsymbol{\tau} \tag{2.5}$$

2.2.3 *Full Pinhole Camera Model*

The full pinhole camera model can be described by combining (2.5) and (2.4). The mapping from a three-dimensional point in world coordinates $\boldsymbol{w} = \begin{bmatrix} u & v & w \end{bmatrix}^T$ to two-dimensional image point $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T$ is [12, p.365]

$$x = \frac{\phi_x(\omega_{11}u + \omega_{12}v + \omega_{13}w + \tau_x) + \gamma(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y)}{\omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z} + \delta_x$$
$$y = \frac{\phi_y(\omega_{21}u + \omega_{22}v + \omega_{23}w + \tau_y)}{\omega_{31}u + \omega_{32}v + \omega_{33}w + \tau_z} + \delta_y \tag{2.6}$$

2.3 GEOMETRIC PRIMITIVES AND PERSPECTIVE TRANSFORMATIONS

The concept of perspective is vital in computer vision. It can be used to give a mathematical description of how a camera projects a three-dimensional scene to a two-dimensional image. It is therefore useful for mapping image coordinates to real world coordinates.

The human brain does a remarkable job at performing projective transformations. We can usually identify lines that are parallel in the real world from an image, even though the lines in the image are not parallel. Similarly, we might interpret an ellipse in an image as a circle in the real world. The brain does this effortlessly, but the same cannot be said for a computer. How can we express this mathematically so that a computer is able to perform perspective transforms on a digital image? What geometric properties change during a perspective transform and what properties are invariant? First, let us introduce some geometric concepts.

2.3.1 *Homogeneous Coordinates*

The Cartesian coordinate system is a natural choice for describing a point or vector in a two-dimensional image. E.g. a point in an image may be described by it's pixel coordinates in Cartesian coordinates as

$$\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T \in \mathscr{R}^2 \tag{2.7}$$

(2.7) may also be represented using homogeneous coordinates [13, p.32]

$$\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{x} & \bar{y} & \bar{w} \end{bmatrix}^T \in \mathscr{P}^2 \tag{2.8}$$

where $\bar{w}$ is an arbitrary scalar, $\bar{x} = wx$, $\bar{y} = wy$ and $\mathscr{P}^2 = \mathscr{R}^3 - \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ which is the two-dimensional projective space. The mapping from homogeneous coordinates back to Cartesian coordinates is simply

$$\boldsymbol{x} = \begin{bmatrix} \bar{x}/\bar{w} & \bar{y}/\bar{w} \end{bmatrix}^T \tag{2.9}$$

Note that any point $\begin{bmatrix} \bar{x} & \bar{y} & \bar{w} \end{bmatrix}^T$ where $\bar{w} \neq 0$ is equivalent to $\begin{bmatrix} \bar{x}/\bar{w} & \bar{y}/\bar{w} & 1 \end{bmatrix}^T$, i.e. all points with $\bar{w} = 1$ lie in the Euclidean space. Points with $\bar{w} = 0$ are called ideal points or points at infinity and cannot be represented with inhomogeneous coordinates[13, p.32].

Lines can also be represented with homogeneous coordinates. First consider a line in Cartesian coordinates

$$ax + by + c = 0 \tag{2.10}$$

Where a, b and c are constants. In homogeneous coordinates (2.10) may be written as

$$ax + by + cz = 0 \;\rightarrow\; \boldsymbol{l} = \begin{bmatrix} a & b & c \end{bmatrix} \tag{2.11}$$

Which means that points and lines have become indistinguishable, they are both represented by triplets. In Cartesian coordinates two lines are parallel if their slopes a/b are equal, but there is no way to find the intersection of two parallel lines. Intuitively it does not make much sense attempting to find intersection between parallel lines, but for projective transformations it becomes very useful. By extending from the Euclidean space $\mathscr{R}^2$ to the projective space $\mathscr{P}^2$ it is possible to algebraically express the intersection of parallel lines in points at infinity. Consider two parallel lines represented with homogeneous coordinates in $\mathscr{P}^2$

$$\begin{aligned} l_0x + l_1y + l_2z = 0 \;&\rightarrow\; \boldsymbol{l}_1 = \begin{bmatrix} l_0 & l_1 & l_2 \end{bmatrix}^T \\ l_0x + l_1y + l_2'z = 0 \;&\rightarrow\; \boldsymbol{l}_2 = \begin{bmatrix} l_0 & l_1 & l_2' \end{bmatrix}^T \end{aligned} \tag{2.12}$$

The intersection of the lines can be found by calculating the cross product

$$\boldsymbol{l}_1 \times \boldsymbol{l}_2 = \begin{bmatrix} l_1l_2' - l_1l_2 & -l_0l_2' + l_2l_0 & 0 \end{bmatrix} \tag{2.13}$$

This means that the only intersection between two parallel lines is at the point where the last element is zero and, as previously mentioned, are called ideal points or points at infinity.

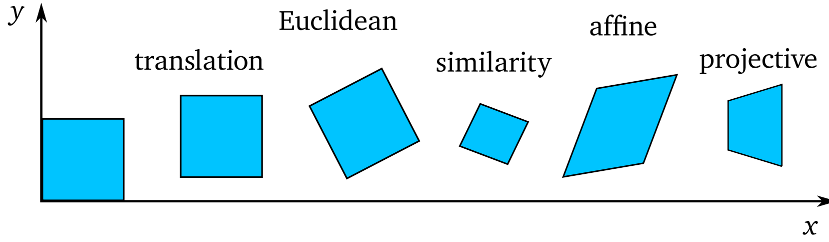


FIGURE 2.10: Geometric transformations. The original rectangle is to the left.

2.3.2 *Transformations*

Following are some two-dimensional transformations as shown in Figure 2.10. Let

$$\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{x} & \bar{y} & \bar{w} \end{bmatrix}^T \tag{2.14}$$

be the homogeneous representation of the Euclidean point $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T$.

Translation

A translation transform is simply a constant translation of every point in x and y direction and can be written as [13, p.36]

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{w} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \bar{x} \tag{2.15}$$

2.3.3 *Euclidean*

Euclidean transformation is a combination of translation and rotation and can be written as [13, p.36]

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{w} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \bar{x} \tag{2.16}$$

Note that all lengths and angles between lines remain the same. Often the Euclidean transformation is referred to as rigid body motion.

2.3.4 *Similarity*

Also known as scaled rotation. It is the same as Euclidean transform, but with an added scaling factor.

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \\ 0 & 0 & 1 \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.17}$$

where a, b and s are arbitrary scalars. Note that we no longer require $s\boldsymbol{R}$ to be orthogonal, i.e. $a^2 + b^2 = 1$ does not need to be satisfied. This means that lengths are no longer preserved, but angles remain the same.

2.3.5 *Affine*

The affine transformation can be written as

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{bmatrix} \bar{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.18}$$

The elements of $\boldsymbol{A}$ and $\boldsymbol{b}$ can be chosen arbitrarily. After an affine transformation angles and lengths are no longer preserved, but parallel lines remain parallel.

2.3.6 *Projective*

The projective transform, also known as homography or projectivity, can be written as

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \bar{\boldsymbol{x}} = \boldsymbol{H}\bar{\boldsymbol{x}} \tag{2.19}$$

Where $\boldsymbol{H}$ is a non-singular 3×3 matrix where only the ratio of the matrix elements matters, i.e. multiplying $\boldsymbol{H}$ with a scalar does not change the projective transformation. Therefore, $\boldsymbol{H}$ is called a homogeneous matrix. A definition of a projectivity can be stated as [14, Definition 2.9 p.33]

Definition 1 *A projectivity is an invertible mapping h from* $\mathscr{P}^2$ *to itself such that three points* $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ *and* $\boldsymbol{x}_3$ *lie on the same line if and only if* $h(\boldsymbol{x}_1)$, $h(\boldsymbol{x}_2)$ *and* $h(\boldsymbol{x}_3)$ *do.*

In other words, straight lines remain straight after the transformation.

2.4 BASIC IMAGE PROCESSING

This section contains some basic image processing techniques that perform an iteration over an image and creates an output image based on an evaluation of either single pixels or groups of pixels. Examples that will be covered are thresholding, filters and morphological operations.

2.4.1 *Point Operators*

Point operators cover the most basic of image processing where individual pixels are considered one at a time without looking at surrounding pixels [9, p.86]. E.g. simply multiplying each pixel value in a grayscale image with a scalar can be used to increase the brightness.

Thresholding

An important point operator is thresholding, which is used to convert an image into a binary image. E.g. if we would like to obtain only parts of a grayscale image with brightness values greater than α, then each pixel value in a new image will be the evaluation of a function

$$\mathrm{dst}(x,y) = \begin{cases} 1 & \text{if } \mathrm{src}(x,y) > \alpha \\ 0 & \text{else} \end{cases} \tag{2.20}$$

where (x,y) is the pixel position, dst is the destination binary image and src is the source image. The new binary image will only consist of two different values, in this case 0 or 1. There are many different thresholding techniques available, but these will not be covered in detail here. Adaptive thresholding techniques are particularly interesting for their ability to adapt the value of α based on neighbor pixels or global properties of the image. Otsu's method is one of the most popular adaptive thresholding techniques [15].

2.4.2 *Group Operators*

Group operators calculate new pixel values not only based on the current pixel, but also the neighboring pixels [9, p.98]. For a linear filter a weighted sum of the current pixel and its surrounding pixels form a pixel value in a new image. The set of weights is often referred to as *kernel* or simply *filter coefficients*. The shape of the kernel can be arbitrary, but it often forms a square. The sum of the weights should be equal to one for normalized filters. For the case of the linear filter the output pixel value $g(x, y)$ can be expressed as [13, p.111]

$$g(x,y) = \sum_{k,l} f(x+k, y+l)h(k,l) \tag{2.21}$$

where f is the input image, h is the kernel and k and l are the number of rows and columns in the kernel. (2.21) may equivalently be expressed as a convolution

$$g(x,y) = \sum_{k,l} f(k,l)h(x-k, y-l) = f * h \tag{2.22}$$

An illustration of this operation is shown in Figure 2.11. Note that the output image will be smaller than the input image, unless e.g. the outer pixels of the input image are simply copied over to the output image.

2.4.3 *Filters*

Filters are used to remove noise, blur images or enhance features. All linear filters can be expressed as a convolution operation (2.22). Not much filtering has been needed in this thesis, but some of the filters that have been used are:

- **Gaussian:** Kernel is weighted Gaussian with mean in the center. Suited for removing noise in images, which is typically Gaussian distributed itself. Can make image appear blurred.
- **Mean:** Also known as box filtering. Every element of the kernel is equally weighted, which means that every new pixel value is simply an average of itself and the surrounding pixels. Useful for smoothing images.
- **Median:** Choose the median of itself and the surrounding pixels. Note that this is not a kernel convolution operation and it is considered a non-linear filter. Can help remove noise while still keeping edges sharp.

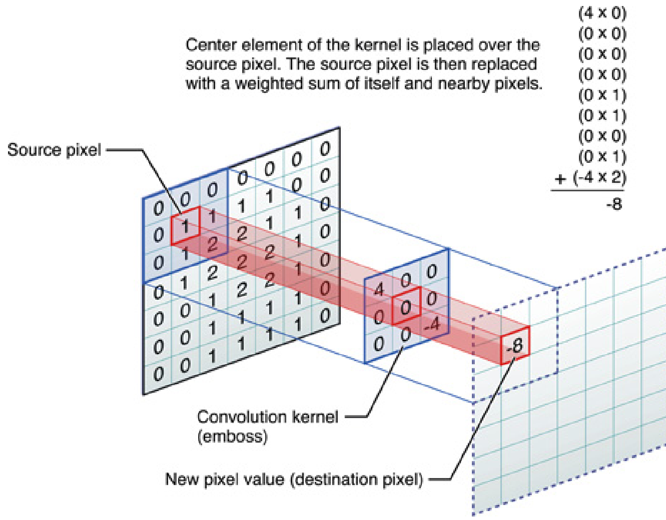


FIGURE 2.11: An illustration of a kernel convolution. Note that the kernel is not normalized in this case. Retrieved on May 8th 2015 from `https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html`

2.4.4 *Morphological Operations*

Morphological operations are very often used on binary images to dilate or erode pixels. Even though it is usually applied for binary images, it can also be used for grayscale images, but binary images are used as examples here. A binary image is iterated over with a structuring element and the output binary image either has increased (dilated) or decreased (eroded) the size of the nonzero areas.

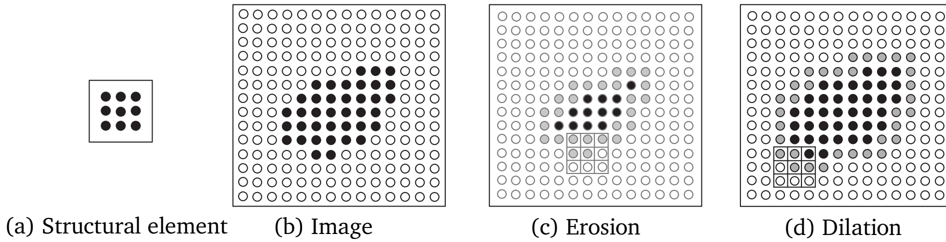
Let the image be defined by pixel values of either 1 or 0. Let $\boldsymbol{X}$ be the set of all nonzero pixels, $\boldsymbol{B}$ the structuring element, x one element of $\boldsymbol{X}$ i.e. one pixel. Then the erosion operation can be expressed mathematically as [9, p.124]

$$\boldsymbol{X} \ominus \boldsymbol{B} = \{x | \boldsymbol{B}_x^1 \subset \boldsymbol{X}\} \tag{2.23}$$

where $\boldsymbol{B}_x^1$ contains the nonzero pixels of $\boldsymbol{B}$. In similar fashion dilation is expressed as

$$\boldsymbol{X} \oplus \boldsymbol{B} = \{x | \boldsymbol{B}_x^2 \subset \boldsymbol{X}^c\} \tag{2.24}$$

where $\boldsymbol{B}_x^2$ contains the zero pixels of $\boldsymbol{B}$ and $\boldsymbol{X}^c$ is the complement of $\boldsymbol{X}$, i.e. it contains all zero pixels.



(a) Structural element (b) Image (c) Erosion (d) Dilation

FIGURE 2.12: Example of morphological operations with a 3 × 3 box element. Black dots represent pixels in the set $\boldsymbol{X}$ and white dots $\boldsymbol{X}^{\boldsymbol{C}}$. Gray dots are either removed or added pixels. (Based on figure 3.33 from "Feature Extraction & Image Processing for Computer Vision, Third Edition" [9])

To illustrate this more clearly, consider Figure 2.12. In this case $\boldsymbol{B}$ is simply a box. Consider the process of erosion. If for some pixel x, $\boldsymbol{B}_{\boldsymbol{x}}$ contains a zero pixel, then the pixel in the output image will be zero. In other words, it is like taking the minimum value of the pixels covered by the structuring element at each pixel and store the value in the output image. Similarly, the dilation process is the same process except for taking the maximum instead of minimum value.

By first applying dilation and then erosion one can achieve a closing operation. This is used a lot in this thesis to merge plants that are close into a smooth row instead of individual plants. Similarly, a opening operation can be achieved by first applying erosion and then dilation, but this has not been used here.

2.5 HISTOGRAM BACKPROJECTION

Histogram backprojection is a technique that makes it possible to compare a histogram of a certain feature/channel of a reference image, e.g. hue color value, to another image and produce a "probability image" highlighting the areas that match well [16]. The technique can be summarized into following steps:

Step 1: Compute histogram of a reference image based on image channels of choice. In this thesis, both hue and saturation have been used together to form a two-dimensional histogram. It is a good idea to normalize the histogram to values between zero and one.

Step 2: Iterate over all pixels of the image we want to compare with. For each pixel, look up the value in the histogram of the reference image and copy this value into a new image, the probability image.

Step 3: After iterating over all of the image, the probability image now consists of pixel values between zero and one representing "probabilities" that they belong to the class defined by the reference image. To finally obtain a binary image one may threshold pixels of high probabilities.

As an example related to agricultural, consider the case of selecting all leafs from an image. If the areas that do not contain leafs have some what different colors than the leafs, then the method of backprojection should be a good method for finding leafs. Examples of this will be shown in later sections.

2.6 ESTIMATING STRAIGHT LINES IN IMAGES

An important step of estimating the direction and position of row crops in an image is line detection and fitting. Two different techniques are presented in this section.

2.6.1 *Hough Transform*

The use of Hough transform in image processing is one of the classic techniques for locating shapes in images [9]. It was first introduced by Hough in 1962 [17], but the first use for detecting lines was in 1972 [18]. It can be used to find both lines, circles and ellipses among other shapes, but only lines will be covered here. The typical line equation

$$y = mx + c \tag{2.25}$$

can be parametrized as

$$r = x\cos\theta + y\sin\theta \tag{2.26}$$

where r is the shortest distance from the origin to the line, and θ is the angle of the line's normal vector. If θ is restricted to $[0, \pi]$, then the parameters r and θ are unique for a line [18]. This means that points that lie on the same line should intersect at one unique point in the Hough space. This is illustrated in Figure 2.13 and can be used to identify lines with the following steps:

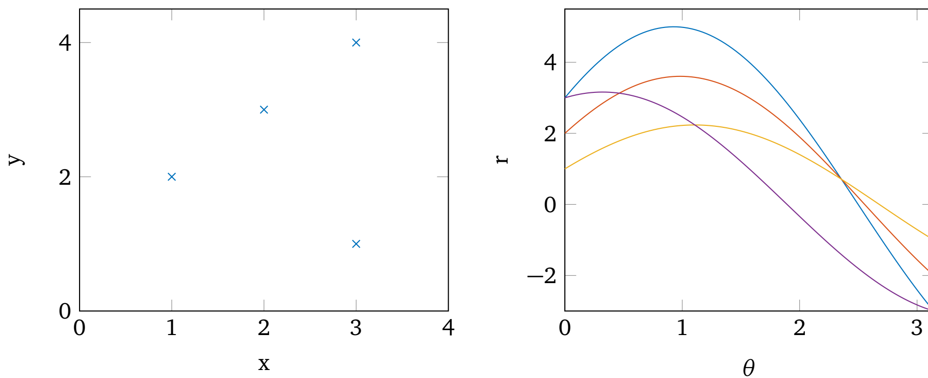


FIGURE 2.13: To the left is a plot of four points in Cartesian space. In Hough space the four points become sinusoidals. Three of the points lie on the same line, so they intersect at the same point in Hough space, while the fourth point is not part of the same line and intersects at different points in Hough space.

Step 1: Divide the Hough space into discrete cells where each cell corresponds to an interval for r and θ. The cells act as accumulators, basically forming a two-dimensional histogram.

Step 2: Locate all edge points in an image. For each edge point (x_i, y_i), calculate r for all values of θ and increment each matching cell by one.

Step 3: After iterating over all edge points, the lines in the image should correspond to the cells with highest values, i.e. lines can be found by searching the cells for local maximums.

In practical implementations there are lots of optimizations that can be done, e.g. reducing the number of values of θ that have to be iterated over.

2.6.2 *Line Fitting*

Line fitting can be formulated as an attempt to find an approximate solution of an overdetermined system, i.e. there are more equations than unknowns. Basically, if one has located several points in an image that are believed to be part of a line, find a line that best fits the points. It is important to understand the difference between this approach and the Hough transform. The Hough transform finds all lines in an image, but with line fitting all the points, or at least most of the points, are used to estimate just one line.

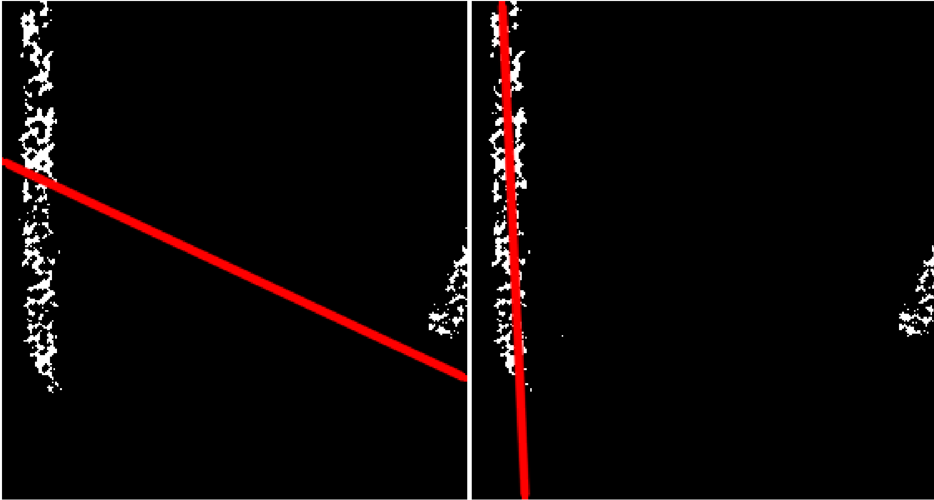


FIGURE 2.14: The white areas show parts of an image that have been selected after a thresholding. The desired behavior is to fit a line to the white areas to the left, but there are outliers to the right of the image. The left image shows a line fitting using standard least squares method, while in the right image a RANSAC-like method is used.

A standard approach to line fitting is the use of a linear least squares method, which should be known from all kinds of research fields. Given n data points (x_i, y_i), $i = 1, \ldots, n$, let $r_i = y_i - f(x_i, \boldsymbol{\beta})$ where $f(x_i, \boldsymbol{\beta})$ is the model function and $\boldsymbol{\beta}$ contains the line parameters to be found. The optimal values for $\boldsymbol{\beta}$ are found by minimizing the sum of square residuals

$$S = \sum_{i=1}^{n} r_i^2 \tag{2.27}$$

This method works well in cases where noise is expected to follow a Gaussian distribution, i.e. data points are expected to deviate evenly on both sides of the actual line. In image processing this is not always the case. Very often there will be outliers that do not belong to the line at all. One of the more robust solutions to fit lines to data sets with outliers is a random sample consensus paradigm, also known as RANSAC [19]. OpenCV has a RANSAC-like implementation with the function *fitLine*. The method can be summarized with the following steps:

Step 1: Pick n randomly selected points.

Step 2: Fit a line to the n selected points using e.g. (2.27).

Step 3: Test if the remaining points fit the line found in previous step using a cost function, which also could be (2.27), but many more cost functions are

available. If the output of this cost function (residual) is lower than some defined threshold, then the point is considered an inliner. Otherwise, the point is considered an outlier.

Step 4: Refit the line to the n randomly selected points plus all the points that were found to be inliners.

Step 5: Repeat steps 3-4 until convergence, i.e. no more inliners are found, or a maximum number of iterations is reached.

Step 6: Check if the line is better than the currently best line by comparing sum of residuals for the inliners. Save it as currently best if better.

Step 7: Repeat steps 1-6 fixed number of times, or until the sum of residuals for the inliners is smaller than some chosen value.

In Figure 2.14 an example binary image is shown with two different line fitting methods.

3 CONTROLLER DESIGN

The control problem of path-following for wheeled mobile robots has been an extensive field of research due to its many use cases. In our case the problem can be simplified to path-following of a straight line for a unicycle-like robot.

The author has previously derived and tested several adaptive controllers for a trajectory tracking problem based on dynamic models of unicycle-like robots (see Appendix B for a detailed paper). While these methods could be easily adapted to this problem, there is simply not enough excitation in the reference signal when following a straight line at constant speed to ensure correct parameter convergence. As it turns out the kinematic model alone is sufficient to develop a controller that meets our specifications. The specifications can be summarized to:

- Follow a straight line with constant forward speed.
- Ability to restrict the line approaching angle θ to ensure that the rear caster wheel does not run over crops.

The last point is important and has been previously researched and simulated as part of this project in [4]. In that paper the problem is tackled using a nonlinear model predictive controller to ensure the rear caster wheel stays in its wheel track. The controller presented here has a slightly simpler approach to the problem and will simply attempt to restrict maximum angle difference to the line/row.

3.1 KINEMATIC MODEL OF ROBOT

The kinematic unicycle model is used to model the kinematics, and is given as

$$\begin{aligned} \dot{x} &= u \cos \psi \\ \dot{y} &= u \sin \psi \\ \dot{\psi} &= \omega \end{aligned} \tag{3.1}$$

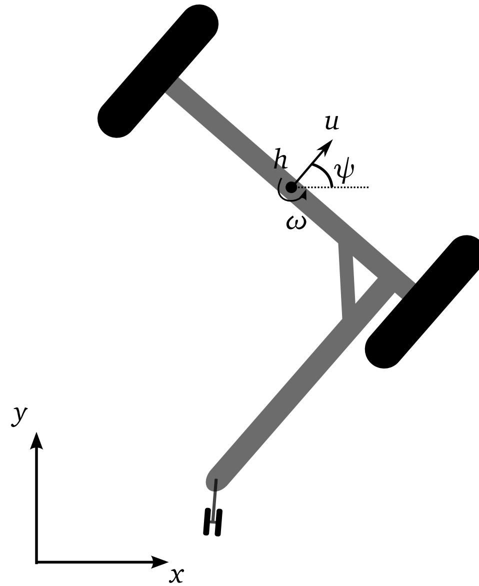


FIGURE 3.1: Illustration of a robot with off center rear mounted caster wheel, similar to Asterix.

where u is forward velocity, ω is the angular velocity and ψ is the heading. The tracking point $h = \begin{bmatrix} x & y \end{bmatrix}^T$ is the point that should follow the line and is located at the center of the wheel axle at the same point as the rotational axis. This is illustrated in Figure 3.1.

For the line-following controller it is more useful to express the position and orientation of the robot relative to the line. In this case it is sufficient to use the distance to the line d and the angle of the line relative to the heading of the robot θ. This is illustrated in Figure 3.2 and may be expressed as

$$\begin{aligned} \dot{d} &= u \sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{3.2}$$

where the sign of d and θ are as defined in Figure 3.2. The goal is to obtain a controller that ensures asymptotic stability at the origin $(d, \theta) = (0, 0)$.

3.2 LINE FOLLOWING CONTROLLER

Since we only consider the kinematic model it is assumed that the dynamics are fast enough to approximate $\omega \approx \omega_r$ where ω_r is the reference signal provided

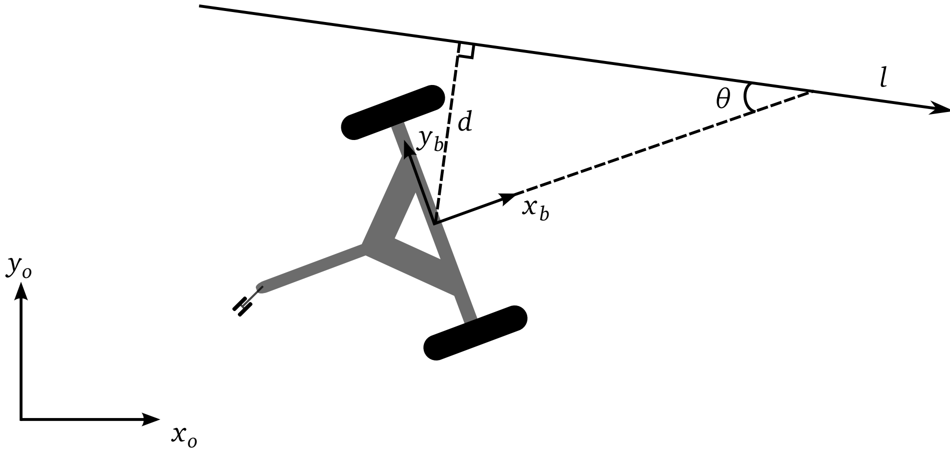


FIGURE 3.2: Illustration of a line l defined in global coordinates that the robot is set to track. An orthogonal projection from the point $(x_b, y_b) = (0,0)$ onto l gives the distance d, while the angle θ is the angle difference of the robot and the line. The controller attempts to achieve $(d, \theta) \rightarrow (0,0)$. In this figure $d < 0$ and $\theta > 0$.

by the control law. In [20] the following control law is shown to provide global asymptotic convergence in the case of a straight line

$$\omega_r = -k_1 u d \frac{\sin\theta}{\theta} - k_2 |u| \theta \tag{3.3}$$

where $k_1, k_2 > 0$ are constant tuning parameters. For all practical purposes $u > 0$ and this will be implicitly assumed for the rest of this thesis. The stability can be analyzed using the following Lyapunov-like function

$$V = \frac{1}{2} k_1 d^2 + \frac{1}{2} \theta^2 \tag{3.4}$$

Differentiating (3.4) along the solution of (3.2) gives

$$\dot{V} = k_1 d u \sin\theta + \theta(-k_1 u d \frac{\sin\theta}{\theta} - k_2 |u| \theta) = -k_2 |u| \theta^2 \leq 0 \tag{3.5}$$

Which means $\dot{V}$ is negative semidefinite. Further analysis is needed to prove asymptotic stability, but there are a few problems related to this control law which might want to make us look into different control laws. Firstly, even though the limit of the function

$$\lim_{\theta \rightarrow 0} \frac{\sin\theta}{\theta} = 1 \tag{3.6}$$

is mathematically smooth and continuous, it can be problematic to implement due to the fact that both numerator and denominator are close to zero. Secondly, it is difficult to restrict the approaching angle (as in the specifications)

when the distance to the line d is large. For these reasons, a different controller is considered. Consider the following controller based on a controller represented in [20]

$$\omega_r = k_1(-uk_2S(d) - \dot{d}) \tag{3.7}$$

where $k_1, k_2 > 0$ and $S(d)$ is a sigmoid function, i.e. its function value is limited to $(-1, 1)\ \forall\ d \in \mathscr{R}$. Inserting $\dot{d}$ from (3.2) into (3.7) gives

$$\omega_r = k_1u(-k_2S(d) - \sin\theta) \tag{3.8}$$

The reason for the choice of a sigmoid function is to add a saturation on d. In practice this means that the approaching angle can be limited. For a large value of d one may approximate $S(d) \approx 1$. Inserting $S(d) = 1$ into (3.8) while letting it equal zero to find equilibrium points we obtain

$$k_2 - \sin\theta = 0 \ \Rightarrow \ \theta = \sin^{-1} k_2 \tag{3.9}$$

This means that the approaching angle can be restricted by choosing appropriate values for k_2. The sigmoid function was chosen to be the hyperbolic function

$$S(d) = \tanh(k_3d) \tag{3.10}$$

with $k_3 > 0$. Inserting (3.10) into (3.8) gives the complete control law

$$\boxed{\omega_r = k_1u(-k_2\tanh(k_3d) - \sin\theta)} \tag{3.11}$$

To prove stability for this consider the following Lyapunov-like function

$$V = \frac{k_1k_2}{k_3}\log(\cosh(k_3d)) + 1 - \cos\theta \tag{3.12}$$

where log is the natural logarithm. Note that V is radially unbounded for d but not for θ. However, for the use case of this controller it is sufficient to restrict the initial conditions of θ to $-\pi/2 < \theta_0 < \pi/2$. Differentiating (3.12) along the solution of (3.2) and (3.11) gives

$$\begin{aligned} \dot{V} &= k_1k_2\tanh(k_3d)\dot{d} + \dot{\theta}\sin\theta \\ &= k_1k_2\tanh(k_3d)u\sin\theta + k_1u(-k_2\tanh(k_3d) - \sin\theta)\sin\theta \\ &= -k_1u\sin^2\theta \le 0 \end{aligned} \tag{3.13}$$

Which means that $\dot{V}$ is negative semidefinite. To prove local asymptotic stability we need to ensure that $d = 0$ and $\theta = 0$ is the only equilibrium point in a closed

region around the origin. It can be seen that $\dot{V} = 0$ can occur if $\theta = 0$ and $d \neq 0$, so we need to ensure that the only point where $\dot{V}$ remains at zero is at the origin.

Stability can be showed using LaSalle's theorem [21]. Let $\Omega = \{\boldsymbol{x} : V(\boldsymbol{x}) \leq V_0\}$ define a closed region around the origin and denote $\boldsymbol{x} = (d, \theta)$ and $V_0 = V(\boldsymbol{x_0})$. All states with initial conditions $\boldsymbol{x_0} \in \Omega$ remain in Ω because $\dot{V} \leq 0$. Let $R \subset \Omega$ be a subset containing the regions where $\dot{V} = 0$, which for this case is all points where $\theta = 0$. From (3.11) and (3.2) and by inserting $\theta = 0$ we have that

$$\dot{\theta} = -k_1 k_2 u \tanh(k_3 d) \tag{3.14}$$

which means that a point in R will not remain in R unless $d = 0$. Let $M \subset R$ contain the maximum invariant set of R, then M will only contain the origin. All the criteria of LaSalle's theorem are satisfied and we can conclude that the system is locally asymptotically stable at the origin.

3.3 CALCULATING ANGLE AND DISTANCE TO LINE

The distance from the robot's wheel center h to the line d and the angle difference of the robot's heading and the line θ has to be calculated. d can be found by performing an orthogonal projection from the point $(x_b, y_b) = 0$ onto the line. Consider the three points and two vectors in Figure 3.3. Let $\boldsymbol{v}$ be a vector perpendicular to the line and $\boldsymbol{r}$ a vector from the point (x_0, y_0) to the line such that

$$\boldsymbol{v} = \begin{bmatrix} y_2 - y_1 & -(x_2 - x_1) \end{bmatrix}^T, \quad \boldsymbol{r} = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 \end{bmatrix}^T \tag{3.15}$$

then the distance d from (x_0, y_0) to the line is given as

$$d = -\hat{\boldsymbol{v}} \cdot \boldsymbol{r} = -\frac{(x_2 - x_1)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \tag{3.16}$$

where $\hat{\boldsymbol{v}}$ is normalized $\boldsymbol{v}$. The choice of minus sign in (3.16) is there to conform with the same sign convention as in Figure 3.2. The angle difference between the robot and the line is found by transforming the line coordinates from global coordinates to robot coordinates. In that case θ is trivial to calculate using the unit line direction vector $\boldsymbol{l^b}$ in robot coordinates and can be determined directly using

$$\boldsymbol{l^b} = \begin{bmatrix} l_1^b & l_2^b \end{bmatrix}^T = \begin{bmatrix} \cos\theta & \sin\theta \end{bmatrix}^T \tag{3.17}$$

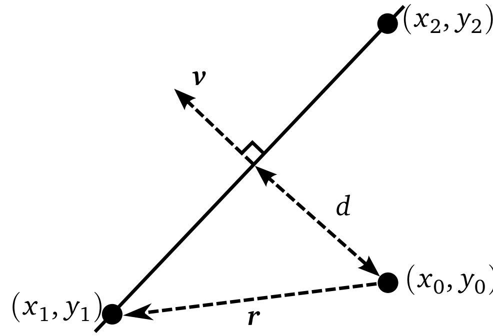


FIGURE 3.3: Illustration of how three points or two vectors can be used to calculate the shortest distance from a point to a line. Based on figure from `http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html` on May 14th 2015.

3.4 ROW ESTIMATION WITH KALMAN FILTER

A "measurement" of a row pose in the form of a line defined in global coordinates is typically obtained several times per second based on processing of camera images. Each measurement will vary slightly from the previous one, i.e. the measurements are noisy. If these measurements are used directly to calculate the states d and θ there will be relatively large discrete jumps in the states which will lead to less smooth operation. For this reason, a filter was implemented to obtain smoother estimates of the line.

Most uncorrelated noise can be modeled as Gaussian noise, i.e. the measurements follow a Gaussian distribution. The most obvious choice to smooth measurements with Gaussian noise is a Kalman filter. In ROS the preferred way to represent a line is by using one point on the line and a direction represented by a unit quaternion. Quaternions are not as intuitive as other representations, but it avoids some common problems in rotation and attitude representation like gimbal locks and non-singularities. The use of quaternions in Kalman filters has been a big research field due to its many uses in e.g. attitude determination. A survey of some of the current nonlinear filtering methods for attitude estimation is provided in [22]. However, a full literature search in this field is outside the scope of this thesis. The approach presented here is merely meant as a simple solution to smooth noisy measurements without much investigation in the wide field of quaternions and their use in filters.

Let a line be represented by a point $\boldsymbol{p}$ and a unit quaternion $\boldsymbol{q}$

$$\boldsymbol{p} = \begin{bmatrix} x_p & y_p & z_p \end{bmatrix}^T, \qquad \boldsymbol{q} = \cos\frac{\alpha}{2} + (u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k})\sin\frac{\alpha}{2} \tag{3.18}$$

where $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ represent unit vectors in Cartesian coordinates, and α represents a rotation around the axis defined by $(u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k})$. For this control problem (3.18) can be simplified a lot. The robot moves on a flat plane, so z can simply be set to $z = 0$. The only rotation considered is that around the z-axis, which means that we can further simplify $u_x = u_y = 0$ and $u_z = 1$. (3.18) is simplified to

$$\boldsymbol{p} = \begin{bmatrix} x_p & y_p \end{bmatrix}^T, \qquad \boldsymbol{q} = \cos\frac{\alpha}{2} + \sin\frac{\alpha}{2}\boldsymbol{k} \tag{3.19}$$

In other words, only three parameters are needed to represent a line in global coordinates: x_p, y_p and α. One could attempt to use only these three states for a Kalman filter, but using α directly can be problematic because after a full rotation there may be a jump from 2π to zero. A possible way to solve this would be to try to keep track of the number of full rotations and let $|\alpha|$ grow past 2π. Another possible solution is to augment the state vector $\boldsymbol{x}$ with a fourth state to form the following state vector

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} x_p & y_p & \sin\frac{\alpha}{2} & \cos\frac{\alpha}{2} \end{bmatrix}^T \tag{3.20}$$

The assumption is that the line does not change, so that the model used for the Kalman filter is simply $\dot{\boldsymbol{x}} = 0$. The following Kalman filter equations were used:

$$\begin{aligned}
\hat{\boldsymbol{x}}_{k|k-1} &= \hat{\boldsymbol{x}}_{k-1|k-1} \\
\boldsymbol{P}_{k|k-1} &= \boldsymbol{P}_{k-1|k-1} + \boldsymbol{Q}_k \\
\tilde{\boldsymbol{y}}_k &= \boldsymbol{z}_k - \hat{\boldsymbol{x}}_{k|k-1} \\
\boldsymbol{S}_k &= \boldsymbol{P}_{k|k-1} + \boldsymbol{R}_k \\
\boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{S}_k^{-1} \\
\hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k \\
\boldsymbol{P}_{k|k} &= (\boldsymbol{I} - \boldsymbol{K}_k)\boldsymbol{P}_{k|k-1}
\end{aligned} \tag{3.21}$$

which were obtained from the usual Kalman filter equations with transition and observation matrices equal to identity matrices. Note that x_3 and x_4 are strongly correlated which may have an undesirable effect on the covariance matrix. One approach to avoid this is to use the fact that the length of the unit quaternion is one, so that e.g. x_4 can easily be calculated from x_3. However, with that approach singularity problems arise once more, as discussed in [23].

Another important thing to note is that since x_3 and x_4 are considered separate states, the length of the quaternion $\sqrt{x_3^2 + x_4^2}$ will not necessarily be equal to one. This means that x_3 and x_4 need to be normalized before reconstructing the

quaternion using (3.19). Further research and analysis should be done to verify or improve upon these results, and is left for future work. However, simulations and tests on real robots show promising results for this simple approach.

4 HARDWARE & SOFTWARE

This chapter covers some of the most important aspects of the hardware setup and software design that went into this project. These elements contributed to a substantial part of the workload, so a chapter is devoted to document the work that was done and to hopefully benefit future students. Parts of this chapter documents preliminary work related to getting the Dogmatix robot up and running, such as setting up motor controller and wheel encoders. This work was documented last semester, but it has been included here for completeness. The chapter covers the setup of the Dogmatix robot, but the Asterix setup is very similar.

4.1 ROBOT OPERATING SYSTEM

Robot Operating System (ROS) is a framework for writing robot software. It is a collection of tools and libraries that aim to simplify robot software development. ROS is open source and in rapid development. It has a very active community that constantly contributes with new packages that help to further simplify development. Tools are available to a wide range of programming languages, with C++ and Python being the most actively developed ones. In this section, some of the core aspects of ROS are introduced. Figure 4.1 shows a simple example of how a system in ROS is divided into nodes and communicate using topics, which might help with understanding while reading this section.

4.1.1 *Nodes*

A node is simply put a process, or similar to the concept of threads in many programming languages. Nodes can easily communicate with each other using topics, which will be explained later. Usually, several nodes work together to form the entire robot system. As an example, some of the nodes running on the Dogmatix robot during typical operation are:

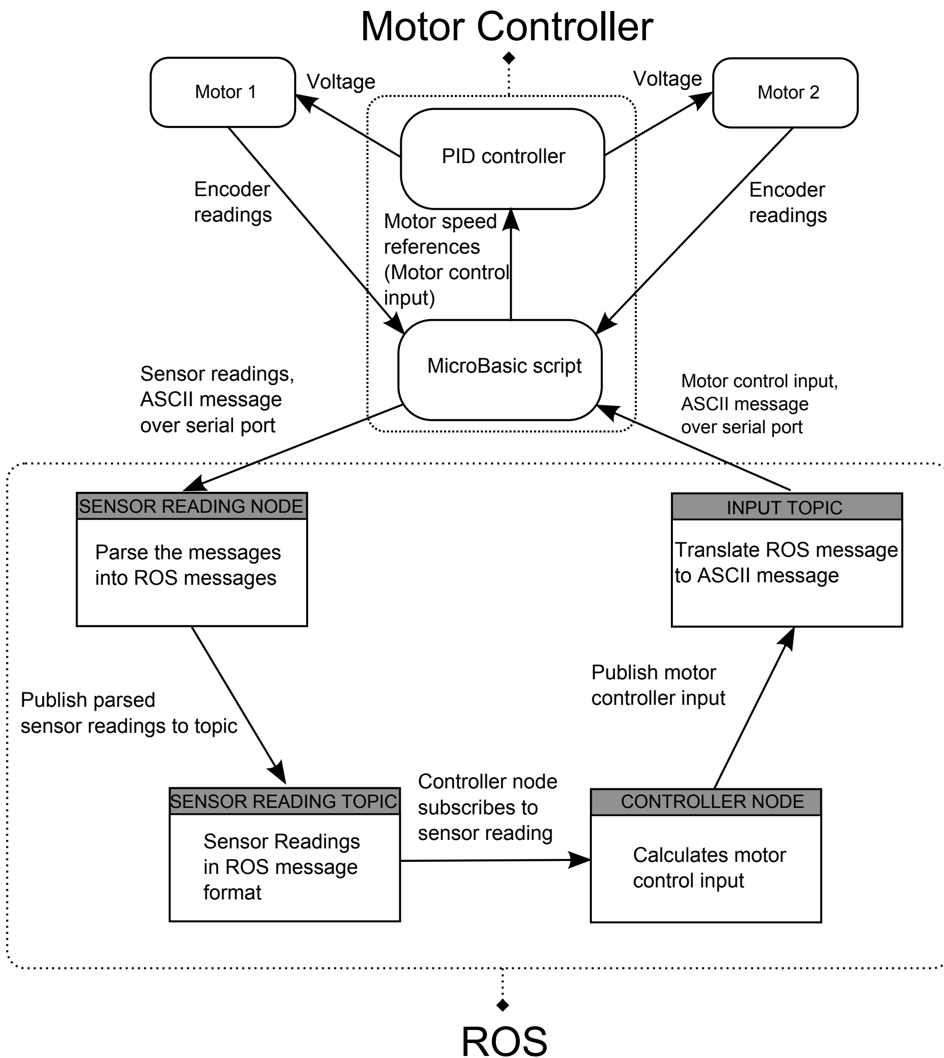


FIGURE 4.1: Data flow diagram showing data flow between motors, motor controller and some of the most relevant ROS nodes and topics.

- **Motor controller driver read node:** Read and parse all incoming messages arriving from the motor controller on the serial port.
- **Wheel encoder node:** Calculate wheel encoder readings into robot velocities and integrate to update the position.
- **Vision node:** Image processing and row estimation.
- **Controller node:** Calculate the speed references to send to the motor controller.

There are more nodes running, but it should be clear that nodes make for a very modular design that is simple to program, while also simplifying typical challenges with real time programming, like communication between threads.

4.1.2 *Topics*

Topics are used for exchanging messages (a message is a simple data structure) between nodes. Nodes can *subscribe* or *publish* to a topic. Consider the following example:

- **Vision node:** Receives images, estimates a row pose in global *odom* coordinates and publishes it to topic */row/pose*.
- **Kalman filter node:** Subscribes to topic */row/pose*, incorporates incoming messages as measurements in the Kalman filter and then publishes the estimates from the Kalman filter to topic */row/kalman/pose*.
- **Controller node:** Subscribes to topic */row/kalman/pose* to receive estimate of row pose, which is used to calculate distance d and θ. Finally, calculate desired speed references u_r and ω_r which are published to topic */controller_inputs*
- **Controller inputs node:** Subscribes to topic */controller_inputs*, recalculates from forward and angular speed into individual wheel speeds and sends the command to the motor controller over serial port.

Working with topics is generally much more pleasant than e.g. handling communication between threads in pure C++.

4.2 OPENCV

OpenCV[1], or *Open Source Computer Vision*, is a computer vision library written in C/C++. It has functions for almost every typical image processing technique, and the functions are optimized for real time applications. There are interfaces available for C, C++, Python and Java, but only C++ have been used in this project.

[1] http://opencv.org/

Other notable computer vision libraries include *Matlab* and *scikit-image*[2] for Python. Matlab is a very high-level language and it is easy to quickly make prototypes. However, it is not well suited for real-time implementation, is costly and has closed source which does not allow for inspection of functions. Scikit-image is a relatively new collection of tools for Python, but it is being rapidly developed. Some of the newest methods in image processing are often developed in Python and will quickly find their way to scikit-image. It is also possible to combine OpenCV and scikit-image since both have Python libraries.

In the end OpenCV and C++ were chosen for this project due to high performance, mature software and plenty of documentation and books available with example code. In addition, OpenCV includes libraries that make it very easy to utilize a Graphics Processing Unit (GPU) for higher performance, which will be done at a later stage in the Asterix project.

4.3 MOTOR CONTROLLER

The robot lacked a working motor controller, so a *Roboteq MDC2230*[3] was installed. It has some features that proved beneficial throughout the project:

- Communication over USB or the serial port with RS232 protocol.
- Two encoder inputs for determining individual wheel speeds.
- Built-in speed controller which allows to input a speed reference to each motor individually. This is done using a PID controller where all three gains can be adjusted.
- Scripting possibilities through a simple scripting language called MicroBasic.

Communication over USB was found to be less reliable (this is also stated in the datasheet), so communication through serial port was used instead.

[2] http://scikit-image.org/

[3] Datasheet is available at `http://www.roboteq.com/index.php/docman/motor-controllers-documents-and-files/documentation/datasheets/mdc2xxx-datasheet-1/2-datasheet-mdc2xxx/file`

Figure 4.2: Picture of a Roboteq MDC22XX motor controller

4.3.1 *Motor Controller Driver*

A driver provides the software interface between the motor controller and the on-board computer. It should send sensor readings from the motor controller to the computer at a regular time interval, and allow for motor inputs to be sent from the computer to the motor controller.

Roboteq provides a simple API for communication between computer and motor controller. However, this provides only a very low-level interface and it does not provide any means of interfacing with ROS. An extension of the API to a full blown driver was needed. Luckily, an implementation of this was already made available [24]. Some modifications were done to the driver code, but the main structure of the program remains. Almost no documentation was provided with the code, so a small explanation is provided below for reference. Figure 4.1 shows a data flow diagram which might provide for a clearer understanding.

- The Roboteq motor controller can be programmed using a simple BASIC-like programming language called MicroBasic. A script is downloaded to the motor controller which includes an infinite loop. This loop sends sensor readings several times per second as an ASCII code using RS232 over the serial port. Sensor readings include encoder readings, motor current, battery supply voltage etc.

- The Roboteq ROS driver includes a node that is constantly listening on the serial port. Whenever a message is received, it parses the ASCII message and stores it in a custom made ROS message. The parsed message is published to an ROS topic.

- Another node subscribes to a different topic listening for motor control inputs. Whenever it receives a message from this topic, it will send a corresponding signal to the motor controller on the serial port.

- Other nodes, such as the the various controller nodes calculating the motor control inputs, can now easily obtain sensor readings and publish it to the motor control input topic.

4.4 WHEEL ENCODERS

Wheel encoders provide rotational velocity readings from each motor. Encoders called *HEDL-5500 G12*[4] were already installed on each motor. Unfortunately, these encoders use a 10-pin connector which is incompatible with the single 6-pin encoder input of the Roboteq controller. The reason for this is that the encoders support sending balanced signals used for removing common-mode interference, but the motor controller does not support balanced signals. By looking at the datasheets, an adapter was made to be able to connect the encoders to the motor controller. The wiring from pin number to pin number is shown in Table 4.1.

TABLE 4.1: Wiring from encoders to motor controller. It seems that encoder and motor controller datasheets has different definitions of A and B, so A was connected to B and vice versa.

Motor	**Encoder Pins**	**Motor Controller Pins**
1	pin 6 (A)	pin 3 (Enc1B)
1	pin 8 (B)	pin 2 (Enc1A)
2	pin 6 (A)	pin 5 (Enc2B)
2	pin 8 (B)	pin 4 (Enc2A)
1 and 2	pin 2 (+5V)	pin 1 (5Vout)
1 and 2	pin 3 (GND)	pin 6 (GND)

[4] Datasheet is available at `http://datasheet.octopart.com/HEDL-5500%23G12-Avago-datasheet-8328570.pdf`

4.4.1 *Tuning*

The encoder readings and speed references sent to the motor controller are somewhat arbitrary numbers. It is more useful to deal with actual angular wheel speeds. The measurements coming directly from the motor encoders are altered by several gearings etc., and therefore have to be multiplied by a constant. This constant could be calculated fairly accurately by finding gear ratios from motor to wheel and so on. However, it is much easier and more accurate to find the constant through a simple experiment. An experiment to determine how much time the wheels use to complete a fixed number of rotations given a constant speed was conducted. Table 4.2 shows a selection of the measurements.

TABLE 4.2: Selected measurements to determine angular wheel speed from encoder measurements.

Motor	Encoder speed	Rotations	Time [s]	Angular speed [rad/s]
1	50	5	44.8	0.70
1	100	10	43.5	1.44
1	150	10	29.5	2.13
1	200	10	22.0	2.86
2	100	10	44.0	1.43

The numbers appear to have a linear relationship. Linear regression analysis comes up with the following conversion from encoder reading ω_{enc} to angular wheel speed ω_w:

$$\omega_w = 0.0143\omega_{enc} \tag{4.1}$$

4.4.2 *Relationship Between Wheel Speeds and Robot Velocities*

The mathematical model of the robot and all the controllers in this report use forward velocity u and angular velocity ω. For that reason, individual angular wheel velocities have to be recalculated to form u and ω. Using the unicycle model with differential steering the velocities are given as [25]:

$$u = \frac{r}{2}(\omega_{w,l} + \omega_{w,r}) \tag{4.2}$$

$$\omega = \frac{r}{d}(\omega_{w,r} - \omega_{w,l}) \tag{4.3}$$

where $\omega_{w,l}$ and $\omega_{w,r}$ are left and right angular wheel velocities, r is wheel radius and d is wheel track, i.e. the distance between the wheels. r was easy to measure, but d is more difficult due to the fact that the wheels are very wide. After a few tests to ensure that the values provided correct values of u and ω, the values of r and d for the Dogmatix robot were found to be

$$r = 0.2698\,\text{m}, \quad d = 0.985\,\text{m} \tag{4.4}$$

As already mentioned, the motor controller needs individual wheel speed references instead of u and ω. These can easily be found by rearranging (4.2) and (4.3):

$$\omega_{w,r} = \frac{1}{r}\left(u + \frac{d}{2}\omega\right) \tag{4.5}$$

$$\omega_{w,l} = \frac{1}{r}\left(u - \frac{d}{2}\omega\right) \tag{4.6}$$

which need to be scaled using (4.1) before being sent to the motor controller.

4.5 IMU & GPS

An IMU and a GPS unit was also available for use to improve velocity and position estimates. The IMU was tested in collaboration with wheel encoders. Measurements were fused together using an ROS implementation of an extended kalman filter (EKF). It turned out, however, that the encoder measurements alone were very accurate in the test conditions with a completely flat floor and virtually no wheel slips. Therefore, to avoid the time usage of correctly mounting and verifying that the IMU measurements were correct, the IMU was not used for the results in this report. The GPS was used briefly during the outdoor field test with Asterix, but not for any other tests.

4.6 ON-BOARD COMPUTER

The "head" of the robot right in the front houses an on-board computer. This is just a normal desktop computer with mini-ITX form factor running Lubuntu 14.04 as operating system. It has four 9-pin D-SUB serial ports, one of which is used for communication with the motor controller. The computer receives wheel speeds from the motor controller and use these to update velocity, position and orientation states. It also sends desired individual wheel speeds back

to the motor controller. Basically, all logic and controllers run on this computer. An Xbox controller was also connected to the computer which allowed for manual control of the robot. A picture of the computer is shown in Figure 4.3.

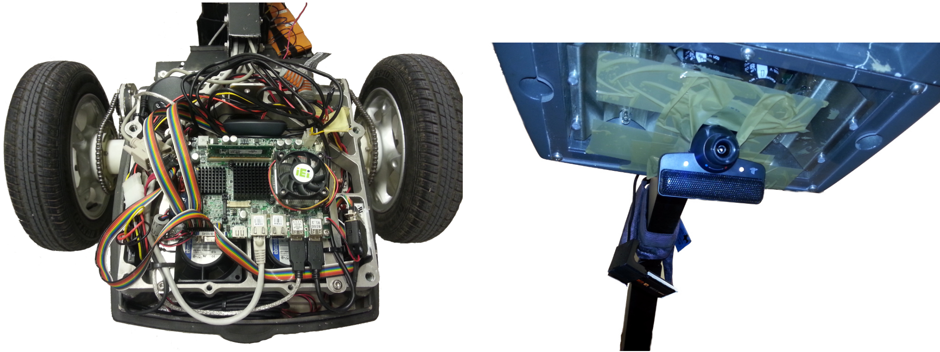


FIGURE 4.3: Left picture shows the on-board computer in the "head" sticking out in the front of Dogmatix. Right picture shows the PlayStation Eye camera which is mounted upside down on the underside of the head.

4.7 CAMERA

A *PlayStation Eye*[5] camera is mounted at the front of the robot pointing downwards at an angle of about 45 degrees. The camera's position and orientation is fixed. PlayStation Eye was developed by Sony to be used with the PlayStation 3 gaming console to allow players to interact with games by doing gestures in front of the camera. It was designed with computer vision in mind, making it suitable for this application. Some advantageous features include:

- Very low cost (about 15 USD at time of purchase).
- Standard USB connection and recognizes as a standard web camera. Expensive higher end cameras often offer more advanced settings, but at the same time usually rely on proprietary drivers.
- High framerate of 60 frames per second at 640×480 resolution.
- Fairly quick shutter speed and exposure time, which means that pictures taken at speed will be sharp.

A picture of the camera is shown in Figure 4.3.

[5] http://us.playstation.com/ps3/accessories/playstation-eye-camera-ps3.html

5 DEVELOPMENT

This chapter covers some of the development that went into the simulator and the various computer vision techniques and methods used for detection and estimation of row crops in fields. It also covers the mapping from image coordinates to robot coordinates. Some details about implementation in OpenCV are also provided.

5.1 SIMULATOR

Developing code directly on a robot can be time consuming and at times even frustrating. Having a good simulation environment can help speed up development massively, so it is worth putting some time into it. During last semester the author had no simulator available. Instead, simulations of controllers were performed using Matlab/Simulink with a nonlinear model. This worked fine for testing models and controller performance, but the Matlab code or Simulink diagram had to be completely rewritten to C++ or Python and incorporated into ROS before testing on the real robot. It became apparent that for the thesis a better simulation environment had to be developed, so a realistic simulator was developed using Gazebo[1], an open source robot simulator. The advantages are many:

- Realistic physics can be achieved by supplying the simulator with masses, inertia matrices, friction coefficient etc.
- The interface between ROS and Gazebo can be made identical to that of ROS and real robots. This means that code developed using the simulator can be directly transferred to the real robot.
- Sensors can be simulated. Wheel odometry, IMU and GPS can easily be simulated with realistic, adjustable noise. Gaussian noise, drift and discrete jumps in measurements can all be simulated.
- A camera can be installed on the robot.

[1] http://gazebosim.org/

- Realistic surroundings can be made, e.g. a realistic field with crops.

The development of the simulator was a fairly time consuming task. Most of it consists of creating XML-like configuration files to describe the robot and environments. This is not the most interesting of tasks to read about, but some of the main parts are documented here.

5.1.1 *Robot Design*

A 3D model of the chassis of the robot was developed using Blender[2], an open source 3D creation suite. A screenshot during the development is shown in Figure 5.1. Once the chassis mesh was created, the mesh was imported to MeshLab[3], an open source tool for processing meshes. MeshLab can compute an inertia matrix from the chassis mesh with the assumption of constant density. The inertia matrix has the following form:

$$I = \begin{bmatrix} i_{xx} & i_{xy} & i_{xz} \\ i_{xy} & i_{yy} & i_{yz} \\ i_{xz} & i_{yz} & i_{zz} \end{bmatrix} \tag{5.1}$$

The real robot will obviously consist of more mass than just the chassis, but as an approximation this should suffice.

The robot is specified in a file using Unified Robot Description Format (URDF), which is an XML format for describing a robot. This is the standard format used in ROS and allows one to add e.g. wheels and camera to the chassis. This is also where masses, inertia matrices and friction coefficients are specified. The electric motors are also specified here, while the PID motor controllers to use speed references are implemented in an ROS node. A graph diagram showing the position and orientation of wheels, imu and camera relative to the chassis is shown in Figure 5.2. Joints connect the different parts and lets e.g. the wheels rotate around the y-axis.

Gazebo does not directly support URDF, but instead offers its own format called Simulator Description Format (SDF), which is also an XML format. It is similar to URDF, but offers more features relevant to the Gazebo simulator. Luckily, there exists parsers that make it possible to use URDF for the most part. In the end, URDF was chosen as the main format for our simulator to comply

[2] `http://blender.org/`
[3] `http://meshlab.sourceforge.net/`

with ROS standards, with the addition of some extra attributes specified using SDF.

The finished robot can be seen in Figure 5.3. The end result is a robot that behaves similarly to the real robot, has the same sensors and communicates with ROS using topics and messages in exactly the same way as the real robot. Code developed using the simulator could be used directly on the real robot without any changes.

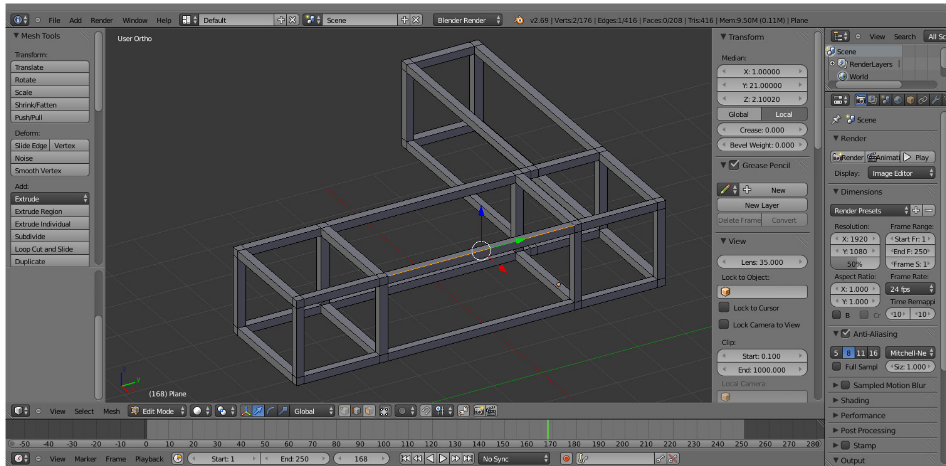


FIGURE 5.1: A realistic chassis was developed using Blender.

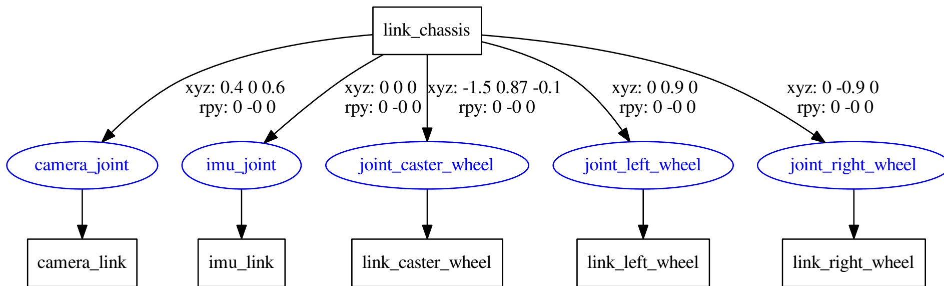


FIGURE 5.2: Graph diagram auto-generated from URDF file illustrating the different parts of the robot, their position relative to the chassis and how they are connected with joints.

5.1.2 *Simulated Field*

A simulated field was created to make a realistic environment for the robot to operate in. The field that was made is similar to the one used in carrot fields.

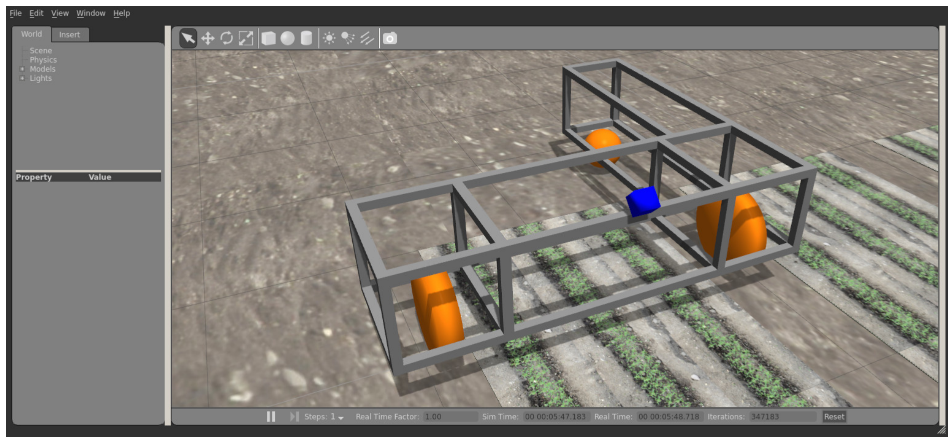


Figure 5.3: Screenshot of Gazebo simulator with the finished robot placed in a simulated field with crops. The small, blue box is the camera.

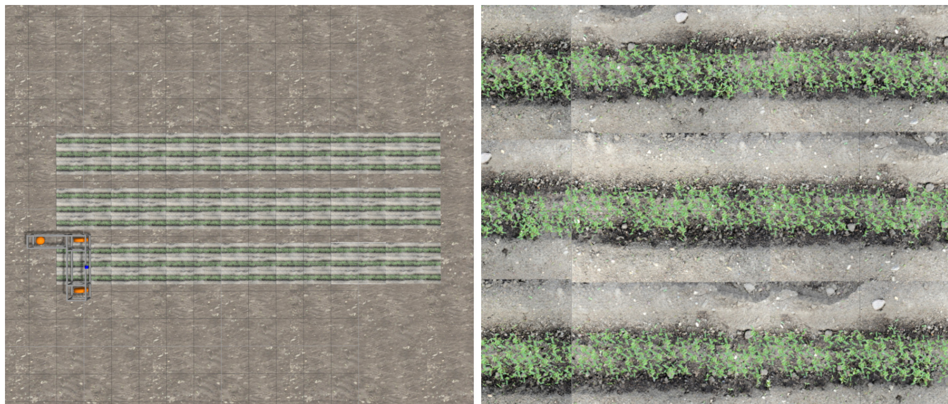


Figure 5.4: Left image shows the entire simulated field. Right image shows a closeup of a row.

Each row consists of three smaller rows, and the robot uses the middle of the small rows for estimating a line to follow. Pictures of the field are included in Figure 5.4. The field was made of real images patched together. There is no height in the field, only flat images.

5.2 OVERVIEW OF COORDINATE SYSTEMS

Representing positions, orientations and velocities in different coordinate systems can greatly simplify problems. ROS provides framework for defining co-

ordinate systems and constantly update their rotational and translational matrices. This makes it easy to convert between coordinate systems at any point in time. E.g. if we would like to know the position of the robot three seconds ago given in a global coordinate system, this can be easily retrieved using the provided ROS framework. More specifically this functionality is provided by the *tf* package[4]. The following coordinate systems (frames) were defined in ROS to be compliant with ROS standards REP-105[5] and they all comply with ”right-hand-rule” with z-axis pointing upwards:

- **Map:** A world fixed frame. The position and orientation (pose) of the robot relative to map frame should not drift significantly with time. Typically this frame should be updated with absolute pose measurements frequently, e.g. from GPS.
- **Odom:** Also a world fixed frame. The only difference from *map* frame is that *odom* frame is allowed to drift without bounds. This obviously makes it a bad choice for long-term global reference frame, but it also ensures it to be continuous, which is an important property for many controllers. Typically the *odom* frame is computed using odometry data, such as wheel odometry, visual odometry or inertial measurement unit (IMU).
- **base_link:** Rigidly attached to the robot. In the case of this robot the origin is chosen to be in the middle of the wheel axle with x-axis pointing straight forward and z-axis up.

The frames are related in a tree, i.e. each frame can only have one parent but many children. An illustration is shown in Figure 5.5.

In addition to these three coordinate systems there’s a fourth one, namely the image coordinate system which is not defined as part of ROS. Instead, a transformation from image coordinates to *base_link* was developed and will be covered in later section. With this transformation in place, ROS handles the rest of the transformations using *tf*.

5.3 MAPPING FROM IMAGE COORDINATES TO ROBOT COORDINATES

As mentioned in Section 4.7 the Dogmatix robot has a camera mounted in the front angled about 45 degrees downwards. The purpose of this is to identify and

---

[4] `http://wiki.ros.org/tf`
[5] `http://www.ros.org/reps/rep-0105.html`

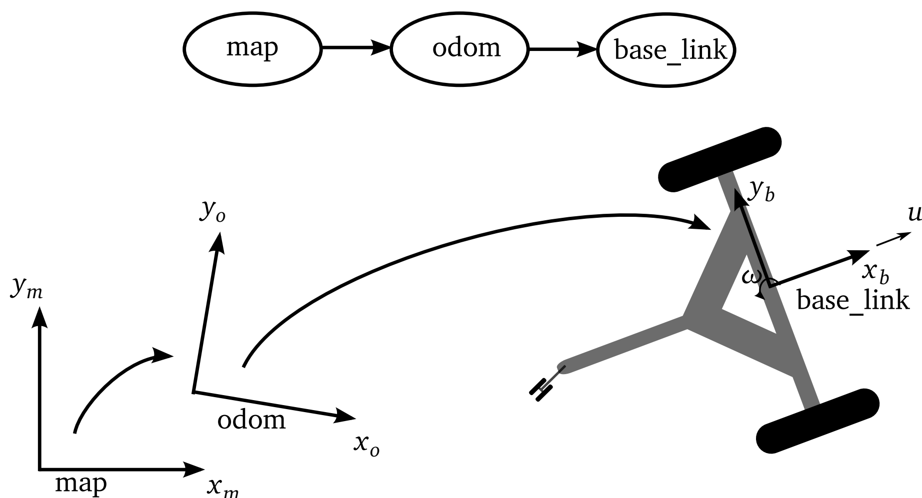


FIGURE 5.5: Illustration of the different coordinate systems. The frames are related in a tree, i.e. each frame can only have one parent but many children.

estimate a crop row and its position and orientation relative to the robot. This means that once the row has been identified in an image, we would like to map the coordinates of the row into *base_link* frame. From *base_link* the coordinates can easily be transformed into *odom* or *map* using the *tf* package.

Recall from Section 2.2 how a number of intrinsic and extrinsic parameters can be used to describe the mapping of three-dimensional world coordinates to two-dimensional image coordinates as given in (2.6). Identifying all the unknown parameters reliably can be a challenge. In our case there are several factors that allow us to simplify the model:

- The extrinsic parameters, i.e. the position and orientation of the camera, are constant since the camera is rigidly mounted to the robot and the robot is assumed to follow the ground at all times.
- The intrinsic parameters remain constant. Zooming or factors that otherwise could alter intrinsic parameters should not occur.
- The ground is assumed to be flat, i.e. it forms a plane in the three-dimensional space. The height of the plants and height differences due to uneven ground are assumed to be negligibly small.

The last bullet point is important. It reduces the mapping problem from world to image coordinates into a simple projective transformation as described in Section 2.3.6 and (2.19). This is illustrated in Figure 5.6. Therefore, the map-

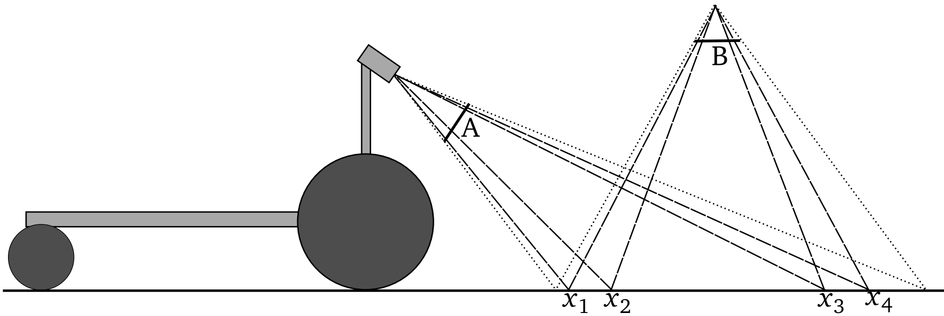


FIGURE 5.6: Illustration of robot with front mounted camera angled about 45 degrees downwards. The image captured by the camera is that of plane A, while B illustrates an image plane after applying a perspective transformation to A. Notice the difference in spacing of the three points x_1, x_2, x_3 and x_4 after projecting to image plane A and B.

ping of a point (pixel) in the image represented with homogeneous coordinates $\bar{\boldsymbol{x}}_{\boldsymbol{i}} = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix}^T$ to a point in *base_link* frame $\bar{\boldsymbol{x}}_{\boldsymbol{b}} = \begin{bmatrix} \bar{x}_b & \bar{y}_b & \bar{w}_b \end{bmatrix}^T$ is given as

$$\bar{\boldsymbol{x}}_{\boldsymbol{b}} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \bar{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{H}\bar{\boldsymbol{x}}_{\boldsymbol{i}} \tag{5.2}$$

where $\boldsymbol{H}$ is a homogeneous matrix. The homogeneous coordinates can easily be transformed to Cartesian coordinates using (2.9). In practice the mapping process will be divided into a few more steps, which will be covered in the next subsections.

5.3.1 *Finding Homography Matrix*

OpenCV has lots of functionality to estimate homography matrices and apply perspective transformations to images. First we will consider ways to estimate $\boldsymbol{H}$. Recall that $\boldsymbol{H}$ has eight degrees of freedom, which means that at least eight equations are needed to find all elements of $\boldsymbol{H}$. The eight equations can for example be the known transformation of four points. Consider Figure 5.7. To the left is a rectangle with its correct dimensions without any perspective, and on the right is the rectangle the way it appears in an image from a perspective view. This means that if the corners of the rectangle can be identified in the image, then we have the known transformations of four points. OpenCV has a function called *getPerspectiveTransform* that takes the

points $(x_1, x_2, x_3, x_4, x'_1, x'_2, x'_3, x'_4)$ as argument and returns an estimate of $\boldsymbol{H}$.

Figure 5.8 shows how this is done in practice. OpenCV has functions to recognize chessboards and finding the corners. A printed chessboard was placed flat on the ground in front of the robot. The located corners are passed as arguments to *getPerspectiveTransform* to estimate $\boldsymbol{H}$. The function *warpPerspective* performs the transformation of the image with the estimated $\boldsymbol{H}$. During the transformation some pixels need to be interpolated while others will be extrapolated.

Another thing that can be adjusted is the "zoom" level, or more precisely the resolution, of the transformed image. Recall from (5.2) that after a projective transform the coordinates need to be transformed from homogeneous to Cartesian coordinates, i.e.

$$\boldsymbol{x_b} = \begin{bmatrix} x_b & y_b \end{bmatrix}^T = \begin{bmatrix} \frac{\bar{x}_b}{\bar{w}_b} & \frac{\bar{y}_b}{\bar{w}_b} \end{bmatrix}^T \tag{5.3}$$

From (5.2) it can also be seen that

$$\bar{w}_b = h_{31} x_i + h_{32} y_i + h_{33} \tag{5.4}$$

Effectively this means that the "zoom" level can be adjusted by altering h_{33}. A larger value of h_{33} leads to smaller values of x_b and y_b which in turn lead to higher resolution, i.e. the image appear more "zoomed". Choosing higher values of h_{33} might prove beneficial as it can provide higher resolution and detail level. Keep in mind though that there is little point in attempting to achieve greater resolution than the input image, as that will eventually just lead to extrapolation of pixels.

5.3.2 *Mapping from Transformed Picture to Robot Coordinates*

After the projective transformation two important properties are restored in the transformed picture:

- Angles are correct. If a line is rotated 45 degrees relative to the robot, the line in the transformed image will also be 45 degrees. In addition, parallel lines will now be parallel.
- Lengths are equal over the entire image. E.g. the chessboard in Figure 5.8 will have the same size no matter where on the image it is located.

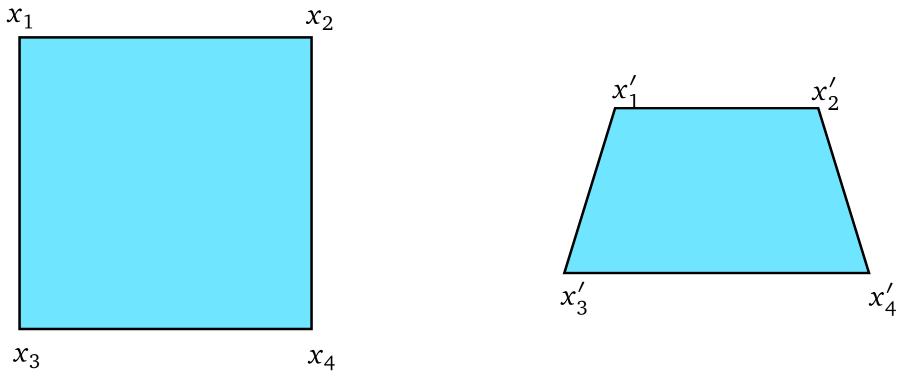


FIGURE 5.7: x'_1, x'_2, x'_3 and x'_4 are points in an image corresponding to corners of a rectangle with corners x_1, x_2, x_3 and x_4. Since we know that the corners in the image correspond to the corners in the real world rectangle, this can be used to estimate a homogeneous matrix.

FIGURE 5.8: The left image shows a picture of a chessboard pattern taken with the camera mounted on Dogmatix. All identified corners have been marked with circles and lines and are used to find a homography matrix for a projective transform. The right image shows the same picture after the projective transformation.

With these two properties the transformation between image pixel coordinates and *base_link* coordinates is fairly trivial. By considering Figure 5.9 it is easy to verify that a pixel point in an image $\boldsymbol{x_i} = (x_i, y_i)$ can be expressed in *base_link* coordinates as

$$x_b = i_x - i_p y_i, \qquad y_b = i_y - i_p x_i \tag{5.5}$$

where i_p is a constant with unit meter per pixel which converts pixels to actual length. i_p can also be found from the chessboard in Figure 5.8. If the chessboard

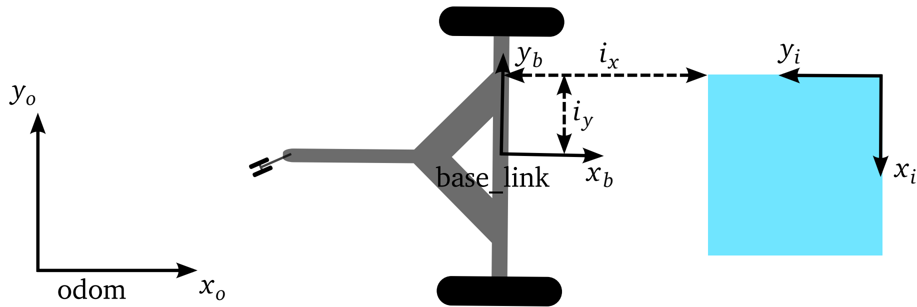


Figure 5.9: Illustration of mapping from image coordinates in a projective transformed picture to robot *base_link* coordinates. The square to the right represents a transformed picture.

has a real width of e.g. 0.50 m and the pixel width in the transformed picture is 80 pixels, then

$$i_p = \frac{0.50\,\text{m}}{80\,\text{pixel}} = 0.00625 \text{ m/pixel} \tag{5.6}$$

5.4 FIND ROW CROPS IN IMAGES

This section will cover the techniques used to find and estimate row crops in a field from images.

5.4.1 *Calculating Reference Histogram*

A histogram backprojection technique is used to create a binary image of all the plants/crops in an image. The first step is to create a reference histogram based on images of plants. A small program was developed that allows one to pick the sections of the image that contains plants and calculate a histogram based on those regions. The program works in the following way:

- An input image that contains clear images of plants is chosen (e.g. closeup of a row crop).
- The user clicks on the plants that should be used as reference with the mouse pointer. The selected pixel is passed on to a flood fill function that attempts to select the entire plant/leaf.

- Once all interesting regions have been selected, a two-dimensional histogram based on the hue and saturation values of the selected plants is calculated. The histogram is saved to file and will be used later for back-projection to identify plants in real time.

A screenshot of the program is shown in Figure 5.10.

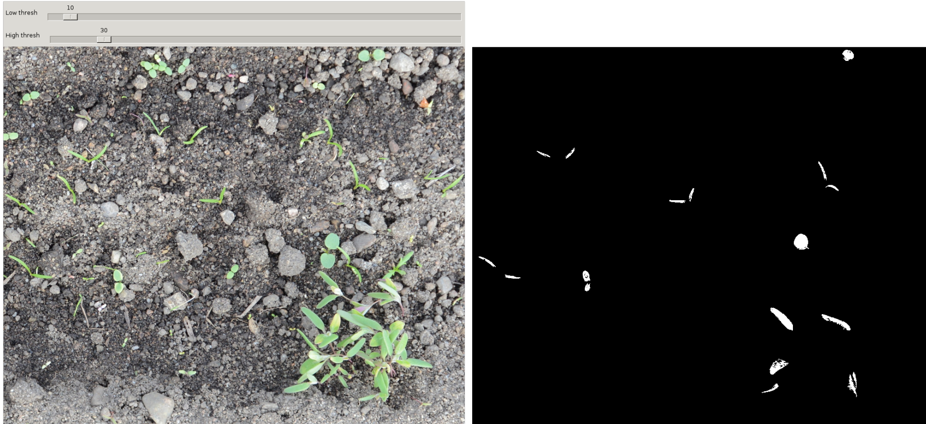


FIGURE 5.10: A screenshot from the program that was developed to select regions of interest in an image and compute a two-dimensional histogram of hue and saturation values from the selected regions. The left picture shows the program window with the input image that allows the user to click on regions they want to select (the leafs in this case), and the right picture is a binary mask that shows the selected regions.

5.4.2 *Histogram Backprojection*

Once a reference histogram of plants is calculated, it can be used to backproject an image to find pixels that have high probabilities of belonging to a plant. High probability pixels are shown as light gray, close to white, while low probability pixels are shown as darker gray, close to black. The pixels that have a higher probability than a specified constant are thresholded, i.e. they are completely white in the binary image. Finally, dilation followed by erosion is applied to the binary image to close gaps and remove outliers. This process is shown in Figure 5.11.

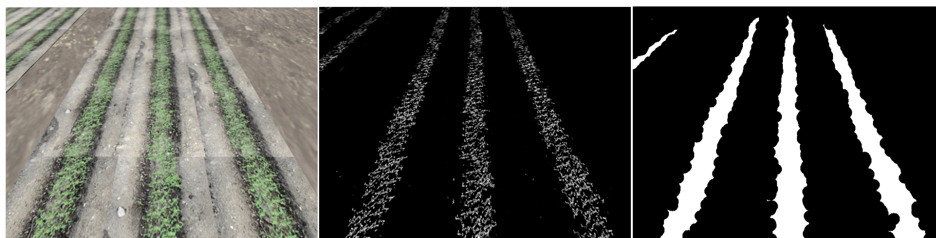


FIGURE 5.11: The left image is a raw image input from the simulated robot's camera while in a simulated field. The middle image shows a probability image after backprojecting the input using the same reference histogram that was calculated in Figure 5.10. The right image shows a binary image after thresholding the probability image and applying dilation and erosion to close gaps and remove outliers.

5.4.3 *Projective Transformation*

In Figure 5.11 it can be seen that the three row crops that are parallel in real life are not parallel in the image. A projective transformation is performed to correct this. This is trivial at this point, since the homography matrix $\boldsymbol{H}$ has already been found using the chessboard that was shown in Figure 5.8. The result after a projective transformation can be seen in Figure 5.12.

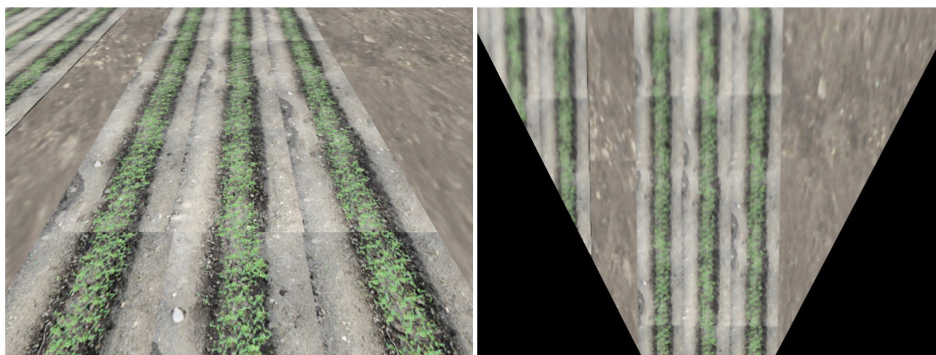


FIGURE 5.12: The left image is a raw image input from the simulated robot's camera while in a simulated field. The right image shows the input after a projective transformation. Note how the row crops are parallel after the transformation.

5.5 ESTIMATING ROW CROP POSE

After performing a projective transformation, backprojection and finally thresholding, the output is a binary image of what should be parallel row crops. The

final step is to estimate the pose, i.e. position and direction, of the middle row. If the image only contains one row, then using a line fitting technique from Section 2.6.2 will work well. However, in this case there are three rows, which means that it if one attempts to run a line fitting method directly one would essentially attempt to fit one line to three rows. This is ensured to give inaccurate results, as there is no guarantee that the middle line will be chosen. To solve this, a method to pick out only the middle row had to be developed.

One possible approach could be to modify the RANSAC method to store not only the best line, but several of the best lines. After all the line candidates have been identified, one would have to pick the three best lines that are spaced apart by some distance. The reason for this is that the method might return several line candidates for the same row, and the goal is to pick the best line candidate for each row. Finally, the "middle" line should be selected, which should be the best pose estimate of the middle row. This method was never implemented and is left for future work if needed.

Another solution is to use Hough lines to find all lines in the image and then try to pick the best line. Hough lines is in fact a very robust tool for this application. It has two very important parameters that can be adjusted:

- **Minimum line length:** Ignore all lines that are too short. Too short lines are unreliable, so having a way to ignore them is important.
- **Maximum line gap:** Allow some gaps in the binary image while still being recognized as the same line. However, if the gap is too big then it is unlikely that the line is part of the row and should be ignored.

OpenCV has a good implementation of Hough lines using a function called *HoughLinesP* based on a probabilistic Hough transform presented in [26]. This probabilistic implementation is based on a "voting" scheme and is fast enough to be used in real time. The function returns a list of identified lines sorted from most to least votes. The problem is that the list will most certainly contain several lines per row, so a method is needed to divide the lines into their respective rows. Therefore, lines are divided into "classes" based on their distance and angle compared to surrounding lines. Finally, the line with the most votes is picked from the "middle" row class.

The method that was finally implemented is called method 1 from here on, and can be summarized in the following steps:

Method 1

Step 1: Find all lines in the binary image using Hough lines that match the criteria for minimum line length and maximum line gap using the OpenCV function *HoughLinesP*.

Step 2: Select the line with most votes, i.e. the first line in the list (denoted l_1). Iterate over all other lines and calculate the distance from a specified point on l_1 to the currently selected line and their angle difference. If the distance and angle difference are small, the line is classified to be in the same row as l_1.

Step 3: Select next line from the list that has not yet been classified and make this into a new class. Repeat step 2 for this line.

Step 4: Repeat step 2-3 until all lines have been classified.

Step 5: If there are less than three classes, return with no line identified. If there are more than three classes, select the three classes that contain the most lines.

Step 6: Select the line with most votes from each of the three classes. These three lines should correspond to their respective rows.

Step 7: Select the line that is located between the other two lines. This line should be the middle row. The selection of the middle line is done by simply averaging the image x coordinates for the end and start points of the lines. The left row will have smallest value and right row will have largest value. This method works well unless the robot is almost perpendicular to the rows.

The entire process from input image to identified middle row is shown in Figure 5.13. Each image is explained below, starting from top left and moving right then bottom left to right:

Image 1: Input image

Image 2: Input image after performing a projective transformation

Image 3: Cropped version of previous image to remove the areas around the rows.

Image 4: Probability image after backprojection.

Image 5: Binary image after thresholding high probability pixels followed by dilation and erosion to fill in gaps and remove potential outliers.

Image 6: All the lines identified by *HoughLinesP*. The identification is done on the binary image, but drawn on the cropped projective transform image.

Image 7: After classification of all lines, the line with the most votes from each class is selected. The left and right rows are drawn with green lines, while the middle row is red.

Finally the identified middle row is transformed to global *odom* coordinates and can be passed on to the controller directly or to the Kalman filter as a measurement for row estimation.

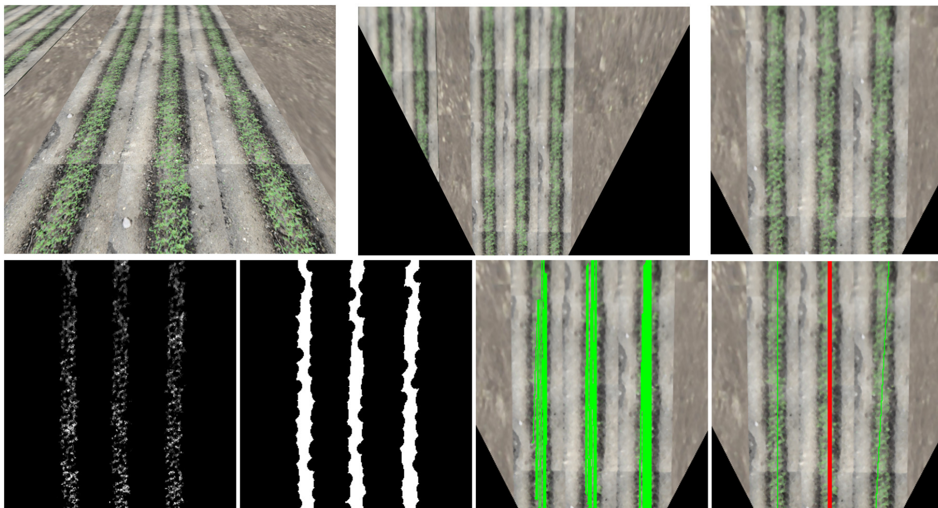


FIGURE 5.13: A simulation example showing the entire process from input image to identified middle row using the probabilistic Hough lines method, method 1, to find straight lines.

5.5.1 *Combining Hough Lines and Line Fitting*

Recall from Section 2.6.1 that the Hough lines algorithm works by finding lines along edges. Consequently, an edge detector is usually applied before applying Hough transform, but that was not done in method 1 that was described in the previous section. In effect, this means that every time the method runs, the line that is selected in the end can be anywhere on the row. It can be to the far left of the row, to the far right, in the middle or cross from left to right. This means

that the identified row pose will vary quite a lot from image to image, even if the images are identical. Since the row pose can be used as a measurement in the Kalman filter that estimates a row, these fluctuations are mostly smoothed. Still, improving the quality of the measurements will be beneficial.

If an edge detector is applied to a binary image, it will be equivalent to finding the contours of the white parts. By doing that it can be expected that there will be fewer detected lines and they will be situated along the contours, i.e. on left and right side of the row. By somehow averaging these lines it should be possible to obtain a line along the middle of the row. Some tests were attempted to achieve this, but the results were not convincing, and some of the robustness was lost when the number of detected lines was decreased.

A line fitting method like least squares does a much better job at finding the center of a row than Hough lines. To utilize this advantage, a method combining both Hough lines and line fitting was developed. The line identified by method 1 provides an approximate location of the middle row. By expanding the area around the line and then applying a line fitting method for that area, the advantages of both methods are achieved. The steps of method 2 can be summarized as:

Method 2

Step 1: Use method 1 to obtain a line that approximates the middle row.

Step 2: Expand the area around the line and choose only this region from the binary image. The binary image should now only include the thresholded middle row.

Step 3: Apply normal least squares method (2.27) to fit a line to the selected are of the binary image. The fitted line should approximate the middle of the row. Note that since only the area just around the row is selected, there shouldn't be any outliers and the normal least squares can be used instead of the much more resource intensive RANSAC method.

Figure 5.14 shows a simulation example using method 2. Notice how the line identified using method 1 is not quite straight and goes from right to left side of the row, but after applying method 2 the line is straighter and closer to the middle of the row.

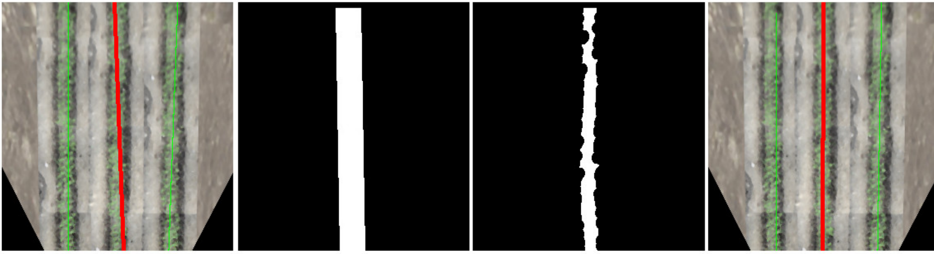


FIGURE 5.14: A simulation example of method 2. Left image shows the middle row identified by method 1, next image shows the expanded area around the line, next image shows the selected part of the binary image ("bitwise and" with previous image), and the right image shows the line after a least squares fitting.

5.6 END OF ROW AND TURNING STRATEGY

At the end of a row the robot should turn around and continue with following the next row, unless it's at the very last row. The procedure is illustrated in Figure 5.15 and can be summarized as:

- Detect the end of a row.
- Exit the row. This means going straight for at least the length of the entire robot to ensure that the rear caster wheel has completely exited the row.
- Turn around 180 degrees, either left or right. If e.g. turning right, the right wheel can remain stationary during the entire turn, because it will enter the next row in the same wheel track.
- After completing the turn, move straight forward while attempting to detect the new row using camera.
- If a new row is detected, start following this.

5.6.1 *Detecting End of Row*

The end of a row is detected if a specified number of seconds has passed without receiving any valid row pose updates. The number of seconds can be specified as a length since the robot moves at constant forward speed. The total length d_r from last row pose update until it is considered the end of a row consists of several lengths:

$$d_r = d_a + d_b + d_c + d_d \tag{5.7}$$

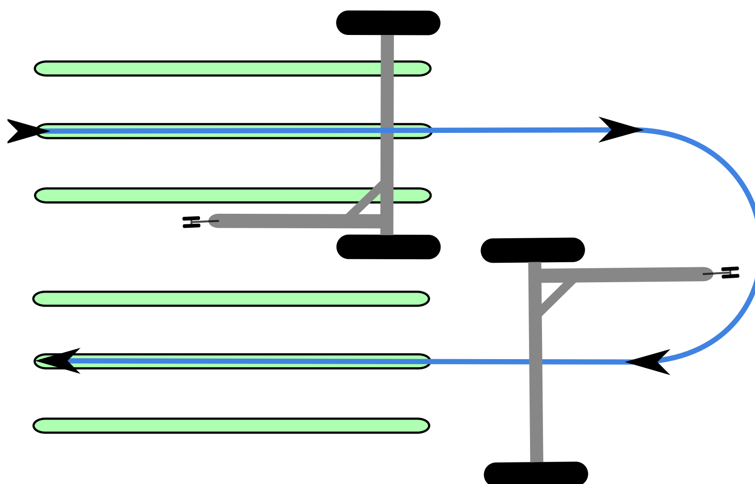


FIGURE 5.15: Illustration of how a turnaround at the end of a row may be performed. It is important that the robot completely exits the row to ensure that the rear caster wheel does not destroy any crops.

where d_a is length of robot, d_b is distance from wheel axle to lowermost part of the image, d_c is minimum line length used in the Hough lines method and d_d is extra length margin. These lengths are illustrated in Figure 5.16. The number of seconds s_r from last row pose update until end of row is detected and turning can begin is simply

$$s_r = \frac{d_r}{u} \tag{5.8}$$

where u is the constant forward speed. This method is a very simple one, but has proved robust in practice. The robustness of this strategy relies on a robust detection of rows. False positive row pose updates will cause the robot to move too far out of the row, while positive false updates can potentially cause the robot to start turning in the middle of a field. With that said, the visual row detection methods have proven to be very robust, so this has not been a problem during testing. In the final system this could also be coupled with e.g. GPS position to ensure that turning only happens when the robot is positioned along the edge of the field.

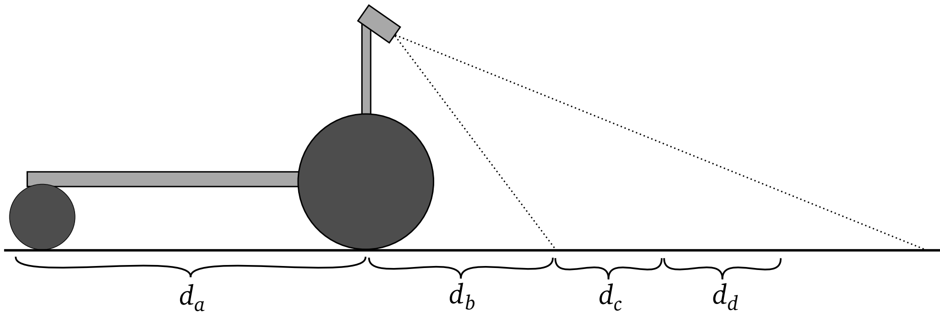


FIGURE 5.16: Illustration of the different lengths that when added up give the total length the robot has to move before starting a turn.

5.6.2 *Turning Controller*

A very simple P-like controller was implemented to turn 180 degrees around at the end of a row.

$$\omega_r = \begin{cases} |k_p(\psi_r - \psi|, & \text{if turning left.} \\ -|k_p(\psi_r - \psi)|, & \text{if turning right.} \end{cases}, \qquad u_r = \left|\frac{d}{2}\omega_r\right| \tag{5.9}$$

where k_p is a gain constant, ψ_r is the reference heading angle and d is the distance between the wheels. In addition, there is a saturation to limit $|\omega_r|$. The absolute values are chosen to ensure that the robot turns in the correct direction. Clearly, this makes the controller unstable if ψ overshoots ψ_r, but that is not a problem because the controller shuts down as soon as ψ reaches ψ_r. At the end of a row, ψ_r is set to the current value of ψ plus $180°$ before the turning controller is activated.

During the turning it is desirable that one of the wheels remain stationary. That is equivalent to following a circle where the stationary wheel is at the center of this circle and the other wheel follows the edge of the circle, which means that u is the speed at the half point of the radius of the circle. The half point is half of the wheel distance d, which gives the control law for u_r in (5.9).

5.7 ENTIRE SYSTEM

The entire system with all the components combined is designed as a state machine. Figure 5.17 shows a diagram of all the states and the transitions. The system consists of only four states:

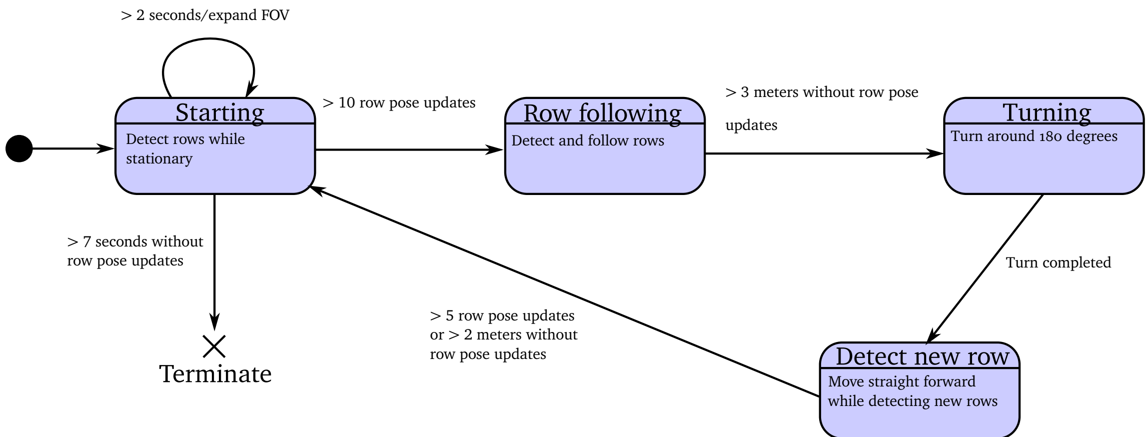


FIGURE 5.17: Diagram of the different states and transitions during operation. The numbers used for seconds and distances are chosen for illustration purposes and can be adjusted to needs.

States

State 1 **Starting state:** The robot remains stationary while attempting to detect rows. If no rows are detected within the first few seconds, the field of view (FOV) of the camera is expanded to search a bigger area. If there are still no detected rows, the system terminates.

State 2 **Row following state:** This is the normal operating state where the robot follows the rows. If the robot has not received any row pose updates in a specified amount of time/distance, end of row is assumed.

State 3 **Turning state:** Turn around 180 degrees in either left or right direction.

State 4 **Detect new row:** Robot moves forward at constant speed while camera is searching for the next row. Once several detections are done, change to starting state.

5.7.1 *Implementation in ROS*

The system is implemented in C++ using ROS libraries. A somewhat cluttered auto generated graph is shown in Figure 5.18. The main part of the system is divided into three different nodes:

- **vision_hough_node:** Receives images from the camera and performs all the image processing. The output of this node is an estimated row pose in global *odom* coordinates.
- **row_global_kalman_filter_node:** Receives row poses from *vision_hough_-node* and incorporates them as measurements in the Kalman filter. The output of this node is the filtered row pose.
- **row_global_controller_node:** Receives filtered row poses from *row_-global_kalman_filter_node* and uses them to calculate reference speeds u_r and ω_r that are sent to the motor controller. The state machine is also implemented in this node.

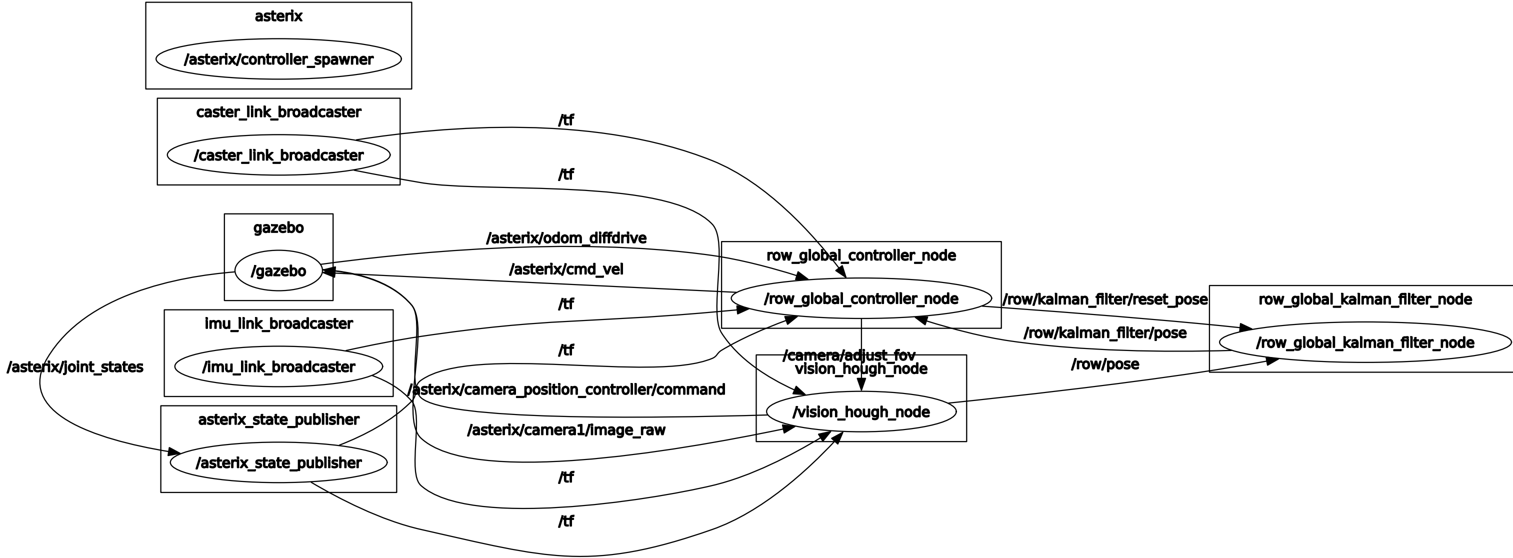


FIGURE 5.18: Auto generated figure from ROS during a simulation that shows the different nodes and how they are communicating with messages. The left part is the simulated robot, while the three nodes to the right make up the system. During operation on a real robot only the left part will look different.

6 SIMULATIONS

This chapter contains a selection of simulations that were performed using the simulation environment that was developed using Gazebo and ROS.

6.1 LINE FOLLOWING CONTROLLER

The line following controller (3.11), also referred to as row controller, was tested by defining a line in global coordinates and letting the robot follow it at constant forward speed starting from different initial positions. The following controller gains were used:

$$k_1 = 2.0, \;\; k_2 = 0.26, \;\; k_3 = 5.0 \tag{6.1}$$

where k_2 was found by defining the desired approaching angle to 15 degrees, i.e. $k_2 = \sin^{-1}(15°) \approx 0.26$. Larger k_1 means larger ω_r which leads to quicker response. Larger k_3 leads to bigger emphasize on the distance to the line d. Since the controller is very intuitive and does not involve any dynamics, it was very easy to tune.

Figure 6.1 shows a simulation where the robot starts in a parallel position to the line, while Figure 6.2 shows a similar simulation when the robot is initially facing away from the line.

The performance is overall good. First of all, it can be seen that the assumption of $\omega \approx \omega_r$ appears to be valid for the simulation. The approaching angle θ stabilizes at the expected value of $\theta = \pm \sin^{-1}(k_2)$ while approaching the line. There is very little overshoot and oscillations once it reaches the line. Note that some Gaussian noise has been added to the speed measurements obtained from wheel odometry in Gazebo.

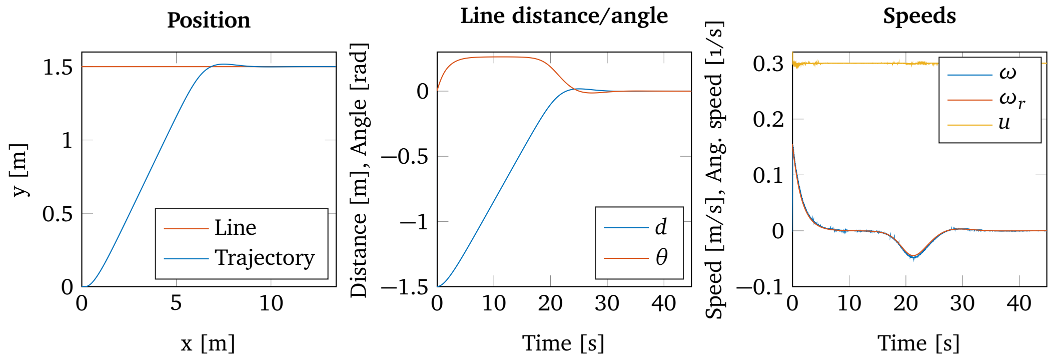


FIGURE 6.1: Simulation of the row controller with robot starting point $(x, y, \psi) = (0, 0, 0)$ and a line defined in global coordinates as $y = 1.5$. Notice how the approaching angle θ stabilizes at $\theta = \sin^{-1}(k_2) \approx 0.262 = 15°$ while approaching the line.

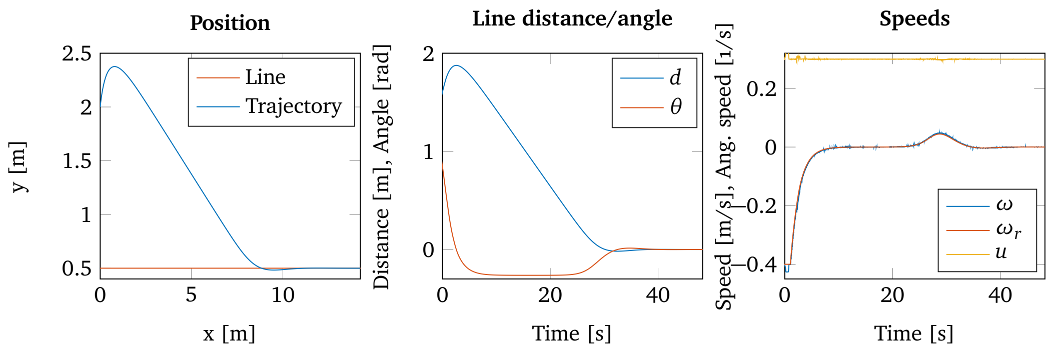


FIGURE 6.2: Simulation of the row controller with robot starting point $(x, y, \psi) = (0, 2, \pi/3)$ and a line defined in global coordinates as $y = 0.5$.

6.2 IDENTIFYING AND FOLLOWING A SINGLE LINE

In this simulation the goal is to verify the identification of a line using a simulated camera, map the line in global coordinates and use the line following controller to follow it.

6.2.1 *Estimating Line Pose*

Simple tests were conducted to test how precise the mapping from image coordinates to global *odom* coordinates is. Using the simulator an infinitely long line can be placed at a known position, e.g. $y = 0$. The reason for doing this is to see how precisely the pose of this line can be determined by using only the camera on the robot. By placing the robot at different positions and orientations around the line, it is easy to verify whether the mapping using computer vision is correct. The line is bright green, making it very easy to identify, and since there are no outliers a simple least squares method is used to fit the line. Pictures from the simulator are shown in Figure 6.3.

Table 6.1 shows the robot's pose in global *odom* coordinates and the angle and distance error of the estimated line's pose. The line is defined as $y = 0$ and the estimated line is represented as a point and a direction. The distance error was defined as the y-component of the point and the angle error is equal to the direction. The transform from robot's *base_link* to global *odom* coordinates is exact in this simulation, so the test will reveal errors in image to *base_link* transform. It can be concluded that the mapping works well, with no more than a few centimeters and degrees error in the tests.

TABLE 6.1: Test results to verify the mapping from image to global coordinates.

Robot pose, (x, y, ψ)	**Angle error [degrees]**	**Distance error [cm]**
$(0, 0, 0)$	0.8°	0.8
$(0, 0.5, 0)$	0.2°	7.4
$(0, 2.0, \frac{\pi}{4})$	1.4°	5.3

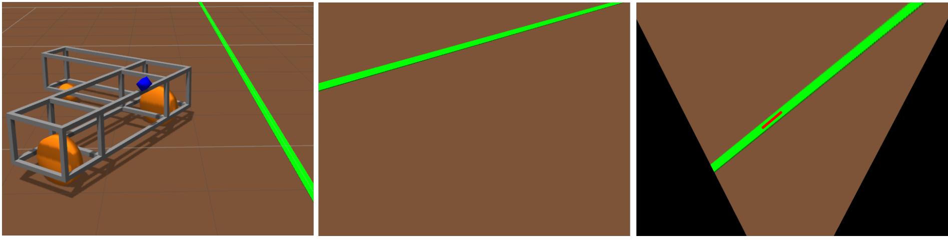


FIGURE 6.3: A simulation to test the mapping of an identified line in an image to global coordinates. Robot pose is $(x, y, \psi) = (0, 2.0, \pi/4)$. The left picture shows the robot's orientation to the line, the middle image shows the raw camera input and the right image shows the input after projective transformation. The small red line is the line identified using least squares method.

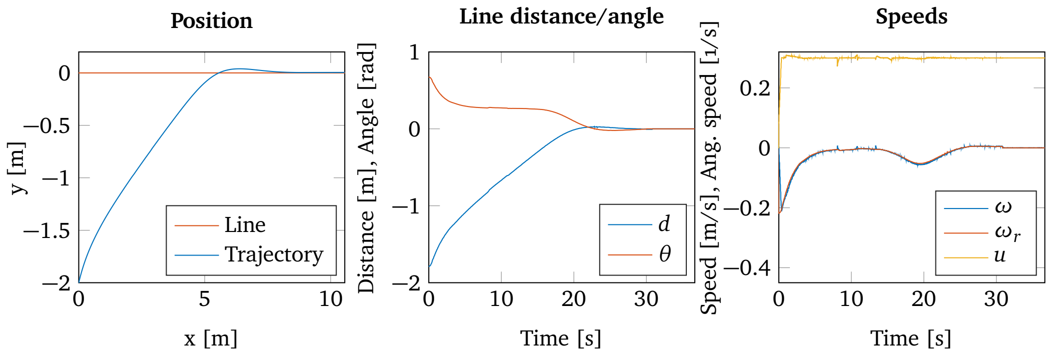


FIGURE 6.4: Simulation of the row controller with camera used for line identification. Robot starting point $(x, y, \psi) = (0, -2, \pi/4)$ and a line defined in global coordinates as $y = 0$.

6.2.2 *Following a Single Line with Camera*

In the previous subsection it was verified that the identification and mapping of a line using the camera worked well. The next simulation supplies the line following controller with the line estimated from the camera. The same green line as in Figure 6.3 and the simple least squares method for line fitting was used for this test. The results are presented in Figure 6.4. The results are very similar to the results obtained when only using the row controller without the camera. Note that the plotted d and θ are calculated from the estimated line, so they don't represent the actual distance and angle error as in simulations without camera. The position plot is same as before and represents the actual position of line and robot.

6.3 IDENTIFYING AND FOLLOWING ROW CROPS

So far the simulations verify that the row controller and line identification and mapping for a single line works well. The next step is to combine this with method 1 and 2 developed in Section 5.5 and Section 5.5.1 to follow the middle of three rows. The configuration that will be used to test both methods is shown in Figure 6.5. Real pictures of plants and rows have been put together to form a fairly realistic representation of a field.

FIGURE 6.5: Simulation of row detection, estimation and following. The robot starts in the position shown in the left image and attempts to follow the middle row. The middle image is the raw camera input, and the right image is the projected image with identified lines.

6.3.1 *Method 1*

Method 1 uses only Hough transform to identify the rows. Figure 6.6 shows the performance. The system works well, and the robot follows the middle row with only very small deviations. However, the estimated d and θ vary quite a lot, which causes ω to become less smooth. The reason for these fluctuations are the problems discussed earlier with the Hough lines method returning "random" lines within the middle row, i.e. every image processed will return a somewhat different line.

6.3.2 *Method 2*

Method 2 uses the output line of method 1 and improves it using a least squares line fitting. Figure 6.7 shows the performance. The performance is similar to method 1, but d and θ are much smoother now, which leads to smoother ω and smoother operation overall. Note that method 2 has a trajectory slightly above method 1. The difference is only about one centimeter, and it is hard to

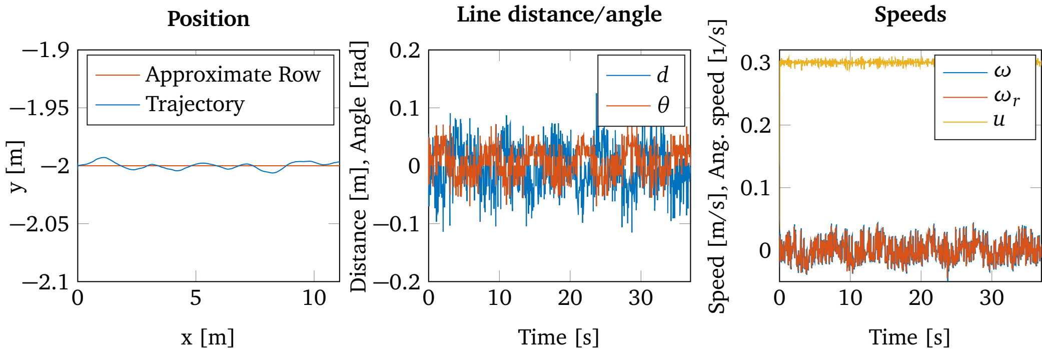


FIGURE 6.6: Simulation of row detection, estimation and following using method 1. The robot starts at $(x, y, \psi) = (0, -2, 0)$ and attempts to follow the middle row that is located at approximately $y = -2$.

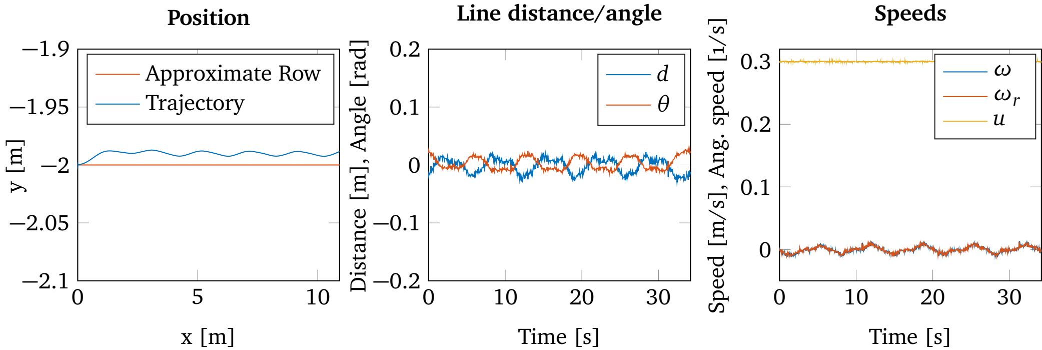


FIGURE 6.7: Simulation of row detection, estimation and following using method 2. The robot starts at $(x, y, \psi) = (0, -2, 0)$ and attempts to follow the middle row that is located at approximately $y = -2$.

tell which one is more correct due to the "random" nature of a row crop. The simulated crop row itself is about 15 centimeters wide.

6.4 ROW ESTIMATION WITH KALMAN FILTER

The final component of the row estimation and following is the Kalman filter to estimate row pose. The purpose of the Kalman filter is not only to smooth the measurements and thereby provide smoother operation, but also to make the system less vulnerable to false measurements. If e.g. a few very wrong row poses are provided directly to the controller, this could be enough to make the robot start turning sharply and go off course. If instead these measurements

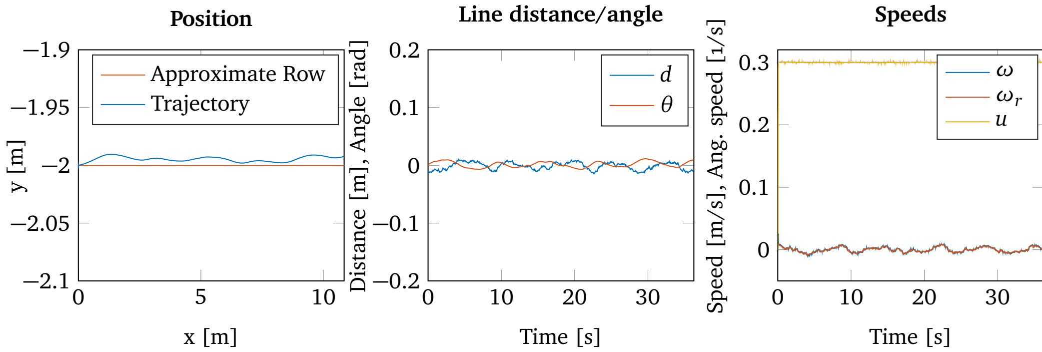


FIGURE 6.8: Simulation of row detection, estimation and following using method 1 with Kalman filter for row estimation. The robot starts at $(x, y, \psi) = (0, -2, 0)$ and attempts to follow the middle row that is located at approximately $y = -2$.

were provided to a filter, the outcome would hopefully be less dramatic and only cause a small deviation.

6.4.1 *Method 1*

As seen in Figure 6.6 method 1 produces fairly noisy row pose measurements, so hopefully a Kalman filter will be able to smooth it. One of the challenges with a Kalman filter is estimating the process and measurement noise covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. In this case the values were found by simply testing a few different values and then adjusting accordingly. It was found that small values for process noise covariance and large values for measurement noise covariance worked well. Figure 6.8 shows the performance using the following values:

$$\boldsymbol{Q} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 10000 \end{bmatrix} \tag{6.2}$$

The results are very promising. Both d and θ are much smoother which also leads to smoother operation. In fact, method 1 with Kalman filter performs equally or even better than method 2 without Kalman filter while following a straight row.

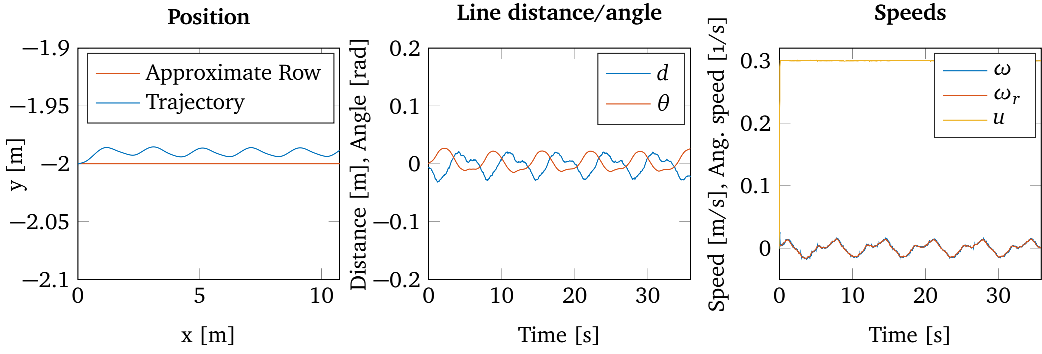


FIGURE 6.9: Simulation of row detection, estimation and following using method 2 with Kalman filter for row estimation. The robot starts at $(x, y, \psi) = (0, -2, 0)$ and attempts to follow the middle row that is located at approximately $y = -2$.

6.4.2 *Method 2*

Method 2 has much less noise than method 1 to begin with, which suggest that it might be wise to reduce the values of R a bit. Figure 6.9 shows the performance of method 2 with Kalman filter using the following values:

$$\boldsymbol{Q} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \tag{6.3}$$

The performance is overall good, even though $\boldsymbol{R}$ has been drastically reduced compared with method 1. There appears to be some oscillations that are slightly amplified compared to method 2 without Kalman filter. The oscillations are probably due to the fact that each row consists of seven repeated pictures, and the center of each row moves slightly. Method 1 is not affected much by this because it does not attempt to find the center in the same way as method 2. Increasing $\boldsymbol{R}$ significantly did not appear to amplify the oscillations any further.

6.5 FULL SYSTEM

Finally, all the parts of the system were combined and the robot was set to traverse the entire field. The robot started in the lower left part of the field as shown in Figure 5.4. Following is the expected behavior which should be considered while referencing Section 5.7 and Figure 5.17:

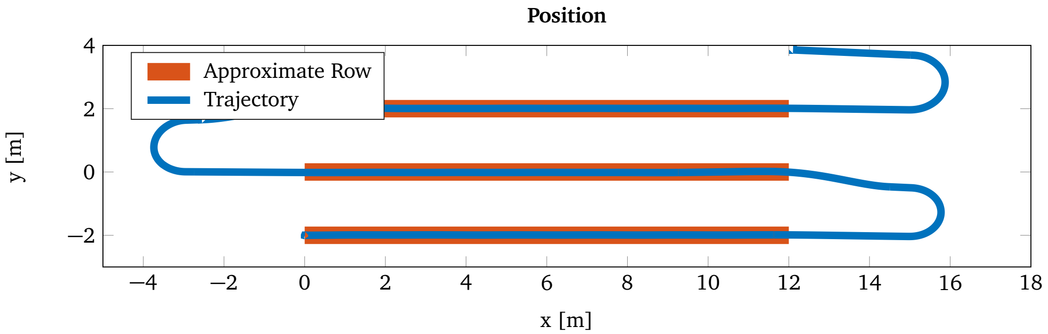


FIGURE 6.10: Simulation of full system using method 2 with Kalman filter for row estimation. The robot starts at $(x, y, \psi) = (0, -2, 0)$.

- The robot starts in state 1. Once enough row pose updates are received it should change to state 2 and start following the row.
- When it reaches the end it should exit the row and change to state 3 to turn left.
- After completed turn it should start going straight and look for the next row. If it detects the new row, change to state 1.
- Gather enough row pose updates in state 1 before changing to state 2 and continue row following.
- After two more successful turns, it should not find any more rows and shutdown after idling in state 1 for some time.

For this simulation it is not that relevant which method is used for row estimation, since all of them provide sufficient performance. The preferred configuration was method 2 with Kalman filter, so this was used for full system tests. Figure 6.10 shows the trajectory of the simulated robot. The system worked exactly as anticipated and followed all the state transitions as described above. A plot of the state transitions will be shown later for tests on the real robot in Figure 7.10. A video of this run is available[1].

[1] https://drive.google.com/open?id=0B0vkABY5y-eqWTNVMWpWMkh2T0U&authuser=0

7 LAB TESTS

This chapter contains a selection of tests that were performed on the Dogmatix robot. All the tests that were run in the simulator were also run on Dogmatix. All tests with Dogmatix were done indoors, which meant that there were no plants available. Also, the rear caster wheel of Dogmatix is centered, which means that it would destroy the middle row. For that reason, red and white striped cordoning bands were used as a replacement for the plants. The bright red color is easily identified.

7.1 LINE FOLLOWING CONTROLLER

The line following controller (3.11) was tested by defining a line in global coordinates and letting the robot follow it at constant forward speed. The same controller gains were used for the real robot as for the simulator. The values are repeated here for reference:

$$k_1 = 2.0, \;\; k_2 = 0.26, \;\; k_3 = 5.0 \tag{7.1}$$

It might seem surprising that there is no need for tuning when changing from simulator to a real robot. However, recall that the control law is only based on kinematics, so as long as the assumption of $\omega_r \approx \omega$ holds, there should not be any real need for tuning.

The performance is shown in Figure 7.1. The robot behaves almost exactly like the simulation results in Figure 6.1. The only difference is slightly more noisy speed measurements acquired from wheel odometry.

7.2 IDENTIFYING AND FOLLOWING A SINGLE LINE

In this test the goal is to verify the identification of a line using the camera mounted on Dogmatix, map the line in global coordinates and use the line following controller to follow it.

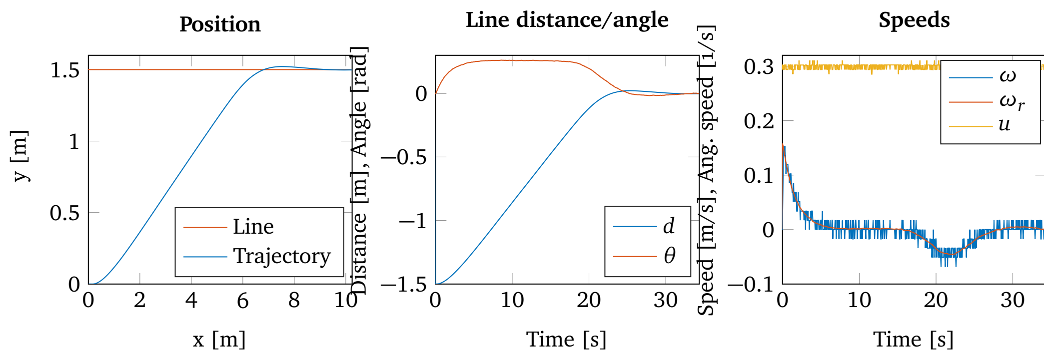


FIGURE 7.1: Real run on Dogmatix with the row controller, robot starting point $(x, y, \psi) = (0, 0, 0)$ and a line defined in global coordinates as $y = 1.5$. Notice how the approaching angle θ stabilizes at $\theta = \sin^{-1}(k_2) \approx 0.262 = 15°$ while approaching the line.

7.2.1 *Estimating Line Pose*

The Dogmatix robot was placed in different positions around a straight line made up of red and white striped cordoning band. The band was attempted placed at $y = 0$ in global *odom* coordinates and angle and distance error was found by comparison with the estimated line found by the robot. Figure 7.2 includes images of the setup and Table 7.1 shows the results. Note that the accuracy used in this table does not represent the actual accuracy of the experiment, it is merely the numbers reported by the robot. It is hard to assess the accuracy of such a setup, but it can be seen that the performance is good and the mapping appears to work well. The performance is especially good around $(x, y, \psi) = (0, 0, 0)$, which is most important since that is the normal operating area during row following.

TABLE 7.1: Test results to verify the mapping from image to global coordinates.

Robot pose, (x, y, ψ)	**Angle error [degrees]**	**Distance error [cm]**
$(0, 0, 0)$	3.3°	1.7
$(0, 0.4, 0)$	1.8°	−4.9
$(0, -1.0, \frac{\pi}{4})$	2.0°	17.2

Figure 7.2: Test with Dogmatix to ensure the mapping of an identified line is accurate. Robot pose is $(x, y, \psi) \approx (0, -1.0, \pi/4)$. The left picture shows the robot's orientation to the line, the middle image shows the raw camera input, and the right image shows the input after projective transformation with the identified line found using least squares method.

7.2.2 *Following a Single Line with Camera*

In the previous subsection it was verified that the identification and mapping of a line using the camera worked well. In the next test the identified line pose is passed on to the line following controller to follow. The same line as shown in Figure 7.2 and the simple least squares method for line fitting was used for this test. The results are presented in Figure 7.3. The results are very similar to the results obtained when only using the row controller without the camera. Note that the plotted d and θ are calculated from the estimated line, so they don't represent the actual distance and angle error as in tests without camera. The position plot is based on integration of wheel odometry data, so some drift will occur over time. Note that the approximate line added to the plot represents the cordoning band and is very approximate.

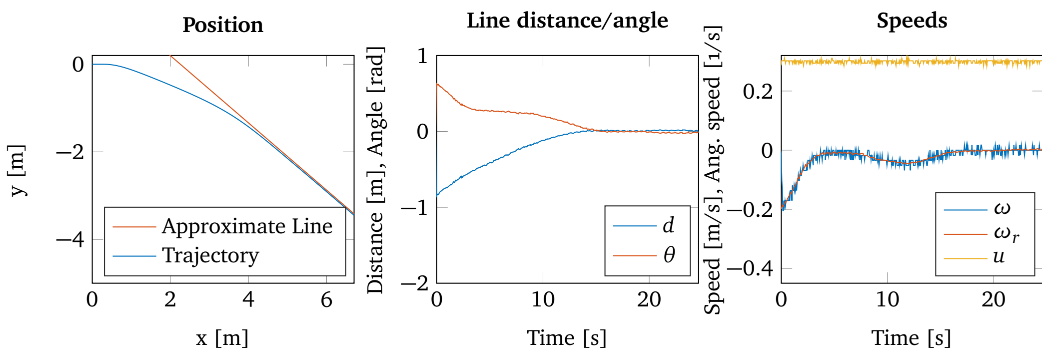


Figure 7.3: Test with Dogmatix with line following controller and camera used for line identification. Robot starting point $(x, y, \psi) = (0, 0, 0)$.

7.3 IDENTIFYING AND FOLLOWING ROW CROPS

The indoor lab did not include any row crops facilities. Instead, three lines of cordoning bands were placed next to each other as a replacement for three rows of crops. This is shown in Figure 7.4. The lines were attempted placed in a straight line such that the middle line corresponds to a line $y = 0$ in global *odom* coordinates, but it is difficult to lay it out completely straight. When the robot starts the position will always be $(x, y, \psi) = (0, 0, 0)$, so the robot has to be totally parallel to the line when starting for it to approximate $y = 0$. Therefore, the approximate lines in the plots in this section will deviate quite a bit from $y = 0$.

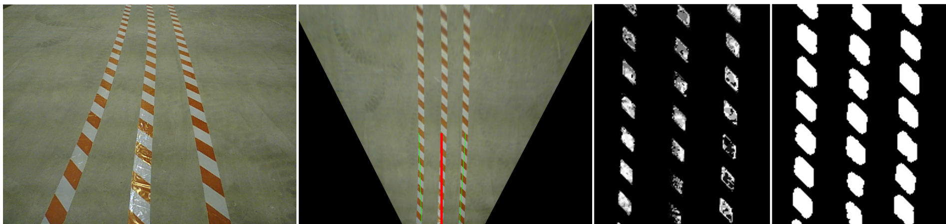


FIGURE 7.4: Three lines of cordoning bands replace the row crops. Left image is raw camera input, next image is input image after projective transformation with middle line identified using method 1, next image is probability image after backprojection and right image is the binary mask used for line identification.

7.3.1 *Method 1*

Method 1 uses only Hough transform to identify the rows. Figure 7.5 shows the performance. The conclusion is pretty much exactly the same as for the simulation. The system works well, and the robot follows the middle row with only very small deviations. However, as with the simulation, the estimated d and θ vary quite a lot, which causes ω to become less smooth.

7.3.2 *Method 2*

Method 2 uses the output line of method 1 and improves it using a least squares line fitting. Figure 7.6 shows the performance. Once more the results from Dogmatix are very similar to the simulation in Figure 6.7 and the same conclusion

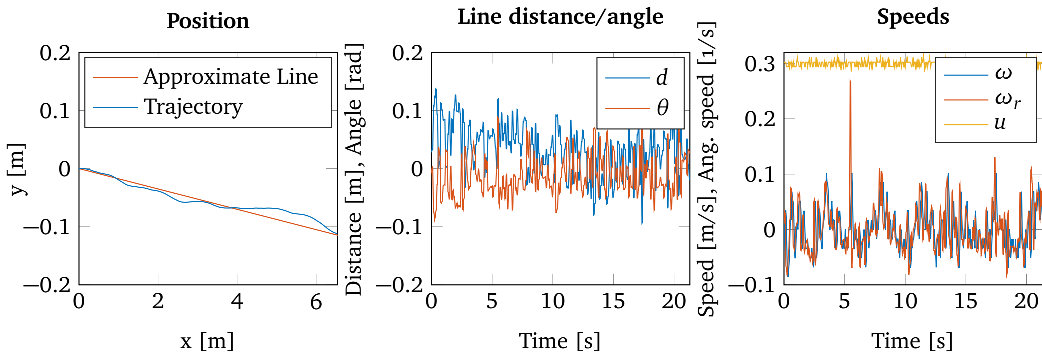


FIGURE 7.5: Real run on Dogmatix with row detection, estimation and following using method 1. The robot starts at $(x, y, \psi) = (0, 0, 0)$ and attempts to follow the middle row.

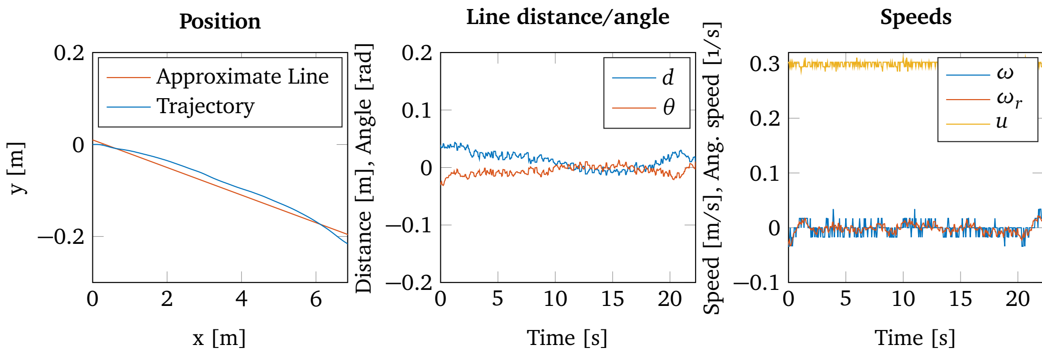


FIGURE 7.6: Real run on Dogmatix with row detection, estimation and following using method 2. The robot starts at $(x, y, \psi) = (0, 0, 0)$ and attempts to follow the middle row.

can be drawn: Smoother d and θ than method 1, which leads to smoother ω and smoother operation overall.

7.4 ROW ESTIMATION WITH KALMAN FILTER

The final component of the row estimation and following is the Kalman filter to estimate row pose. The output of method 1 or 2 is used as measurements in the Kalman filter. The pose estimates from the Kalman filter is then used to calculate d and θ which are used in the controller. This section uses the same setup as in Section 7.3 with the only addition being the Kalman filter.

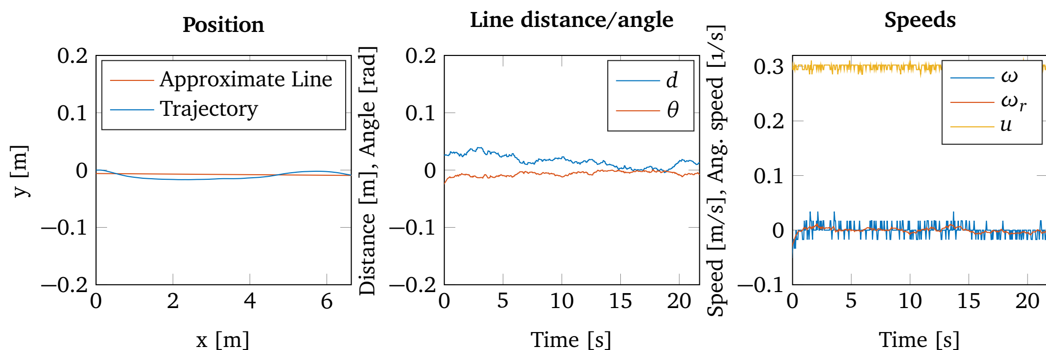


FIGURE 7.7: Real run on Dogmatix with row detection, estimation and following using method 1 with Kalman filter for row estimation. The robot starts at $(x, y, \psi) = (0, 0, 0)$ and attempts to follow the middle row.

7.4.1 *Method 1*

Method 1 had fairly noisy values of d and θ, so there is some room for improvement. Values for process and measurement noise covariances that were found to work well are

$$\boldsymbol{Q} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \tag{7.2}$$

The performance is shown in Figure 7.7. There is a big improvement from Figure 7.5, exactly like with the simulations. Much smoother d, θ and ω, and the trajectory is also a bit smoother.

7.4.2 *Method 2*

Method 2 already provided much smoother measurements than method 1, but the Kalman filter still might be able to improve the estimate. The same values for $\boldsymbol{Q}$ and $\boldsymbol{R}$ were used for method 2, even though the measurements were much less noisy:

$$\boldsymbol{Q} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \tag{7.3}$$

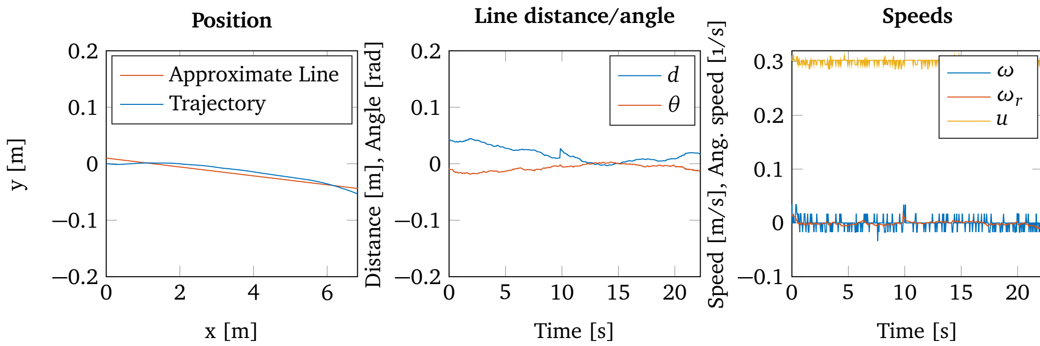


FIGURE 7.8: Real run on Dogmatix with row detection, estimation and following using method 2 with Kalman filter for row estimation. The robot starts at $(x, y, \psi) = (0, 0, 0)$ and attempts to follow the middle row.

The results are shown in Figure 7.8. The performance is very similar to the tests without Kalman filter, but once again d and θ are slightly smoother which also leads to smoother ω.

7.5 FULL SYSTEM

Finally, all the elements are put together to test the full system. The "field" that was used is the same as for the simulation: Three rows where each row consists of three smaller rows, and each small row is a line of cordoning band. A picture of the field with Dogmatix in operation is shown in Figure 7.9. The expected behavior is the same as for the simulation, which was described in Section 6.5.

A plot of the trajectory and the state the robot is in is included in Figure 7.10, and a video which is described in Appendix A is also included[1]. The robot followed the rows until the entire field was covered, and the state transitions are exactly as expected. The system was found to be very robust. Several tests with different row spacings were also performed, and the robot had no problems detecting the next row after completing the turn. In cases where it did not identify any rows after completing a turn, it always managed to find it after expanding the field of view of the camera.

[1] https://drive.google.com/open?id=0B0vkABY5y-eqNkZnUjkwb3M3Vms&authuser=0

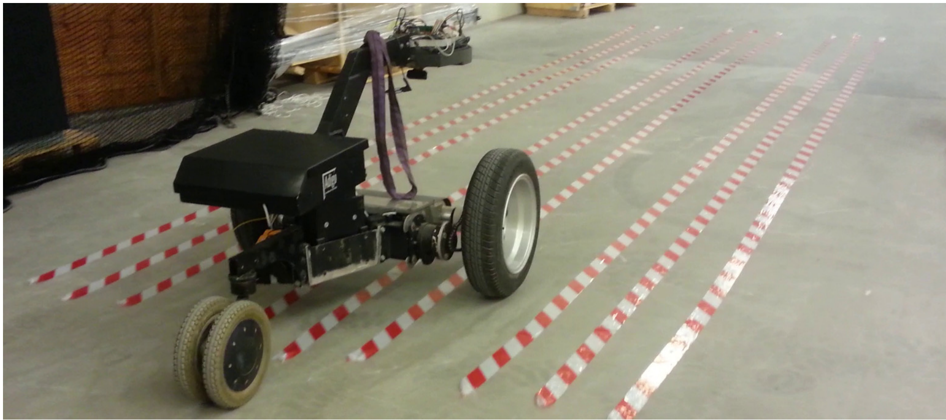


FIGURE 7.9: Full system test with Dogmatix.

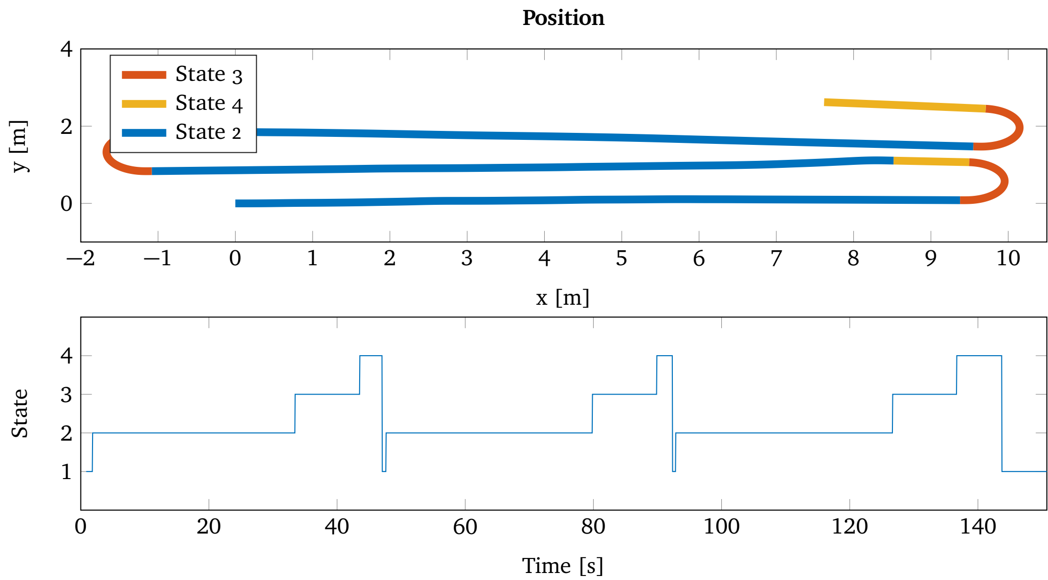


FIGURE 7.10: Real run on Dogmatix with full system using method 2 and Kalman filter for row estimation. The robot starts at $(x, y, \psi) = (0, 0, 0)$ and starts following the row. The upper plot shows the trajectory with color coding for different states, and the bottom plot shows which state the robot is in at different times.

8 FIELD TESTS

A trip to a field with carrot crops in Rygge, Østfold was conducted in early May. The purpose of the trip was to collect image data and test the row following controller and turning strategy at the end of a row. Unfortunately, the carrots were in a very early stage, which meant that the plants were very small and hard to see. Figure 8.1 shows the difference between early and late stages of the growing process. The plants were too small to be spotted after the usual downscaling and perspective transformation that were applied during lab tests, so different techniques had to be considered during the few hours of field testing.

Figure 8.1: The same carrot field in Rygge, Østfold during different stages of the growing process of carrot crops. Left picture is taken in June/July 2014 and right picture is taken during this field test in early May 2015.

8.1 IMAGE ANALYSIS

A closeup of a row is shown in Figure 8.2. The tiny plants were used to create the hue and saturation histogram used for backprojection. From a distance it was very difficult to spot the plants.

The first thing that was done was to increase the resolution of the image from 640×480 to 2448×1920, more than 15 times pixel count increase. This allows for greater details at a cost of lower frame rate and longer processing time,

but this alone was not sufficient to fix the problem. The next strategy was to drop the perspective transform of the image to ensure no details are lost in the transformation. By doing this the identified lines will not be parallel due to the perspective of the camera, but that can be overcome by applying the same homography transform to the line coordinates instead of the image.

There was some success with this technique, but it was not robust enough to perform any row following. Figure 8.3 shows a successful identification of the rows. However, by looking at the binary mask it should be obvious that it is not very robust. If anything it shows how well the Hough line method works. While the robot was moving it was very difficult to get robust and consistent identification, so it was not good enough to run any tests with the line following controller. As a result, no successful runs of the row controller with real plants were done with the robot during this field test.

FIGURE 8.2: Left image is a closeup of the plants that were used to calculate a histogram of hue and saturation values that could be used to identify plants. Right image shows the probability image after backprojecting the left image.

8.2 TURNING AT END OF ROW

A test of the identification and turning strategy at the end of a row was conducted by adding red and white striped cordoning bands in place of the plants in each row to provide sufficient visual aid. This should be similar to the amount of plants that can be expected a few weeks later in the growing process. The test turned out to be a complete success. A picture of the robot while turning is shown in Figure 8.4. The only slight problem that occurred was some wheel slip, but not enough to cause any real problems. At the time the tests were performed, only wheel encoders were used to estimate velocities. If wheel encoder

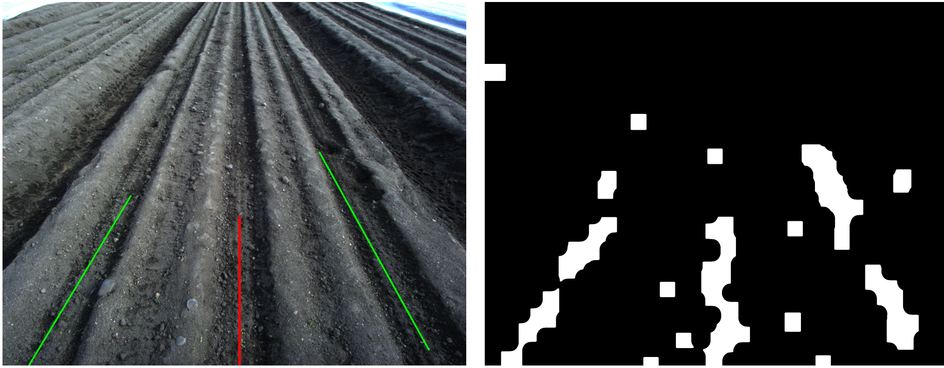


FIGURE 8.3: Left image shows the raw input image with identified rows using method 1. Right image shows the binary mask that was used for the identification after backprojection, thresholding pixels with high probability of being a plant followed by heavy dilation and erosion. There are not enough plants to ensure robust and consistent identification of rows.

measurements are combined with IMU measurements in a filter, the effects of wheel spin should be negligible. A video of this run is available[1].

8.3 RECREATING FIELD IN SIMULATOR

Since we only spent a few hours in field, there was not enough time to attempt new computer vision techniques for identifying rows with little to no plants. Therefore, the field was recreated in the simulator by patching together pictures taken that day. Figure 8.5 shows a comparison of the real field and the simulated field. The simulated field is quite similar to the real field, but it lacks depth. However, when simply looking straight down the rows the simulation is convincing.

There are not enough plants in the field to reliable use that for row identification, so other features have to be used. There is very little color information available since everything is just soil, which means that the usual approach with identification using hue and saturation values will not work. A feature that can be used is the darker areas in each row. By first converting the image to a gray scale image, an adaptive threshold technique can be used to extract the darker areas. Adaptive threshold techniques were briefly introduced in Section 2.4.1.

[1] https://drive.google.com/open?id=0B0vkABY5y-eqNTdoaTI1d3ZhWDg&authuser=0

FIGURE 8.4: A test of the identification and turning strategy at the end of a row. The robot successfully identified the end, kept driving straight for a specified length to completely exit the row, turned around, identified next row and continued with following it.

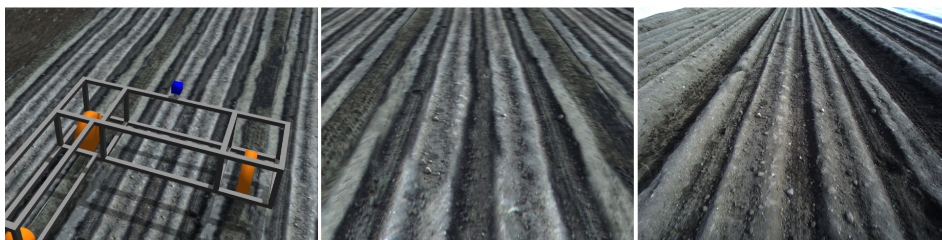


FIGURE 8.5: Left image shows the simulated robot placed in the simulated field, middle image is captured by the simulated robot, and right image is a real picture captured by Asterix.

In this case the areas we want to extract are the darker areas, so an inverse adaptive thresholding technique is applied:

$$\mathrm{dst}(x,y) = \begin{cases} 1 & \text{if } \mathrm{src}(x,y) < \alpha(x,y) + C_o \\ 0 & \text{else} \end{cases} \tag{8.1}$$

where $\alpha(x,y)$ is the adaptive threshold value and C_o is a constant offset parameter. A kernel with a specified size and weighting is used (recall what a kernel is from Section 2.4.2) to calculate $\alpha(x,y)$ based on surrounding pixels. The offset parameter can be adjusted to ensure decent performance in cases of large dark areas with little light. In such conditions the binary image might end up all white without the offset parameter. OpenCV has an implementation

named *adaptiveThreshold* which supports both mean and Gaussian weighted kernels and an offset parameter.

A very large mean weighted box sized kernel was found to give good results. A small kernel will adapt the threshold value to keep all small details in the image, but in this case we would like to extract relatively wide rows without the small details. The steps for finding a binary image of the rows can be summarized to:

Step 1: Perform projective transformation on the input image and convert it to a gray scale image.

Step 2: Use an adaptive thresholding method with a very large kernel. The kernel used is mean weighted.

Step 3: Apply dilation to close any gaps in the binary image and then erosion to remove outliers.

The output after these steps should be a binary image with only the three rows in white. After that the procedure is exactly like earlier and both method 1 or 2 can be used. Figure 8.6 shows the result using inverse adaptive thresholding and method 1 for identifying the middle row. It works very well, and a complete run over the entire field was performed with success. The trajectory of the simulated robot using method 2 with Kalman filter is shown in Figure 8.7. The performance is almost identical to the test in the field with plants that was shown in Figure 6.10. The only difference is that the rows were placed closer to each other this time to better reflect the actual field. A video of this run is available[2].

While the inverse adaptive thresholding technique works equally well in simulations as the hue and saturation backprojection technique, it is uncertain how robust it is in real fields. The rows are darker primarily because there are elevations on both sides of the rows which causes the elevations to be brighter. If the sun is lighting the ground from directly above, the rows might not appear darker than the elevations. Obviously, the system must be robust enough to handle direct sunlight, so in practice this technique might not provide sufficient robustness. More tests in real fields are needed to conclude anything.

[2] https://drive.google.com/open?id=0B0vkABY5y-eqeUdiRGIzZXBzMVU&authuser=0

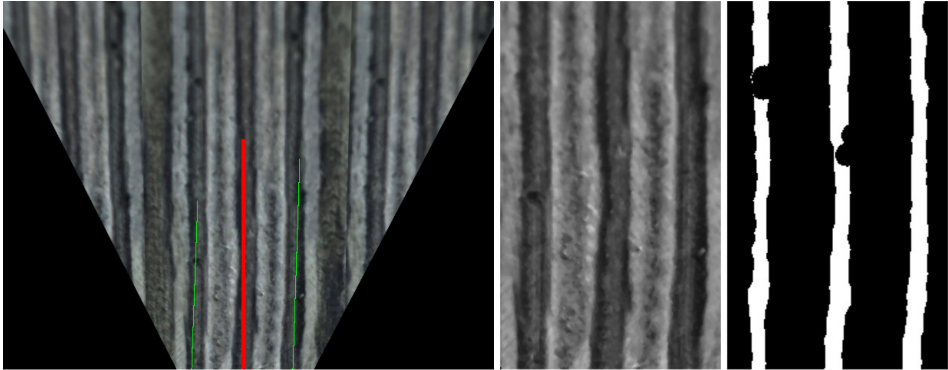


FIGURE 8.6: Left image shows input image after projective transformation and identified lines using method 1, middle image shows the cropped region that is used for inverse adaptive thresholding and right image shows the binary image after thresholding, dilation and erosion.

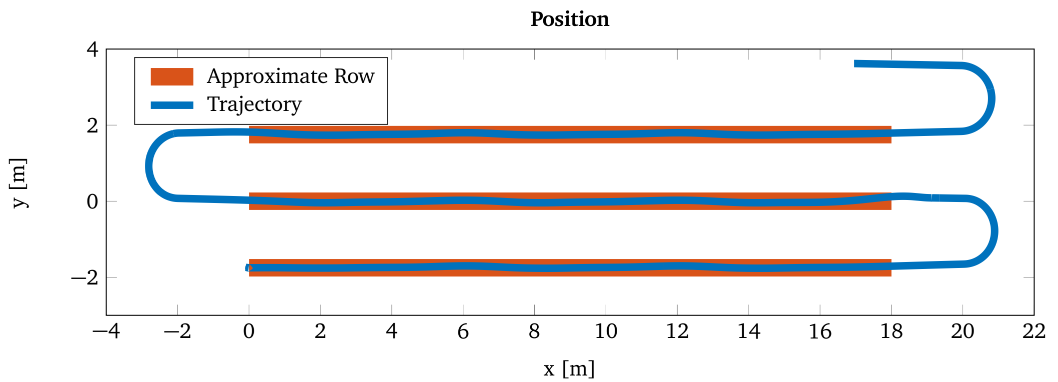


FIGURE 8.7: Simulation of full system using method 2 with Kalman filter for row estimation in the field without plants. The robot starts at $(x, y, \psi) = (0, -1.75, 0)$.

9 CONCLUDING REMARKS

A method for extracting plants from images based on hue and saturation values has been developed. The output of this method is a binary image. From the binary image, the rows are identified and the position and orientation (pose) of each row is estimated. The estimated pose is used as a measurement in a Kalman filter, whose output serves as input for a controller that follows the row.

All methods were tested on both the simulator, which was developed as part of the thesis, and real robots. The results from simulator and real robots are almost identical, suggesting that the simulator provides a realistic platform for testing and development.

A field test at a carrot field was conducted in early May with the Asterix robot. At this point in the growing process, the plants were not sufficiently large for the row estimation method to offer robust performance. The field conditions were recreated in the simulator, and a row identification method based on an inverse adaptive thresholding technique was developed. This method can identify rows in conditions without plants, but it remains to be seen how robust it will be on a real field.

The method developed for detecting the end of a row, turning around and identifying the next row has worked very well in the simulator and tests performed with red and white striped cordoning bands as replacement for plants in rows. It remains to be tested in rows with actual plants.

VIDEOS

Some videos have been included to demonstrate the system during operation. Following are the links and a short explanation of every video. Apologies for the long links, but they are clickable in the PDF:

- `https://drive.google.com/open?id=0B0vkABY5y-eqNkZnUjkwb3M3Vms&authuser=0`: Test of the full system on Dogmatix with an indoor field made of cordoning bands. This is the same run that was shown in Figure 7.10.
- `https://drive.google.com/open?id=0B0vkABY5y-eqNTdoaTI1d3ZhWDg&authuser=0`: Test of the end of row detection and turning controller on Asterix in a real field. Due to lack of plants in the field, some cordoning bands were placed in the rows to provide sufficient visual aid. A picture from this video was shown in Figure 8.4.
- `https://drive.google.com/open?id=0B0vkABY5y-eqWTNVMWpWMkh2T0U&authuser=0`: Simulation of full system on a field with plenty of green plants as shown in Figure 6.10. Top right image is raw camera input, the image below is the input after projective transformation with identified rows, next image is the binary mask, next image is the probability image after backprojection, and to the far left is the cropped image that is backprojected.
- `https://drive.google.com/open?id=0B0vkABY5y-eqeUdiRGIzZXBzMVU&authuser=0`: Simulation of full system on a field with almost no plants as shown in Figure 8.7, using inverse adaptive thresholding to find the darker rows. Top right image is raw camera input, the image below is the input after projective transformation with identified rows, next image is the binary mask, next image is a gray scale image before thresholding and to the far left is the cropped image. This row detection mode is not as robust as the one used for fields with plants, so at some points in the movie the wrong row is identified as middle row. However, because of the Kalman filter, a few false measurements does not cause it to steer completely off course.

B

PAPER

A paper documenting the work that was done last semester has been written while working on the Master's thesis. The paper was written by the first author, while second and third author contributed with good feedback. The paper has been submitted to the *19th International Conference on System Theory, Control and Computing* (ICSTCC 2015) in Cheile Gradistei, Romania, October 14th-16th 2015[1]. The paper is currently awaiting acceptance. The full paper follows in the next pages.

[1] `http://www.aie.ugal.ro/homeconf`

Experimental Comparison of Adaptive Controllers for Trajectory Tracking in Agricultural Robotics

Jarle Dørum, Trygve Utstumo, Jan Tommy Gravdahl
Department of Engineering Cybernetics
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway

***Abstract*—This paper describes the development of several controllers to handle a trajectory tracking problem for a differentially wheeled robot. Both simulations and tests on a real robot were performed. A simple kinematic controller has been implemented to calculate desired velocities based on current position and trajectory. In order to also consider the current velocities, i.e. the dynamics of the system, the output of this controller was used as input to a dynamic controller derived from a nonlinear model. The dynamic controller was made adaptive by using an on-line parameter estimation scheme to estimate the unknown parameters of the nonlinear model. Lastly, a direct model reference adaptive controller (MRAC) based on a linear model was derived and implemented as an alternative to the adaptive dynamic controller.**

Fig. 1. A picture of the prototype robot on a field test.

I. Introduction

This paper presents part of the ongoing research for developing an agricultural robot that autonomously navigates in row crops while identifying and precision spraying individual weed leaves with herbicide. The robot is a differentially steered robot with two rear mounted caster wheels, and may be modeled as a unicycle-like robot. A picture of the prototype during testing in row crops is shown in Fig. 1.

Previous research on the project includes development of a precision drop-on-demand nozzle for herbicide application [1], a model predictive row controller [2] to minimize potential crop damage during operation and attitude estimation in agricultural robotics [3].

The nozzle array presented in [1] is intended to only be slightly wider than the row crops, meaning that the robot has to follow the row crops precisely. A small offset could mean that the weed is out of reach for the nozzles, leaving the weed untreated. This motivates the research in this paper to find a trajectory tracking controller that minimizes the tracking error.

Another aspect to consider is changing physical properties of the robot. For example, the weight of the robot will change as herbicide and fuel is consumed. To ensure satisfactory performance at all times, several adaptive approaches that update the controller gains continuously have been tested.

Unicycle-like robots are used extensively in all kinds of fields and numerous models and controllers have been described in publications. In this paper a nonlinear model proposed in [4] has been used for simulations. The same model was also used in [5] to develop an adaptive dynamic controller, which has been implemented and tested here but with a different adaptation law. In [6] an adaptive controller using adaptive backstepping is presented. [7] developed a model reference adaptive controller (MRAC) for the tracking problem, but only simulations were performed. A similar direct MRAC has been derived here and implemented on the robot for testing.

The most important contribution of this paper is the comparison of two different adaptive controllers implemented on the same robot. The author is not aware of any previous implementations of the MRAC controller presented here on a real robot.

Different approaches to row crop guidance systems has been thoroughly explored and reviewed in [8]. However, this paper focuses merely on tracking a smooth and well defined trajectory without considering how to obtain the trajectory. The results obtained should be applicable to most unicycle-like robots.

II. Mathematical Model

The model used for simulations and some of the controller designs in this paper was presented in [4]. It is given as

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\\ \dot{u}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix} u\cos\psi - a\omega\sin\psi \\ u\sin\psi + a\omega\cos\psi \\ \omega \\ \frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}u \\ -\frac{\theta_5}{\theta_2}u\omega - \frac{\theta_6}{\theta_2}\omega \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2} \end{bmatrix} \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \\ 0 \\ \bar{\delta}_u \\ \bar{\delta}_\omega \end{bmatrix} \tag{1}$$

where $\boldsymbol{h} = \begin{bmatrix} x & y \end{bmatrix}^T$ is the position, ψ is the heading angle, u is forward velocity, ω is angular velocity and a is the distance from center of wheel axis to $\boldsymbol{h}$ as shown in Fig. 2. Motor inputs are given as velocities instead of torque values, which means that the motor controller is assumed to have a PID

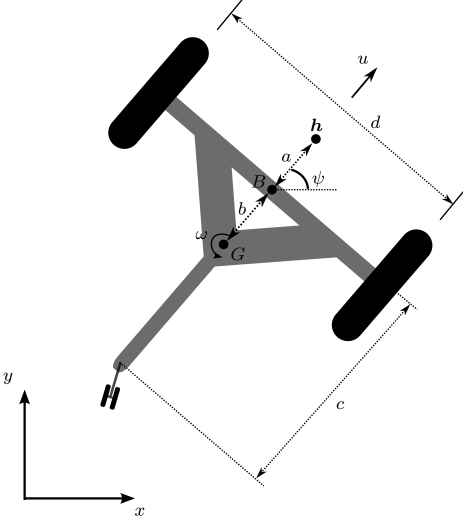


Fig. 2. Drawing of unicycle-like robot with centered rear mounted caster wheel, similar to the robot all tests were performed on.

controller or similar. $\boldsymbol{\theta}$ is a collection of physical parameters derived in [4] and included here for reference:

$$\begin{aligned}
\theta_1 &= \left(\frac{R_a}{k_a}(mR_t r + 2I_e) + 2rk_{DT}\right)/(2rk_{PT}) \\
\theta_2 &= \left(\frac{R_a}{k_a}(I_e d^2 + 2R_t r(I_z + mb^2)) + 2rdk_{DR}\right)/(2rdk_{PR}) \\
\theta_3 &= \frac{R_a}{k_a} mbR_t/(2k_{PT}) \\
\theta_4 &= \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right)/(rk_{PT}) + 1 \\
\theta_5 &= \frac{R_a}{k_a} mbR_t/(dk_{PR}) \\
\theta_6 &= \frac{R_a}{k_a}\left(\frac{k_a k_b}{R_a} + B_e\right) d/(2rk_{PR}) + 1
\end{aligned} \tag{2}$$

where R_a is motor resistance, k_a motor torque multiplied by gear ratio, k_b motor voltage multiplied by gear ratio, r wheel radius, I_e motor moment of inertia, B_e motor viscous friction coefficient, $k_{PT}, k_{DT}, k_{PR}, k_{DR}$ are PID motor controller gains, I_z moment of inertia about vertical axis at center of mass, m mass.

$$\boldsymbol{\delta} = \begin{bmatrix}\delta_x & \delta_y & 0 & \bar{\delta}_u & \bar{\delta}_\omega\end{bmatrix}^T \tag{3}$$

represent the uncertainties of the system caused by wheel slips and forces exerted by the caster wheel. For the purpose of this paper it has been assumed that $\boldsymbol{\delta} = 0$.

III. Controller Design

In many cases unicycle-like robots operate at low speeds and often inhibit low moment of inertia. In other words, the dynamics of u and ω are so fast that in many cases one may simplify $u \approx u_{ref}$, $\omega \approx \omega_{ref}$ and only study the kinematic model given by

$$\begin{bmatrix}\dot{x} \\ \dot{y} \\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}u_{ref}\cos\psi - a\omega_{ref}\sin\psi \\ u_{ref}\sin\psi + a\omega_{ref}\cos\psi \\ \omega_{ref}\end{bmatrix} \tag{4}$$

For larger robots operating at higher speeds the dynamics cannot simply be ignored. In the next sections, various controller designs are considered.

A. Trajectory Tracking Controller

Let $\boldsymbol{h_d}(t) = \begin{bmatrix}x_d(t) & y_d(t)\end{bmatrix}^T$ denote the time varying reference trajectory for the robot. The tracking error is defined as $\tilde{\boldsymbol{h}} = \begin{bmatrix}x_d - x & y_d - y\end{bmatrix}^T$. The kinematics from (1) may be written as

$$\begin{bmatrix}\dot{x} \\ \dot{y} \\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}\cos\psi & -a\sin\psi \\ \sin\psi & a\cos\psi \\ 0 & 1\end{bmatrix}\begin{bmatrix}u \\ \omega\end{bmatrix} \tag{5}$$

Recalling that $\boldsymbol{h} = [x \;\; y]^T$

$$\dot{h} = \begin{bmatrix}\dot{x} \\ \dot{y}\end{bmatrix} = \begin{bmatrix}\cos\psi & -a\sin\psi \\ \sin\psi & a\cos\psi\end{bmatrix}\begin{bmatrix}u \\ \omega\end{bmatrix} = \boldsymbol{A}\begin{bmatrix}u \\ \omega\end{bmatrix} \tag{6}$$

Multiplying (6) with the inverse of $\boldsymbol{A}$ gives

$$\begin{bmatrix}u \\ \omega\end{bmatrix} = \boldsymbol{A}^{-1}\begin{bmatrix}\dot{x} \\ \dot{y}\end{bmatrix} = \begin{bmatrix}\cos\psi & \sin\psi \\ -\frac{1}{a}\sin\psi & \frac{1}{a}\cos\psi\end{bmatrix}\begin{bmatrix}\dot{x} \\ \dot{y}\end{bmatrix} \tag{7}$$

A controller based on inverse kinematics is proposed in [5]:

$$\begin{bmatrix}u^c_{ref} \\ \omega^c_{ref}\end{bmatrix} = \begin{bmatrix}\cos\psi & \sin\psi \\ -\frac{1}{a}\sin\psi & \frac{1}{a}\cos\psi\end{bmatrix}\begin{bmatrix}\dot{x}_d + l_x\tanh\left(\frac{k_x}{l_x}\tilde{x}\right) \\ \dot{y}_d + l_y\tanh\left(\frac{k_y}{l_y}\tilde{y}\right)\end{bmatrix} \tag{8}$$

where $k_x, k_y > 0$ are controller gains and $l_x, l_y > 0$ are saturation constants. u^c_{ref} and ω^c_{ref} are desired forward and angular velocities, respectively. The controller is shown in [5] to have an asymptotically stable equilibrium at the origin $\tilde{\boldsymbol{h}} = \begin{bmatrix}0 & 0\end{bmatrix}^T$ under the assumption of $u = u^c_{ref}$ and $\omega = \omega^c_{ref}$. Note that the name trajectory tracking controller and kinematic controller both refer to the same controller for the rest of this paper.

B. Dynamic Controller

The kinematic controller will work adequately as long as the dynamics of the system are fast enough, i.e. the assumption of $u \approx u^c_{ref}$ and $\omega \approx \omega^c_{ref}$ is reasonable. In cases where the dynamics are too slow to be ignored or high precision tracking is required, the kinematic controller alone may no longer be sufficient.

Consider the dynamic part of (1)

$$\begin{bmatrix}\dot{u} \\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}\frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}u \\ -\frac{\theta_5}{\theta_2}u\omega - \frac{\theta_6}{\theta_2}\omega\end{bmatrix} + \begin{bmatrix}\frac{1}{\theta_1} & 0 \\ 0 & \frac{1}{\theta_2}\end{bmatrix}\begin{bmatrix}u_{ref} \\ \omega_{ref}\end{bmatrix} \tag{9}$$

Rearranging gives

$$\begin{bmatrix}u_{ref} \\ \omega_{ref}\end{bmatrix} = \begin{bmatrix}\theta_1\dot{u} - \theta_3\omega^2 + \theta_4 u \\ \theta_2\dot{\omega} + \theta_5 u\omega + \theta_6\omega\end{bmatrix} \tag{10}$$

Which may be written as

$$\begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} = \begin{bmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} 0 & 0 & -\omega^2 & u & 0 & 0 \\ 0 & 0 & 0 & 0 & u\omega & \omega \end{bmatrix} \boldsymbol{\theta} \tag{11}$$

Motivated by the inverse dynamics in (11), [5] proposes the controller given as

$$\begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix} = \begin{bmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -\omega^2 & u & 0 & 0 \\ 0 & 0 & 0 & 0 & u\omega & \omega \end{bmatrix} \boldsymbol{\theta} \tag{12}$$

where

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = \begin{bmatrix} \dot{u}^c_{ref} + k_1 \tilde{u} \\ \dot{\omega}^c_{ref} + k_2 \tilde{\omega} \end{bmatrix}, \quad \begin{matrix} \tilde{u} = u^c_{ref} - u \\ \tilde{\omega} = \omega^c_{ref} - \omega \end{matrix} \tag{13}$$

and $k_1, k_2 > 0$ are constant gains. In order to implement (12), the values of $\boldsymbol{\theta}$ must be known. Measuring or otherwise obtaining the parameters needed to calculate $\boldsymbol{\theta}$ may prove hard, thus the need to estimate $\boldsymbol{\theta}$ becomes a necessity. Replacing $\boldsymbol{\theta}$ with the estimate $\hat{\boldsymbol{\theta}}$ in (12) gives

$$\boldsymbol{\nu_{ref}} = \hat{\boldsymbol{D}}\boldsymbol{\sigma} + \boldsymbol{E}\hat{\boldsymbol{\theta}} \tag{14}$$

where

$$\boldsymbol{\nu_{ref}} = \begin{bmatrix} u_{ref} \\ \omega_{ref} \end{bmatrix}, \ \boldsymbol{E} = \begin{bmatrix} 0 & 0 & -\omega^2 & u & 0 & 0 \\ 0 & 0 & 0 & 0 & u\omega & \omega \end{bmatrix}, \quad \hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\theta}_1 & 0 \\ 0 & \hat{\theta}_2 \end{bmatrix}, \ \hat{\boldsymbol{\theta}} = \begin{bmatrix} \hat{\theta}_1 & \hat{\theta}_2 & \hat{\theta}_3 & \hat{\theta}_4 & \hat{\theta}_5 & \hat{\theta}_6 \end{bmatrix}^T \tag{15}$$

Following is a stability analysis similar to what was done in [5]. (11) may be written as

$$\boldsymbol{\nu_{ref}} = \boldsymbol{D}\dot{\boldsymbol{\nu}} + \boldsymbol{E}\boldsymbol{\theta} \tag{16}$$

Similarly, (14) is written

$$\boldsymbol{\nu_{ref}} = \hat{\boldsymbol{D}}\boldsymbol{\sigma} + \boldsymbol{E}\hat{\boldsymbol{\theta}} = \boldsymbol{G}\boldsymbol{\theta} - \boldsymbol{G}\tilde{\boldsymbol{\theta}} = \boldsymbol{D}\boldsymbol{\sigma} + \boldsymbol{E}\boldsymbol{\theta} - \boldsymbol{G}\tilde{\boldsymbol{\theta}} \tag{17}$$

where $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}$ and

$$\boldsymbol{G} = \begin{bmatrix} \sigma_1 & 0 & -\omega^2 & u & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & u\omega & \omega \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{bmatrix} \tag{18}$$

(13) may be written as

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = \dot{\boldsymbol{\nu}}^{\boldsymbol{c}}_{\boldsymbol{ref}} + \boldsymbol{K}\tilde{\boldsymbol{\nu}} \tag{19}$$

where $\boldsymbol{K} = \text{diag}(k_1, k_2)$. Combining (16), (17) and (19)

$$\boldsymbol{D}\dot{\boldsymbol{\nu}} + \boldsymbol{E}\boldsymbol{\theta} = \boldsymbol{D}\dot{\boldsymbol{\nu}}^{\boldsymbol{c}}_{\boldsymbol{ref}} + \boldsymbol{D}\boldsymbol{K}\tilde{\boldsymbol{\nu}} + \boldsymbol{E}\boldsymbol{\theta} - \boldsymbol{G}\tilde{\boldsymbol{\theta}} \tag{20}$$

$$\dot{\tilde{\boldsymbol{\nu}}} = -\boldsymbol{K}\tilde{\boldsymbol{\nu}} + \boldsymbol{D}^{-1}\boldsymbol{G}\tilde{\boldsymbol{\theta}} \tag{21}$$

where $\dot{\tilde{\boldsymbol{\nu}}} = \dot{\boldsymbol{\nu}}^{\boldsymbol{c}}_{\boldsymbol{ref}} - \dot{\boldsymbol{\nu}}$ describes the error dynamics of the system. For this analysis, $\boldsymbol{\theta}$ is considered known, i.e. $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$, reducing (21) to

$$\dot{\tilde{\boldsymbol{\nu}}} = -\boldsymbol{K}\tilde{\boldsymbol{\nu}} \tag{22}$$

Consider the following Lyapunov-like function

$$V = \frac{1}{2}\tilde{\boldsymbol{\nu}}^T\boldsymbol{P}\tilde{\boldsymbol{\nu}} \tag{23}$$

where $\boldsymbol{P} = \boldsymbol{P}^T > \boldsymbol{0}$. Differentiating (23) along the solution of (22) gives

$$\dot{V} = -\tilde{\boldsymbol{\nu}}^T\boldsymbol{P}\boldsymbol{K}\tilde{\boldsymbol{\nu}} < 0 \ \forall \ \tilde{\boldsymbol{\nu}} \neq 0 \tag{24}$$

Which means that $\dot{V}$ is negative definite and global asymptotic stability can be concluded.

C. *On-line Parameter Estimation*

The dynamic controller given by (14) needs a good estimate $\hat{\boldsymbol{\theta}}$ in order to perform well. One approach to estimate $\hat{\boldsymbol{\theta}}$ is to log a test run with sufficiently excited input signal and use an off-line system identification technique, e.g. least-squares method. Another approach is to estimate $\hat{\boldsymbol{\theta}}$ on-line using an adaptation law $\dot{\hat{\boldsymbol{\theta}}}$. This section shows the derivation of $\dot{\hat{\boldsymbol{\theta}}}$ using the gradient method, which is motivated by the minimization of a cost function.

Consider (10) written on the form

$$\boldsymbol{\nu_{ref}} = \boldsymbol{\varphi}^T\boldsymbol{\theta} = \begin{bmatrix} \dot{u} & 0 & -\omega^2 & u & 0 & 0 \\ 0 & \dot{\omega} & 0 & 0 & u\omega & \omega \end{bmatrix} \boldsymbol{\theta} \tag{25}$$

where $\boldsymbol{\nu_{ref}} = [u_{ref} \ \ \omega_{ref}]^T$. Filtering both sides gives

$$\frac{\boldsymbol{\nu_{ref}}}{\Lambda(s)} = \frac{\boldsymbol{\varphi}^T\boldsymbol{\theta}}{\Lambda(s)} = \frac{1}{\Lambda(s)} \begin{bmatrix} su & 0 & -\omega^2 & u & 0 & 0 \\ 0 & s\omega & 0 & 0 & u\omega & \omega \end{bmatrix} \boldsymbol{\theta} \tag{26}$$

Which may be written as the parametric model

$$\boldsymbol{z} = \boldsymbol{\Phi}^T\boldsymbol{\theta} \tag{27}$$

where $z = \frac{\boldsymbol{\nu_{ref}}}{\Lambda(s)}$, $\boldsymbol{\Phi}^T = \frac{\boldsymbol{\varphi}^T}{\Lambda(s)}$ and $\Lambda(s)$ is chosen to be a Hurwitz polynomial of degree one, e.g. $\Lambda(s) = s + 1$. Note that $\boldsymbol{z}$ and $\boldsymbol{\Phi}$ are available measurements, while $\boldsymbol{\theta}$ is unknown. An estimate of $\boldsymbol{z}$ denoted $\hat{\boldsymbol{z}}$ is generated as

$$\hat{\boldsymbol{z}} = \boldsymbol{\Phi}^T\hat{\boldsymbol{\theta}} \tag{28}$$

where $\hat{\boldsymbol{\theta}}$ is the currently best estimate of $\boldsymbol{\theta}$. A normalized estimation error is defined as

$$\boldsymbol{\epsilon} = (\boldsymbol{M}^T\boldsymbol{M})^{-1}(\boldsymbol{z} - \hat{\boldsymbol{z}}) = (\boldsymbol{M}^T\boldsymbol{M})^{-1}(\boldsymbol{z} - \boldsymbol{\Phi}^T\hat{\boldsymbol{\theta}}) \tag{29}$$

where $\boldsymbol{M}^T\boldsymbol{M} = \boldsymbol{I} + \boldsymbol{N_s}^T\boldsymbol{N_s}$ is a diagonal matrix that normalizes the estimation error, and $\boldsymbol{N_s}^T\boldsymbol{N_s}$ is another diagonal matrix for design of the normalized signal. The reason for this normalization is to ensure boundedness, i.e.

$$\boldsymbol{\Phi}\boldsymbol{M}^{-1} \in \mathcal{L}_\infty \tag{30}$$

If $\boldsymbol{\Phi} \in \mathcal{L}_\infty$, then $\boldsymbol{M} = \boldsymbol{I}$ is sufficient. If it is not, choosing

$$\boldsymbol{M}^T\boldsymbol{M} = \boldsymbol{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi} \tag{31}$$

will ensure (30) is satisfied [9, p. 172]. An instantaneous cost function $\boldsymbol{J}(\hat{\boldsymbol{\theta}})$ is defined as

$$\boldsymbol{J}(\hat{\boldsymbol{\theta}}) = \frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{M}^T\boldsymbol{M}\boldsymbol{\epsilon} = \frac{1}{2}(\boldsymbol{z} - \boldsymbol{\Phi}^T\hat{\boldsymbol{\theta}})^T(\boldsymbol{M}^T\boldsymbol{M})^{-1}(\boldsymbol{z} - \boldsymbol{\Phi}^T\hat{\boldsymbol{\theta}}) \tag{32}$$

The gradient of (32) is

$$\Delta\boldsymbol{J}(\hat{\boldsymbol{\theta}}) = -\boldsymbol{\Phi}(\boldsymbol{M}^T\boldsymbol{M})^{-1}(\boldsymbol{z} - \boldsymbol{\Phi}^T\hat{\boldsymbol{\theta}}) = -\boldsymbol{\Phi}\boldsymbol{\epsilon} \tag{33}$$

Motivated by this, the following adaptation law for generating $\hat{\boldsymbol{\theta}}(t)$ is proposed

$$\dot{\hat{\boldsymbol{\theta}}} = -\boldsymbol{\Gamma}\Delta\boldsymbol{J}(\hat{\boldsymbol{\theta}}) = \boldsymbol{\Gamma}\boldsymbol{\Phi}\boldsymbol{\epsilon} \tag{34}$$

where $\boldsymbol{\Gamma} = \boldsymbol{\Gamma}^T >$ is a diagonal gain matrix. According to [9, p. 175], (34) ensures that

1) $\hat{\boldsymbol{\theta}}, \boldsymbol{\epsilon} \in \mathcal{L}_\infty$

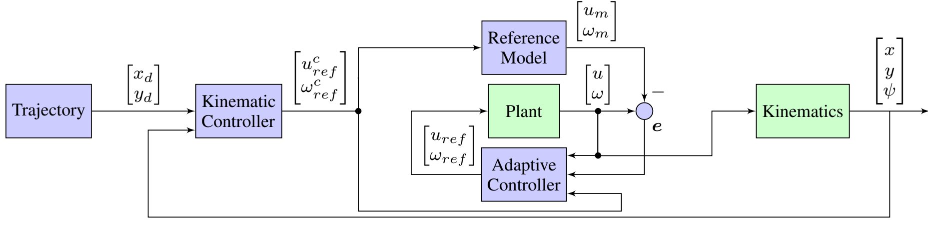


Fig. 3. Block diagram of the model reference adaptive controller.

2) $\boldsymbol{\epsilon}, \boldsymbol{N_s}^T\boldsymbol{\epsilon}, \dot{\hat{\boldsymbol{\theta}}} \in \mathcal{L}_\infty$

independent of the boundedness properties of $\boldsymbol{\Phi}$. In other words, both parameters and estimation errors should remain bounded. It does not, however, ensure that $\tilde{\boldsymbol{\theta}}(t) = \boldsymbol{\theta}(t) - \hat{\boldsymbol{\theta}}(t) \to 0$ as $t \to 0$. To ensure that the parameters $\hat{\boldsymbol{\theta}}$ do in fact converge to their actual value $\boldsymbol{\theta}$, $\boldsymbol{\Phi}$ must be persistently excited (PE), i.e., it satisfies [9, p. 254]

$$\alpha_1 \boldsymbol{I} \geq \frac{1}{T_0}\int_t^{t+T_0} \boldsymbol{\Phi}(\tau)\boldsymbol{\Phi}^T(\tau)\,\mathrm{d}\tau \geq \alpha_0 \boldsymbol{I}, \quad \forall t \geq 0 \tag{35}$$

for some $T_0, \alpha_0, \alpha_1 \geq 0$. It is in general difficult to show that $\boldsymbol{\Phi}$ is PE for an input signal $\boldsymbol{\nu_{ref}}$, and especially in a case like this where $\boldsymbol{\Phi}$ has some nonlinear elements.

D. Adaptive Dynamic Controller

The results from section III-B and section III-C may be combined to form an adaptive dynamic controller. The on-line parameter estimation operates independently from the dynamic controller and vice versa, making it a modular design. This may prove beneficial in cases where parameter estimation is only needed parts of the time, or if it is desirable to run parameter estimation without running the dynamic controller. The control laws are given by

$$\boldsymbol{\nu_{ref}} = \hat{\boldsymbol{D}}\boldsymbol{\sigma} + \boldsymbol{E}\hat{\boldsymbol{\theta}}, \quad \dot{\hat{\boldsymbol{\theta}}} = \boldsymbol{\Gamma}\boldsymbol{\Phi}\boldsymbol{\epsilon} \tag{36}$$

where the notation is the same as in section III-B and section III-C.

E. Direct Model Reference Adaptive Controller

In this section, a simple direct Model Reference Adaptive Controller (MRAC) scheme as shown in Fig. 3 is derived. The concept is to design a model of similar structure to the plant (robot), let the tracking reference be an input to the model, and make the output of the plant track the output of the model. For the direct MRAC approach, this is made possible by developing adaptation laws for the controller gains directly without having to identify actual system parameters.

Consider a simplified, linear model of the dynamics of (1) given by

$$\begin{bmatrix} \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} au + bu_{ref} \\ c\omega + d\omega_{ref} \end{bmatrix} \tag{37}$$

where a, b, c, d are unknown system parameters (not the same as those introduced in Fig. 2). In this case, u and ω are considered decoupled and will be analyzed separately. A reference model u_m for u is chosen to be

$$\dot{u}_m = -a_m u + b_m u^c_{ref} \tag{38}$$

Laplace transforming (37) and (38) gives

$$u = \frac{b}{s-a}u_{ref}, \quad u_m = \frac{b_m}{s+a_m}u^c_{ref} \tag{39}$$

The following control law is proposed

$$u_{ref} = -k^*_u u + l^*_u u^c_{ref} \tag{40}$$

Inserting (40) into (39) gives

$$u = \frac{bl^*_u}{s-a+bk^*_u}u^c_{ref}, \quad u_m = \frac{b_m}{s+a_m}u^c_{ref} \tag{41}$$

It is desirable to make the transfer functions of (41) equal. Choosing

$$l^*_u = \frac{b_m}{b}, \quad k^*_u = \frac{a+a_m}{b} \tag{42}$$

ensures equal transfer functions. However, it is not possible to implement since the values of a and b are unknown. Instead of using the control law (40), a control law using estimates of k^*_u and l^*_u is proposed

$$u_{ref} = -k_u(t)u + l_u(t)u^c_{ref} \tag{43}$$

where $k_u(t)$ and $l_u(t)$ are the currently best estimates of k^*_u and l^*_u, respectively. Adding and subtracting $b(-k^*_u u + l^*_u u^c_{ref})$ to $\dot{u}$ yields

$$\dot{u} = au + bu_{ref} + b(-k^*_u u + l^*_u r) - b(-k^*_u u + l^*_u r) \tag{44}$$

which, after combining with (42), may be written as

$$\dot{u} = -a_m u + b_m u^c_{ref} u + b(k^*_u u - l^*_u u^c_{ref} + u_{ref}) \tag{45}$$

Laplace transforming (45) gives

$$u = \underbrace{\frac{b_m}{s+a_m}u^c_{ref}}_{=u_m} + \frac{b}{s+a_m}(k^*_u u - l^*_u u^c_{ref} + u_{ref}) \tag{46}$$

Define the tracking error $e_u = u - u_m$ to obtain

$$e_u = \frac{b}{s+a_m}(k^*_u u - l^*_u u^c_{ref} + u_{ref}) \tag{47}$$

Since k^*_u and l^*_u are unknown, our best estimate of the tracking error $\hat{e}_u$ is

$$\hat{e}_u = \frac{b}{s+a_m}(k_u(t)u - l_u(t)u^c_{ref} + u_{ref}) \tag{48}$$

Inserting u_{ref} from (43) into (48) simply gives $\hat{e}_u = 0$, i.e. the estimated tracking error is zero. Note that the estimation error

$\epsilon_u = e_u - \hat{e}_u = e_u = u - u_m$ is equal to the tracking error e_u. Combining (43) and (47) while defining the gain parameter estimation errors $\tilde{k}_u(t) = k_u(t) - k_u^*$, $\tilde{l}_u(t) = l_u(t) - l_u^*$ gives

$$e_u = \frac{b}{s + a_m}(-\tilde{k}_u u + \tilde{l}_u u_{ref}^c) \tag{49}$$

$$\dot{e}_u = -a_m e_u + b(-\tilde{k}_u u + \tilde{l}_u u_{ref}^c) \tag{50}$$

Consider the Lyapunov-like function

$$V = \frac{1}{2}e_u^2 + \frac{|b|}{2\gamma_1}\tilde{k}_u^2 + \frac{|b|}{2\gamma_2}\tilde{l}_u^2 \tag{51}$$

with $\gamma_1, \gamma_2 > 0$. Differentiating (51) along the solution of (49) gives

$$\begin{aligned}\dot{V} = &- a_m e_u^2 + |b|\tilde{k}_u\left(-e_u u \,\mathrm{sgn}(b) + \frac{1}{\gamma_1}\dot{k}\right)\\ &+ |b|\tilde{l}_u\left(e_u u_{ref}^c \,\mathrm{sgn}(b) + \frac{1}{\gamma_2}\dot{l}\right)\end{aligned} \tag{52}$$

Choosing

$$\dot{k}_u = \gamma_1 e_u u \,\mathrm{sgn}(b), \quad \dot{l}_u = -\gamma_2 e_u u_{ref}^c \,\mathrm{sgn}(b) \tag{53}$$

ensures

$$\dot{V} = -a_m e_u^2 \leq 0 \tag{54}$$

Thus it is shown that $\dot{V}$ is negative semi definite, V has an upper bound $V(0)$ and bounded below by zero, i.e. $0 \leq V(t) \leq V(0)$. From the boundedness of $V(t)$ and (51), it is clear that $e_u, \tilde{k}_u, \tilde{l}_u \in \mathcal{L}_\infty$. u_{ref}^c, the output of the kinematic controller (8), is bounded, so $u_{ref}^c \in \mathcal{L}_\infty$. The transfer functions of (41) are in $\mathcal{L}_1$, and it follows from [9, p. 80] that $u, u_m \in \mathcal{L}_\infty$. This means that all signals of (49) are bounded and $\dot{e}_u \in \mathcal{L}_\infty$. From [9, p. 74] it follows that since $V(t)$ is bounded from below and non-increasing, it has a finite limit as $t \to \infty$, denoted V_∞. It can also be seen that

$$\begin{aligned}||e_u||_2 &= \left(\int_0^\infty e_u^2(\tau)\mathrm{d}\tau\right)^{1/2} = \left(\int_0^\infty -\frac{1}{a_m}\dot{V}(\tau)\mathrm{d}\tau\right)^{1/2}\\ &= \left(\frac{1}{a_m}(V(0) - V_\infty)\right)^{1/2}\end{aligned} \tag{55}$$

which is clearly bounded, so that $e_u \in \mathcal{L}_2$. Finally, [9, p. 80] shows that since $\dot{e}_u, e_u \in \mathcal{L}_\infty$ and $e_u \in \mathcal{L}_2$, then $e_u(t) \to 0$ as $t \to \infty$.

The results obtained show that the tracking objective of making the output of the plant $u(t)$ track the output of the reference model $u_m(t)$ is achieved. It does not, however, guarantee that $k_u(t), l_u(t) \to k_u^*, l_u^*$ as $t \to \infty$, i.e. the poles of the plant may differ from those of the reference model. This should be of less concern, since $k_u(t), l_u(t)$ are bounded and the true values of k_u^*, l_u^* are not of any real importance.

In a very similar manner the same results are found for ω. A summary of the control laws are given in Table I.

A modification that was done to provide for a more robust implementation was to add a small feedback loop to (53) to get

$$\dot{k}_u = \gamma_1 e_u u \,\mathrm{sgn}(b) - \alpha k_u, \quad \dot{l}_u = -\gamma_2 e_u u_{ref}^c \,\mathrm{sgn}(b) - \beta l_u \tag{56}$$

where $0 < \alpha << 1$ and $0 < \beta << 1$.

TABLE I. Control Laws for The MRAC

Plant	Reference Model	Control Law
$u = \frac{b}{s-a}u_{ref}$	$u_m = \frac{b_m}{s+a_m}u_{ref}^c$	$u_{ref} = -k_u(t)u + l_u(t)u_{ref}^c$, $\dot{k}_u = \gamma_1 e_u u \,\mathrm{sgn}(b)$ $\dot{l}_u = -\gamma_2 e_u u_{ref}^c \,\mathrm{sgn}(b)$ $e_u = u - u_m$
$\omega = \frac{d}{s-c}\omega_{ref}$	$\omega_m = \frac{d_m}{s+c_m}\omega_{ref}^c$	$\omega_{ref} = -k_\omega(t)\omega + l_\omega(t)\omega_{ref}^c$, $\dot{k}_\omega = \gamma_3 e_\omega \omega \,\mathrm{sgn}(d)$ $\dot{l}_\omega = -\gamma_4 e_\omega \omega_{ref}^c \,\mathrm{sgn}(d)$ $e_\omega = \omega - \omega_m$

IV. Simulations And Real Runs

Simulations were done using Matlab/Simulink, while real runs were performed on the robot shown in Fig. 4. The robot is running Robot Operating System (ROS) and all controllers were implemented in C++. The motor controller has a low level PID controller that uses individual motor velocities as setpoints, and motor acceleration can be saturated to ensure slower dynamics. Without limits on acceleration the dynamics were so fast that all controllers had equal performance. For both simulations and tests on the real robot the following figure eight trajectory was used:

$$\begin{aligned}x_d(t) &= r_e \sin(2\omega_e t)\\ y_d(t) &= r_e(\cos(\omega_e t) - 1)\end{aligned} \tag{57}$$

For the simulations $r_e = 1\,\mathrm{m}$ and $\omega_e = 0.3\,\mathrm{rad/s}$, while for the real runs the radius was slightly smaller, $r_e = 0.6\,\mathrm{m}$ and $\omega_e = 0.3\,\mathrm{rad/s}$. In all cases the distance from wheel axle to $h(x, y)$ was chosen to be $a = 0.10\,\mathrm{m}$. The on-line parameter estimation method was tested in simulations where $\boldsymbol{\theta}$ is known to ensure that the estimated $\hat{\boldsymbol{\theta}}$ converges to its actual value $\boldsymbol{\theta}$. It was found that $\hat{\boldsymbol{\theta}}$ does indeed converge correctly while attempting to track the figure eight, which means that the input signal is sufficiently excited to ensure convergence.

A comparison of all controllers is shown in Fig. 8 and the estimated parameters are shown in Fig. 5 and Fig. 6. Fig. 7 shows the controller inputs during a real run.

It is obvious from Fig. 8 (a) that the kinematic controller alone does not provide sufficient performance in this case. The dynamic controller shows good performance given accurate estimates $\hat{\boldsymbol{\theta}}$. The adaptive dynamic controller appears to be able to improve upon the performance of the dynamic controller as $\hat{\boldsymbol{\theta}}$ adapts (shown in Fig. 6).

In Fig. 8 (i) and (j) the motor acceleration saturation limit was increased to make the system a bit faster. It is interesting that the MRAC improves greatly when the dynamics are faster, while the adaptive dynamic controller has almost identical performance to the case with slower dynamics.

V. Conclusion

Two different adaptive dynamic controllers for tracking a trajectory were implemented on a differentially wheeled robot and compared with non-adaptive kinematic and dynamic controllers. The MRAC configuration, which the author has been

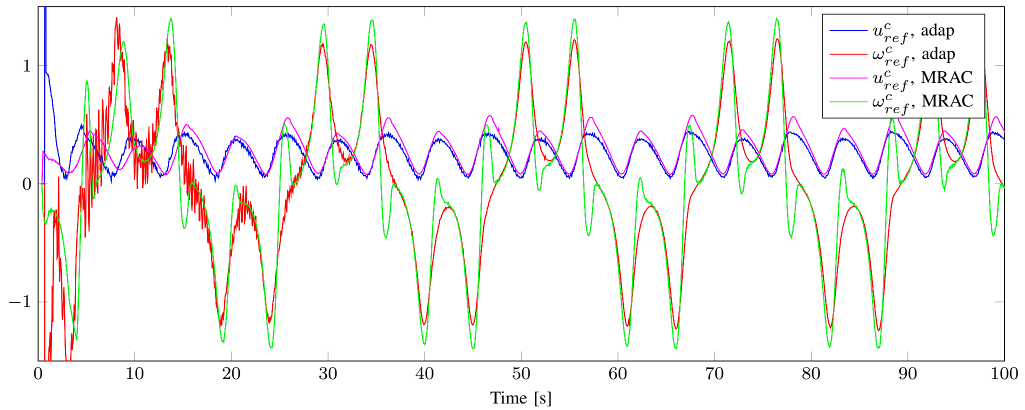


Fig. 7. Motor controller inputs from the MRAC and the adaptive dynamic controller during a test run with the slow dynamics. The inputs become smoother as the parameters adapt.

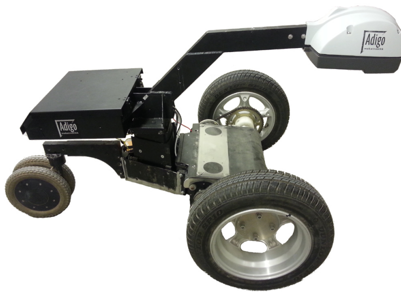


Fig. 4. A picture of the robot used for the tests.

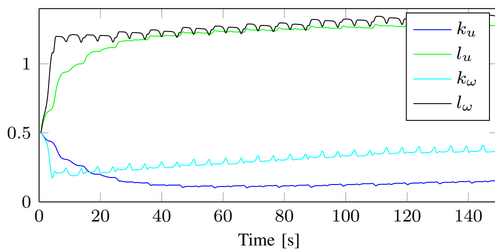


Fig. 5. Controller gains for the MRAC during a real run. All controller gains have initial values of 0.5.

unable to find previous papers presenting real implementations of, delivered the best performance of all controllers on a system with fairly slow dynamics, while the other adaptive dynamic controller had equal or better performance on a very slow system.

REFERENCES

[1] F. Urdal, T. Utstumo, S. A. Ellingsen, J. K. Vatne, and T. Gravdahl, "Design and control of precision drop-on-demand herbicide application in agricultural robotics," in *The 13th international Conference on Control, Automation, Robotics and Vision, Singapore*. IEEE, 2014.

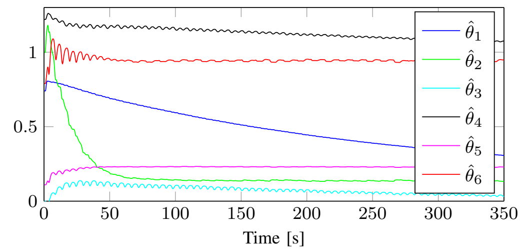


Fig. 6. Estimated system parameters during a real run with the adaptive dynamic controller with $\hat{\boldsymbol{\theta}}_{\mathbf{0}} = [0.74\ 1.00\ 0.00\ 1.22\ 0.11\ 0.79]^T$ and $\mathbf{\Gamma} = \text{diag}(1.0, 1.0, 1.0, 1.0, 1.0, 1.0)$.

[2] T. Utstumo, T. Berge, and J. T. Gravdahl, "Non-linear model predictive control for constrained robot navigation in row crops," *in the Proceedings of the 2015 IEEE International Conference on Industrial Technology (ICIT 2015), Seville, Spain*, March 17-19 2015.

[3] T. Utstumo and J. T. Gravdahl, "Implementation and comparison of attitude estimation methods for agricultural robotics," *Agricontrol. Vol. 4. No. 1. 2013*, pp. 52–57, 2013.

[4] C. De La Cruz and R. Carelli, "Dynamic Modeling and Centralized Formation Control of Mobile Robots," *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, pp. 3880–3885, Nov. 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4153091

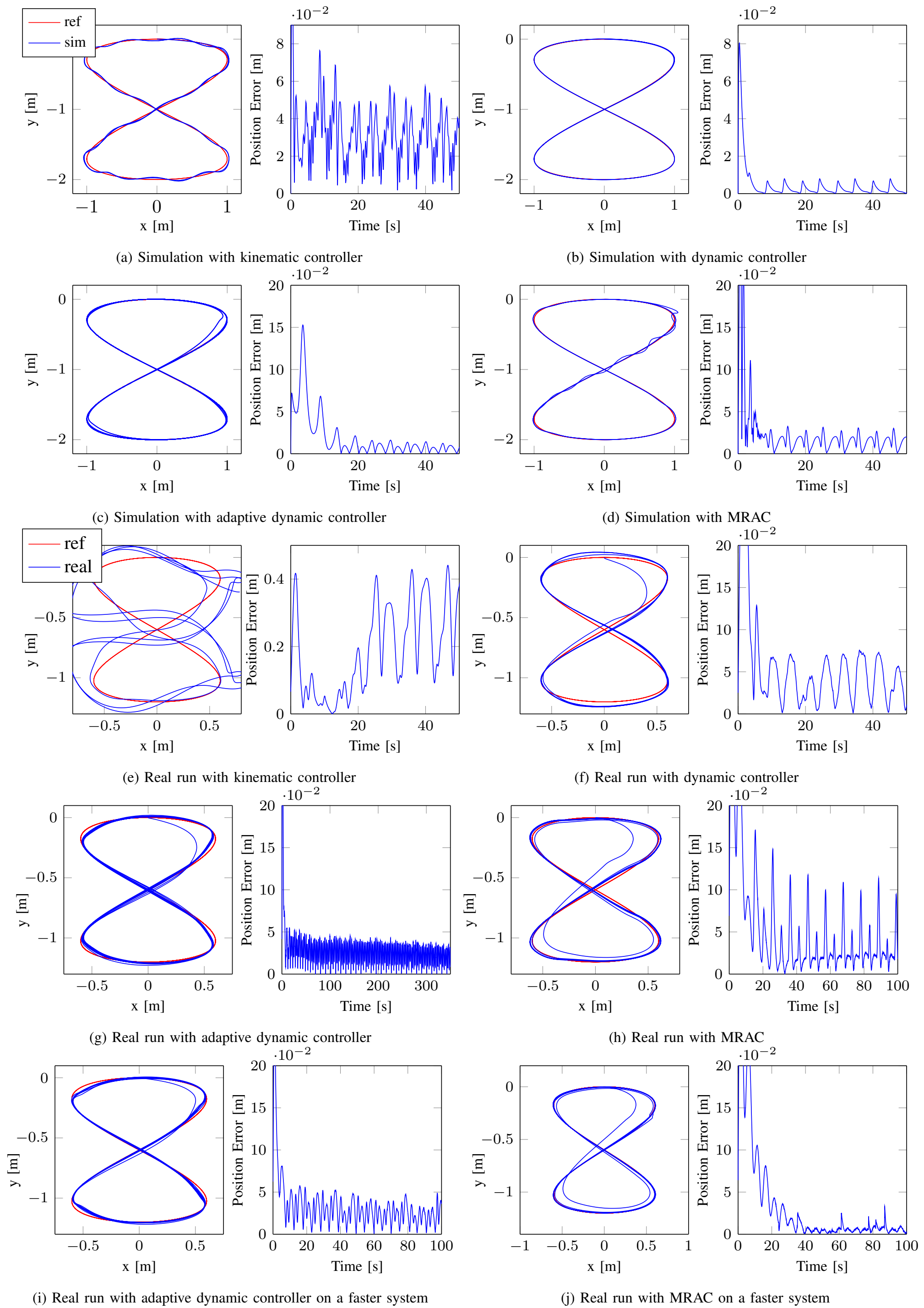
[5] F. N. Martins, W. C. Celeste, R. Carelli, M. Sarcinelli-Filho, and T. F. Bastos-Filho, "An adaptive dynamic controller for autonomous mobile robot trajectory tracking," *Control Engineering Practice*, vol. 16, no. 11, pp. 1354–1363, Nov. 2008. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0967066108000373

[6] T. Fukao, H. Nakagawa, and N. Adachi, "Adaptive tracking control of a nonholonomic mobile robot," *Robotics and Automation, IEEE Transactions on*, vol. 16, no. 5, pp. 609–615, Oct 2000.

[7] E. Canigur and M. Ozkan, "Model reference adaptive control of a nonholonomic wheeled mobile robot for trajectory tracking," in *Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on*, July 2012, pp. 1–5.

[8] F. Dong, W. Heinemann, and R. Kasper, "Development of a row guidance system for an autonomous robot for white asparagus harvesting," *Computers and Electronics in Agriculture*, vol. 79, no. 2, pp. 216 – 225, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169911002304

[9] P. Ioannou and J. Sun, *Robust Adaptive Control*. Dover Publications, 2013. [Online]. Available: https://books.google.no/books?id=ffavAAAAQBAJ



(a) Simulation with kinematic controller

(b) Simulation with dynamic controller

(c) Simulation with adaptive dynamic controller

(d) Simulation with MRAC

(e) Real run with kinematic controller

(f) Real run with dynamic controller

(g) Real run with adaptive dynamic controller

(h) Real run with MRAC

(i) Real run with adaptive dynamic controller on a faster system

(j) Real run with MRAC on a faster system

Fig. 8. Comparison of all the controllers with a relatively slow system and two selected runs on a faster system.

BIBLIOGRAPHY

1. Grift, T., Zhang, Q., Naoshi, K. & Ting, K.
A Review of Automation and Robotics for The Bio-Industry.
Biomechatronics Engineering **1,** 37–54 (2008).

2. Johnson, G. A., Mortensen, D. A. & Martin, A. R.
A Simulation of Herbicide Use Based On Weed Spatial-Distribution.
Weed Research **35,** 197–205 (1995).

3. Urdal, F., Utstumo, T., Ellingsen, S. A., Vatne, J. K. & Gravdahl, T. *Design and control of precision drop-on-demand herbicide application in agricultural robotics* in *The 13th international Conference on Control, Automation, Robotics and Vision, Singapore* (2014).

4. Utstumo, T., Berge, T. & Gravdahl, J. T.
Non-linear Model Predictive Control for constrained robot navigation in row crops.
in the Proceedings of the 2015 IEEE International Conference on Industrial Technology (ICIT 2015), Seville, Spain (Mar. 2015).

5. Utstumo, T. & Gravdahl, J. T. Implementation and Comparison of Attitude Estimation Methods for Agricultural Robotics. *Agricontrol. Vol. 4. No. 1. 2013,* 52–57 (2013).

6. Sainz-Costa, N., Ribeiro, A., Burgos-Artizzu, X. P., Guijarro, M. & Pajares, G. Mapping wide row crops with video sequences acquired from a tractor moving at treatment speed. *Sensors* **11,** 7095–7109 (2011).

7. Biber, P., Weiss, U., Dorna, M. & Albert, A.
Navigation system of the autonomous agricultural robot Bonirob
in *Workshop on Agricultural Robotics: Enabling Safe, Efficient, and Affordable Robots for Food Production (Collocated with IROS 2012), Vilamoura, Portugal* (2012).

8. Johnson, S. *Stephen Johnson on Digital Photography* ISBN: 9780596523701.
<http://books.google.pl/books?id=0UVRXzF91gcC>
(O'Reilly Media, Incorporated, 2006).

9. Nixon, M. & Aguado, A.
in *Feature Extraction & Image Processing for Computer Vision, Second Edition* (2012).
ISBN: 0123725380. doi:10.1016/B978-0-12-396549-3.00003-3.

10. Koschan, A. & Abidi, M. *Digital Color Image Processing* ISBN: 9780470230350.
<http://books.google.pl/books?id=lK8rrNJMZBoC> (Wiley, 2008).

11. Lukac, R. & Plataniotis, K. *Color Image Processing: Methods and Applications*
ISBN: 9781420009781. <http://books.google.pl/books?id=Bj0Mgc9Wqw4C>
(CRC Press, 2006).

12. Prince, S. J. D. *Computer Vision: Models, Learning, and Inference* 1st.
ISBN: 1107011795, 9781107011793
(Cambridge University Press, New York, NY, USA, 2012).

13. Szeliski, R. Computer Vision : Algorithms and Applications. *Computer* **5,** 832 (2010).

14. Hartley, R. I. & Zisserman, A. *Multiple View Geometry in Computer Vision* Second (Cambridge University Press, ISBN: 0521540518, 2004).

15. A Threshold Selection Method from Gray-Level Histograms. *Systems, Man and Cybernetics, IEEE Transactions on* **9,** 62–66 (Jan. 1979).

16. Swain, M. & Ballard, D. Indexing via color histograms. *[1990] Proceedings Third International Conference on Computer Vision.* doi:10.1109/ICCV.1990.139558 (1990).

17. Hough, P. V. C. Method and means for recognizing complex patterns. *US Patent 3,069,654* **21,** 225–231 (1962).

18. Duda, R. O. & Hart, P. E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Commun. ACM* **15,** 11–15 (Jan. 1972).

19. Fischler, M. A. & Bolles, R. C. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography* 1981. doi:10.1145/358669.358692.

20. Morro, A., Sgorbissa, A. & Zaccaria, R. Path Following for Unicycle Robots With an Arbitrary Path Curvature. *Robotics, IEEE Transactions on* **27,** 1016–1023 (Oct. 2011).

21. Khalil, H. *Nonlinear Systems* ISBN: 9780130673893. <https://books.google.no/books?id=t%5C_d1QgAACAAJ> (Prentice Hall, 2002).

22. Crassidis, J. L., Markley, F. L. & Cheng, Y. *Survey of Nonlinear Attitude Estimation Methods* 2007. doi:10.2514/1.22452.

23. Shuster, M. D. A Survey of Attitude Representations. *The Journal of the Astronautical Sciences* **1,** 439–517 (1993).

24. Purvis, M. *ROS driver for serial-connected Roboteq motor drivers* https://github.com/g/roboteq. 2014.

25. Zhang, Y. Z. Y., Hong, D. H. D., Chung, J. & Velinsky, S. Dynamic model based robust tracking control of a differentially steered wheeled mobile robot. *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)* **2.** doi:10.1109/ACC.1998.703528 (1998).

26. Matas, J., Galambos, C. & Kittler, J. Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding* **78,** 119–137 (2000).