Department of Automation and Systems
Center of Technology
Federal University of Santa Catarina

Vision-based Sensor System for Mobile Robot Navigation

Final Thesis submitted to the Federal University of Santa Catarina
as a requisite to obtain the degree of

Control and Automation Engineer

by

Hugo Leonardo Gosmann

Florianópolis, May 1999.

Vision-based Sensor System for Mobile Robot Navigation

Hugo Leonardo Gosmann

This thesis was judged in the context of the discipline **EEL 5901: Final Thesis** and was approved in its final version by the **Department of Automation and Systems** (Undergraduate Degree in Control and Automation Engineering) of this University.

Florianópolis, May 1999.

Examiners:

Dr. Sjur Jonas Vestli
Supervisor at ETHZ

Prof. Jean-Marie Farines
Supervisor at UFSC

Prof. Augusto Humberto Bruciapaglia
Head of the Department

Prof. Guilherme Bittencourt
Invited Professor

Giulliano Carlo Jesus Pereira, first student

Gil Riella, second student

to my family...

Acknowledgements

I owe a thousand thanks to so many people that it would be impossible not to forget somebody. Forgive me for my limited memory.

First of all I would like to thank Prof. Dr. G. Schweitzer that gave me the opportunity to go to Switzerland and work at the IfR.

Thanks to Sjur Vestli, my supervisor at IfR, for his help, guidance and countless discussions and also to Nicola Tomatis and Kai Arras from EPFL for their suggestions and help.

"Grazie" to my italian friend Roberto Brega for his help with XOberon and for harmoniously sharing the same office with me.

Special thanks go to Frau Erika Cassoli for all her friendship and also technical help that facilitated my stay in Switzerland.

I would like to express my gratitude to Martin Adams and Shao-Jü Woo for their friendship and for the great time I had in Männedorf.

My thanks to Prof. Jean-Marie Farines, for his valuable suggestions, encouragement and support from the very beginning of this project.

I would also like to thank all my colleagues at UFSC for their friendship during the last 5 years. My special gratitude to Mario A. Cologni, Giulliano C. J. Pereira and Patrick L. Morei.a and to professors Augusto H. Bruciapaglia and Alexandre Trofino.

Many thanks to Salete, our secretary at UFSC, for her help with all the bureaucratic stuff at our university.

My sincere thanks to Mrs. Vera Marcelino for her support in the last phase of this work and to Mr. Gileno Marcelino for allowing me to establish my headquarters at his home office.

I owe deep gratitude to my parents, my sister and brother, and my aunt Clara for all their support during these 5 months far away from home.

Last but not least, I would like to deeply thank my girlfriend Giuliana for her constant support, motivation, patience and love through all these months of separation.

Abstract

This work corresponds to a 5-month internship in mobile robotics carried out by the author at the Institute of Robotics of the ETHZ (Swiss Federal Institute of Technology).

During this period of time a method for mobile robot navigation in a known environment was developed. The technique combines position estimation from odometry with observations of the environment from a CCD camera. Fixed lamps in the environment provide landmarks. The position of these landmarks is known a priori by the robot.

At each localization cycle an image of the environment is captured and processed by the vision system. The information obtained is then used by a Kalman filter to correct the position and orientation of the robot.

The system was implemented in the SmartROB, a mobile robot platform developed at the ETHZ. Results from experiments in a real environment are presented.

Resumo (Estendido)

Este trabalho corresponde ao Projeto de Fim de Curso desenvolvido pelo autor durante 5 meses no Instituto de Robótica do ETHZ (Swiss Federal Institute of Technology). Durante este período um método para navegação de robôs móveis foi estudado. Este método é descrito em mais detalhes na sequência.

Na navegação de robôs móveis uma das questões mais importantes diz respeito à localização. Como determinar a posição do robô a partir de informações a priori a respeito do ambiente e também dos dados provenientes de sensores? No presente trabalho explora-se a solução deste problema aplicando-se a idéia da "fusão de sensores". Um sistema de visão foi escolhido, mas a mesma técnica pode ser utilizada com outros tipos de sensores como laser scanners ou sonares por exemplo.

O processo de localização pode ser interpretado como a determinação das coordenadas (x, y, θ) do robô com relação a um sistema de coordenadas de referência. Para um robô dotado de rodas, a forma mais natural, simples e barata para realizar esta tarefa é usando a hodometria, ou seja, usando as informações proveniente de encoders instalados em cada um dos motores acoplados às rodas. O problema é que por vários fatores como imprecisões no modelo, folgas mecânicas e irregularidades no chão, esta técnica gera um acúmulo ilimitado de erros que leva o robô a perder sua referência.

Para superar este problema outros sensores devem ser incorporados ao robô para possibilitar uma localização mais precisa e robusta. No caso de um sistema de visão, marcas artificiais são instaladas em pontos predeterminados do ambiente. As marcas escolhidas para a realização deste trabalho foram lâmpadas fluorescentes. Por serem fontes de luz elas provêem um bom nível de contraste nas imagens, facilitando a sua detecção.

A idéia de combinar as informações provenientes da hodometria com informações do sistema de visão é implementada na prática utilizando-se um filtro de Kalman. Este filtro é composto de dois modelos. Um é o modelo da planta, que descreve como a posição do robô muda a cada intervalo de tempo. O outro é o modelo do sensor, que descreve a relação entre a posição do robô em um dado instante e a posição das marcas no ambiente.

A cada ciclo do algoritmo, a posição do robô e a incerteza associada a ela são primeiramente calculadas a partir da hodometria. Em seguida calculam-se os ângulos com relação a posição atual do robô, onde se espera encontrar as

marcas do ambiente. Este cálculo é possível porque as posições das marcas são conhecidas a priori pelo robô.

No passo seguinte obtém-se as informações do sistema de visão. Uma imagem do ambiente é capturada e dela são extraídas as posições (ângulos) das lâmpadas em relação ao robô. Compara-se então estes ângulos obtidos pelo sensor com os ângulos esperados calculados anteriormente. Essa comparação visa separar os ângulos que realmente correspondem às marcas do ambiente daqueles que foram introduzidos de forma espúria.

Após este processo calcula-se o ganho do filtro de Kalman que é usado para corrigir a posição atual do robô e a sua respectiva incerteza, finalizando assim o ciclo.

O ponto chave para entender todo o processo é perceber que através da hodometria tem-se sempre à disposição uma aproximação inicial da posição atual do robô, mesmo que esta não seja muito precisa. A informação proveniente do sistema de visão dirá então, que o robô está numa posição um pouco diferente daquela estimada no início do ciclo. Neste momento entra o filtro de Kalman, que combina estas duas informações de forma estatística atribuindo a cada uma delas um peso correspondente às incertezas a elas associadas. O resultado obtido é uma nova posição (agora corrigida) com uma menor incerteza associada.

O algoritmo de localização desenvolvido neste trabalho foi implementado em um robô chamado SmartROB. Este robô é uma plataforma básica de robô móvel totalmente desenvolvida no Instituto de Robótica do ETHZ. Os resultados mostraram que quando se utiliza somente a hodometria para realizar o processo de localização, existe um aumento ilimitado da incerteza em relação à posição real do robô, inviabilizando assim uma navegação de precisão.

Por outro lado, quando se combina a hodometria com o sistema de visão, consegue-se realizar uma navegação precisa, mantendo a incerteza sempre limitada. A implementação realizada, apesar de não ter sido ótima, possibilitou o entendimento e também a ilustração, na prática, do funcionamento de um sistema simples de navegação de robôs móveis.

The Three Laws of Robotics

#1 A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

#2 A robot must obey orders given it by human beings except where such orders would conflict with the First Law.

#3 A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

ISAAC ASIMOV

Contents

List of Figures

List of Symbols

$\nabla \mathbf{f}$	state transition Jacobian
G	validation gate
$\nabla \mathbf{h}$	measurement Jacobian
n_O	number of observations (extracted landmarks)
n_E	number of expected landmarks
$\mathbf{P}(k+1\|k)$	vehicle position estimate covariance
$\mathbf{p}_i$	position vector of landmark i
$\mathbf{Q}(k)$	plant noise covariance matrix
$\mathbf{R}(k)$	observation noise covariance matrix
$\mathbf{S}(k)$	innovation covariance
$\mathbf{u}(k)$	vehicle control input
$\mathbf{v}(k)$	plant noise
$\mathbf{W}(k)$	Kalman gain
$\mathbf{w}(k)$	measurement noise
$\mathbf{x}(k)$	vehicle position state vector
$\hat{\mathbf{x}}(k+1\|k)$	vehicle position estimate
$Z(k)$	current set of observations (extracted landmarks)
$\hat{Z}(k)$	current set of predicted landmarks
$\mathbf{z}_j(k)$	extracted landmark or observation
$\hat{\mathbf{z}}_i(k)$	predicted landmark
$\nu(k)$	innovation

Chapter 1

Introduction and Motivation

Two pages of the final manuscript... Although they look like a first draft, they had already been rewritten and retyped - like almost every other page - four or five times. With each rewrite I try to make what I have written tighter, stronger and more precise, eliminating every element that is not doing useful work. Then I go over it once more, reading it aloud, and am always amazed at how much clutter can still be cut.

- WILLIAM ZINSSER, On Writing Well (1990)

Leonard and Durrant-Whyte [LDW92], summarized the general problem of mobile robotics by the following three questions: "Where am I?", "Where am I going?" and "How should I get there?".

The first question corresponds to *localization*, one of the major tasks of autonomous robot navigation. How can I work out where I am in a given environment, based on what I can see and what I have previously been told? The second and third questions are essentially those to specifying a goal and being able to plan a path that results in achieving this goal. Investigation of the latter two questions usually come under the domain of path planning and obstacle avoidance.

In a typical indoor environment with a flat floor, the task of localization becomes a

matter of determining the Cartesian coordinates (x, y) and the orientation θ of the robot on a two dimensional plan. For a wheeled robot, odometry (also known as *dead reckoning*) is one of the most important means of achieving this task. In practice, optical encoders that are mounted on both drive wheels, feed discrete wheel increment information to a processor. This processor continually updates the robot's state using geometric equations. However, with time, odometric localization unboundedly accumulates errors due to several problems:

(i) surface roughness and undulation may cause the distance to be over-estimated;

(ii) wheel slippage can cause the distance to be under-estimated;

(iii) variations in load can distort the odometry wheels and introduce additional errors;

(iv) discrete sampling of wheel increments also contribute to inaccurate measurements.

Although good approaches have already been investigated [CK97], all these problems make it rather difficult to perform really accurate navigation only using odometry. Therefore, some other kind of position updating method must be used. To reach its destination with reasonable accuracy, the robot requires external sensors and sensor fusion algorithms to relate knowledge about its environment to the information obtained from its sensors.

Many authors have extensively studied different kinds of sensors, such as laser scanners [Arr96, HGB97, Ada99], vision sensors [Kro89, CC92, CSCD97, Tom98b] and sonars [LDW92]. The idea is to combine position estimation from odometry with observations of the environment obtained from the chosen sensor.

In case of a laser scanner, reflectors (also known as *beacons*) are used. The positions of these beacons are known *a priori* by the robot, and constitute the map of the environment.

Using sonar, the idea is slightly different. Planes, cylinders, corners and edges (known

as *targets*) are used to model the environment.

For a vision sensor, fixed objects in the known environment provide *landmarks* which are listed in a database. Landmarks are distinct features (*e.g.* rectangles, lines, circles) that a robot can recognize from its sensory inputs. In general, landmarks have a fixed and known position, relative to which a robot can localize itself. They are carefully chosen to be easy to identify; for example, there must be sufficient contrast to background. Before a robot can use landmarks for navigation, their characteristics must be known and stored in the robot's memory. The main task in localization is then to recognize the landmarks reliably and to calculate the robot's position.

In all these cases, the Kalman filter is the basic tool to approach navigation and sensor fusion. It combines all measurement data to get an optimal estimate of the system state in a statistical sense. The inputs to a Kalman filter are the system measurements. The *a priori* information are the system dynamics and the noise properties of the system and the sensors. The outputs of the Kalman filter are the estimated system state and the innovation (*i.e.*, the difference between the predicted and observed measurement). The extended Kalman filter is a version of the Kalman filter that can handle non-linear dynamics or non-linear measurement equations [BSF88, May90, AG92, GA93].

Many other sensors and methods can be used for mobile robot positioning, which are beyond the scope of this work. A good starting point is the extensive survey presented by Borenstein and Everett in [BEF96].

1.1 Specification of our Problem

In the recent years the ETHZ (Swiss Federal Institute of Technology), through the IfR (Institute of Robotics), has developed a mobile robot platform called SmartROB.

The SmartROB system is a versatile, easy to use, mobile robot kit, suitable for the realization of a wide variety of tasks. It has been used with great success since 1992 in the lecture "Smart Mechatronic Product Design", where groups of students develop a mechatronic product based on a predeterminately posed problem.

The present work has been motivated from the very beginning by the success of this experience and also by the interest of embedding a vision system in the SmartROB, reinforced by the importance and evidence of the subject in the research world nowadays. Moreover, the recent development at the ETHZ of a new framer grabber compatible with XOberon and the availability of a CCD camera also contributed to our choice.

The main goals of this work are: first, to better understand where the problems in mobile robotics are and in which directions the solutions to them are heading to; second, to implement a simple localization system, using vision and odometry, to be used in the SmartROB.

The chosen environment was part of the Laboratory room at the ETHZ. In this area, three fluorescent lamps were vertically placed in predetermined positions to be used as landmarks. Figure 1.1 shows a map of this environment and Figure 1.2 a photo of the area in the Laboratory. The SmartROB, equipped with a CCD camera and a frame grabber, should be able to localize itself and navigate using information from odometry (encoders) and the vision systems (position of landmarks extracted from images).

1.2 Structure of this work

This work is structured as follow:

Chapter 2 presents the SmartROB system. The mechanical, electrical and software features are described, including the real-time operating system XOberon.

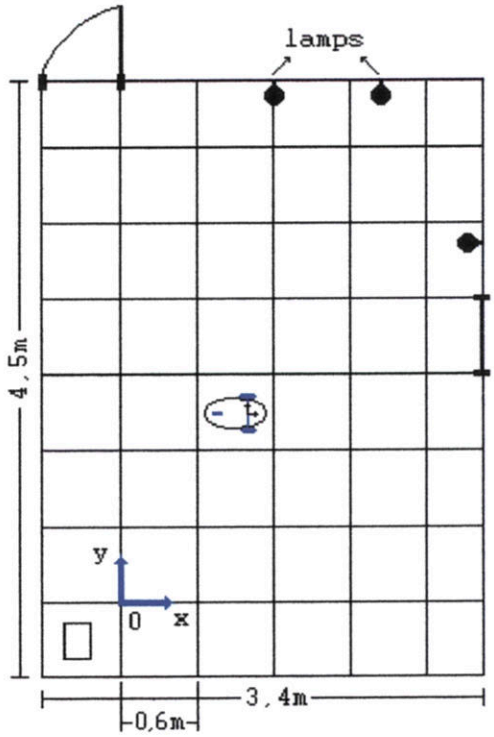

Figure 1.1: Map of the Environment

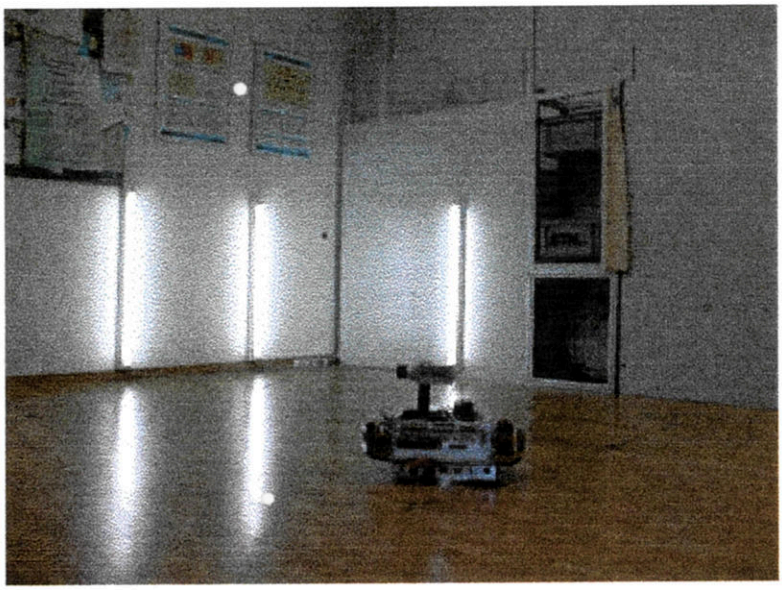

Figure 1.2: The Laboratory Area

Chapter 3 discusses the localization method. The theoretical aspects of the Kalman filter are presented and discussed with respect to our problem. An implementation of a localization algorithm is proposed.

Chapter 4 discusses our vision system. A method of identifying landmarks is proposed.

Chapter 5 presents some experiments and discusses our results.

Finally, **Chapter 6** provides an overall discussion of this work and also proposes some ideas on how to continue it.

Chapter 2

Description of the System

Sometimes, to be able to understand the whole
we need to know all details.

Anonymous

2.1 The Robot

The SmartROB system is a mobile robot platform based on a Power PC 604 microprocessor. In the last 6 years the SmartROB has been used by undergraduate and graduate students at the ETHZ in the development of different products in mechatronics. Each year a different problem is posed to the students, which search for their own solutions.

Problems like finding golf-balls in a square field and placing them in a center hole, building a tower of wooden blocks as high as possible, collecting different objects in a labyrinth, playing soccer, finding burning candles in a labyrinth and covering them with a cup to extinguish the fire and developing an autonomous vacuum cleaning robot have already been investigated.

Although these themes seem quite ludic, they involve many non-trivial problems in mobile robotics that require intelligent approaches to be solved. This challenging task

encourages students to use their creativity and improve their knowledge in different areas such as sensors, actuators, mechanics, electronics and software [SBB98].

To fulfill the requirements of a general-purpose platform for quickly developing simple products in mechatronics, an appropriate hardware is needed. Below, the basic features of the SmartROB system are briefly described. Figure 2.1 shows a picture of it.

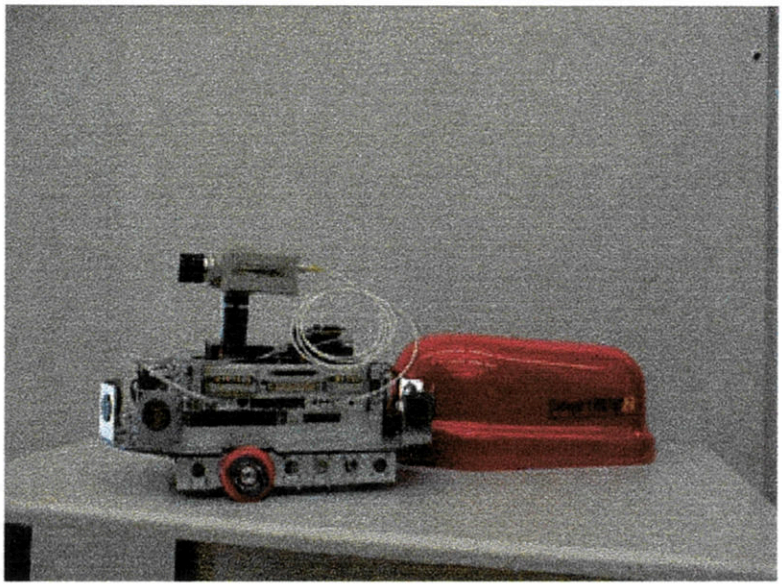

Figure 2.1: The SmartROB System

2.1.1 Electronics

The heart of the SmartROB system is a Power PC 604 32-bit processor running at $300MHz$. It has a $16MB$ (expandible to $128MB$) on-board ECC DRAM and a $32KB$ L1 cache. An Ethernet transceiver interface with 32-bit PCI local bus DMA and an asynchronous serial port are also available.

It contains multiple analogue and digital inputs and outputs and eight $100W$ power amplifiers ($24V$, $4A$ each), one needed per DC servomotor.

The whole electronics is assembled in a small rack including the power supply for ± 12 and $5V$ for the microprocessor and for the electrically isolated periphery and power electronics. Additional PCI Mezzanine Card (PMC) slots and a general-purpose slot allow easy extension.

2.1.2 Mechanics

The basic mechanical hardware of the SmartROB is a mobile platform with three wheels. The chassis is based on aluminum profiles. The two front wheels are driven by two independently controlled DC-motors with integrated gearbox. The third wheel is a castor. For optimal position and speed control, the motors are equipped with encoders.

2.1.3 Software & Periphery

The Power PC 604 processor runs the real-time operating system XOberon [Bre98] developed at the Institute of Robotics, which is based on Oberon (object oriented successor of Modula2 and Pascal) [Rei91, Mös93]. A complete software library for handling a wide range of processor functions, I/O, basic sensor programming and robot motion functions are also available.

Due to its modularity, the compatibility of its components and the ease of program development and testing, the system allows various configurations of mechatronic products to be quickly built and tested.

XOberon is a development system that runs either on Macintosh, Sun or PC host computers. The target system (The SmartROB in this case) is connected through a serial link with the host computer. A monitor program that executes basic commands, *e.g.* for downloading and debugging programs, runs on the target processor.

The main features of XOberon are:

- Real-time processing;
- Deadline-driven scheduler;
- Safe Memory Architecture with Paging Support;
- Safe dynamic linking and loading;
- Garbage Collector;
- Web server with CGI support, allowing remote control and diagnosis;
- Telnet server, FTP server, TFTP server and SMTP (Mail) client;
- Database-oriented driver architecture;
- High reliability and run-time safety;
- Digital configurable controllers for PID position and velocity control;
- Drivers for periphery.

Different sensors can also be integrated with the SmartROB, such as infrared triangulation sensors (mounted on a servomotor for scanning), ultrasound sensors (with micro-controller and serial interface) and frame grabber for standard B/W as well as color cameras (composite or S-Video).

2.2 Vision Hardware

A frame grabber and a CCD camera compose the hardware used in our vision system.

2.2.1 The PMC-FG frame grabber

The PMC-FG frame grabber is a PMC (PCI Mezzanine Card) capturing module, based on the Bt848A processor.

The Bt848A integrates a NTSC/PAL/SECAM composite and S-video decoder, scaler, DMA controller, and PCI Bus master on a single device.

The PMC format family is usable on (but not limited to) single VME64 boards, Multibus I and II boards, single slot Futurebus + modules, desktop computers, portable computers, servers and similar type of applications. The electrical and logical layers are based on the PCI Specification from the PCI Special Interest Group.

The PMC-FG features precise video capturing hardware for applications that require high color accuracy. Hardware features include:

- High color accuracy with low pixel jitter;
- PCI bus master design for real-time image capturing to system memory;
- Image capturing resolution up to full-size: 768×576 (PAL and SECAM) and 640×480 (NTSC);
- Horizontal and vertical clipping and scaling of captured images to minimize system memory usage and bus bandwidth requirements;
- Common color output formats, including RGB and Y8 (grayscale);
- Continuos, software-controlled image captures;
- Four multiplexed composite video inputs (one input can be S-video);
- $+12V$ output for powering cameras and other devices.

2.2.2 The CCD Camera

The camera used in this work was a common CCD (Charge-Coupled Device) camera, which contains a semiconductor chip with a light-sensitive grid, used for converting images into electrical signals. It was already at hand in the Laboratory of the IfR and no further technical data was available.

Chapter 3

Localization

No matter where you go, there you are.

The motto of the USS Excelsior.
STAR TREK VI, The Undiscovered Country

Localization is the process of determining the position of the robot with respect to a global reference frame (*i.e.* a coordinate system). It is a cyclic process that should continuously keep the robot on track.

In our case, position updates are produced by a matching algorithm that uses an initial estimate of the vehicle position from odometry. A vision-based sensor system is then used. The aim of this system is to recognize known landmarks in the environment based on an *a priori* map, and then determine the position and orientation of these landmarks relative to the robot.

A combination of information from odometry and from the vision system is used to update the robot's position. The tool used to fuse this information is the Kalman filter, which is described in the following sections.

3.1 Modeling Odometry

Before talking about the Kalman filter, we should dedicate some words to explain in more detail the odometry model. In the SmartROB, the odometry calculation is based on two optical encoders mounted on the front wheels. We assume that Δs_l is the distance traveled by the left wheel, and Δs_r is the distance traveled by the right wheel in the time cycle Δt.

Let the origin of the robot's coordinate system be the middle point of the front axle, as shown in Figure 3.1. Then, the position and orientation changes of the robot are calculated using:

$$\Delta s = \frac{1}{2}(\Delta s_r + \Delta s_l), \quad \Delta\theta = \frac{1}{H}(\Delta s_r - \Delta s_l) \tag{3.1}$$

where H is the distance between the two front wheels. In other words, the robot's location changes by a translation forward through the distance Δs followed by a rotation counterclockwise through the angle $\Delta\theta$.

3.2 The Kalman Filter

We denote the position and orientation of the vehicle at time step k by the state vector $\mathbf{x}(k) = [x(k), y(k), \theta(k)]^T$ comprising a Cartesian location and an angle defined with respect to a global coordinate system, as shown in Figure 3.1. At initialization, the robot starts at a known position (*e.g.* the origin), and has an *a priori* map of n_E landmarks, whose locations are specified by the set of known vectors $\{\mathbf{p}_i = (p_x, p_y) \mid 1 \leq i \leq n_E\}$.

At each time step, observations $\mathbf{z}_j(k+1)$ of these landmarks are taken. In our vision system, an image is captured and then processed to extract the landmarks. Each extracted landmark is converted into an angle with respect to the robot coordinate system. These angles compose our observations.

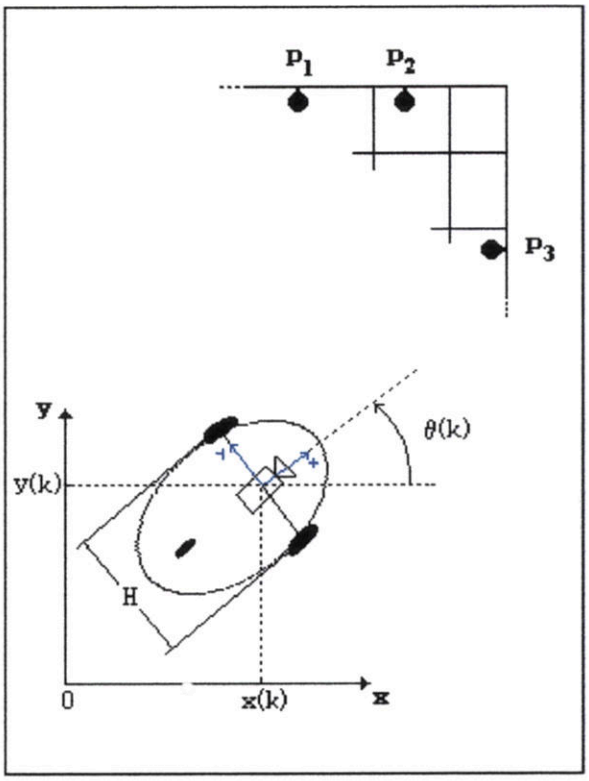

Figure 3.1: Global and Local Coordinate Systems

The extended Kalman filter is then used to associate measurements $\mathbf{z}_j(k+1)$ with the correct landmarks $\mathbf{p}_i$ to compute $\hat{\mathbf{x}}(k+1|k+1)$, the updated estimate of the vehicle's position.

The Kalman filter consists of two models: a *plant* model and a *measurement* model.

3.2.1 The Plant Model

The plant model describes how the vehicle's position $\mathbf{x}(k)$ changes with time in response to a control input $\mathbf{u}(k)$ and a noise disturbance $\mathbf{v}(k)$, and has the form

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) + \mathbf{v}(k), \qquad \mathbf{v}(k) \sim N(\mathbf{0}, \mathbf{Q}(k)) \tag{3.2}$$

where $\mathbf{f}(\mathbf{x}(k), \mathbf{u}(k))$ is the non-linear state transition function. We use the notation $\mathbf{v}(k) \sim N(\mathbf{0}, \mathbf{Q}(k))$ to indicate that the noise source is assumed to be zero-mean Gaussian with covariance $\mathbf{Q}(k)$.

The control input $\mathbf{u}(k) = [\Delta s(k), \Delta\theta(k)]^T$ comes from the odometry model discussed in the previous section, and leads us to the following state transition function:

$$\mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) = \begin{bmatrix} x(k) + \Delta s(k) \cos\theta(k) \\ y(k) + \Delta s(k) \sin\theta(k) \\ \theta(k) + \Delta\theta(k) \end{bmatrix} \tag{3.3}$$

3.2.2 The Measurement Model

The robot is equipped with a CCD camera and a frame grabber. For our convenience the camera was installed in the origin of the robot's coordinate system. The information we have *a priori* is the position of our n_E landmarks with respect to the global coordinate system. Our vision system will provide us with angles corresponding to the extracted landmarks (*i.e.* observations). Then we can define the set of observations

$$Z(k) = \{\mathbf{z}_j(k) \mid 1 \leq j \leq n_O\} \tag{3.4}$$

where n_O is the number of *observed* landmarks. We have already seen that n_E is the total number of known landmarks, which will also be referred to as the number of *expected* landmarks.

Details about the vision system will be discussed in Chapter 4.

The measurement model relates a sensor observation to the vehicle position and has the form:

$$\mathbf{z}_j(k) = \mathbf{h}(\mathbf{x}(k), \mathbf{p}) + \mathbf{w}_j(k), \qquad \mathbf{w}_j(k) \sim N(\mathbf{0}, \mathbf{R}_j(k)) \tag{3.5}$$

The measurement function $\mathbf{h}(\mathbf{x}(k), \mathbf{p})$ expresses an observation $\mathbf{z}(k)$ from the sensor as a function of the vehicle position $\mathbf{x}(k)$. It has the following form:

$$\mathbf{h}(\mathbf{x}(k), \mathbf{p}) = \arctan\left(\frac{\mathbf{p}_y - y(k)}{\mathbf{p}_x - x(k)}\right) - \theta(k). \tag{3.6}$$

Each observation is assumed corrupted by a zero-mean, Gaussian disturbance $\mathbf{w}_j(k)$ with covariance $\mathbf{R}_j(k)$.

3.3 The Localization Cycle

Given the *a posteriori* vehicle position estimate $\hat{\mathbf{x}}(k|k)$ and its covariance $\mathbf{P}(k|k)$ for time k, the current control input $\mathbf{u}(k)$, the current set of observations $Z(k+1)$ and the *a priori* map, compute the new *a posteriori* estimate $\hat{\mathbf{x}}(k+1|k+1)$ and its covariance $\mathbf{P}(k+1|k+1)$. The algorithm consists of the following steps: position prediction, observation, measurement prediction, matching, and estimation.

3.3.1 Vehicle Position Prediction

First, using the plant model and the knowledge of the control input $\mathbf{u}(k)$, we predict the robot's new location at time step $k+1$:

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{f}(\hat{\mathbf{x}}(k|k), \mathbf{u}(k)). \tag{3.7}$$

Next we compute $\mathbf{P}(k+1|k)$, the variance associated with this prediction:

$$\mathbf{P}(k+1|k) = \nabla\mathbf{f}\ \mathbf{P}(k|k)\ \nabla\mathbf{f}^T + \mathbf{Q}(k) \tag{3.8}$$

where $\nabla\mathbf{f}$ is the Jacobian of the state transition function $\mathbf{f}(\hat{\mathbf{x}}(k|k), \mathbf{u}(k))$ obtained by linearizing about the updated state estimate $\hat{\mathbf{x}}(k+1|k)$:

$$\nabla\mathbf{f} = \begin{bmatrix} 1 & 0 & -\Delta s(k)\sin(\theta(k)) \\ 0 & 1 & \Delta s(k)\cos(\theta(k)) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

3.3.2 Observation

The next step is to obtain the observation set $Z(k+1)$ from the vehicle's sensor system on the new vehicle location, that is, capture an image and apply an algorithm to identify the landmarks. Then convert these landmarks into angles.

3.3.3 Measurement Prediction

Now we use the predicted robot location $\hat{\mathbf{x}}(k+1|k)$ and the *a priori* map to generate predicted observations for each landmark $\mathbf{p}_i$:

$$\begin{aligned} \hat{\mathbf{z}}_i(k+1) &= \mathbf{h}_i(\hat{\mathbf{x}}(k+1|k), \mathbf{p}_i) \\ &= \arctan\left(\frac{\mathbf{p}_{i_y} - y(k)}{\mathbf{p}_{i_x} - x(k)}\right) - \theta(k), \quad i = 1, \ldots, n_E \end{aligned} \tag{3.10}$$

to yield the set of predictions:

$$\hat{Z}(k+1) = \{\hat{\mathbf{z}}_i(k+1) \mid 1 \leq i \leq n_E\} \tag{3.11}$$

which contains n_E predicted (*i.e* expected) landmarks. Figure 3.2 illustrates the calculation of $\hat{\mathbf{z}}_i(k+1)$.

The predicted state estimate $\hat{\mathbf{x}}(k+1|k)$ is used to compute the measurement Jacobian

$$\nabla\mathbf{h}_i = \begin{bmatrix} \frac{\mathbf{p}_{i_y} - y(k)}{(\mathbf{p}_{i_x} - x(k))^2 + (\mathbf{p}_{i_y} - y(k))^2} \\ \frac{-\mathbf{p}_{i_x} - x(k)}{(\mathbf{p}_{i_x} - x(k))^2 + (\mathbf{p}_{i_y} - y(k))^2} \\ -1 \end{bmatrix}^T \tag{3.12}$$

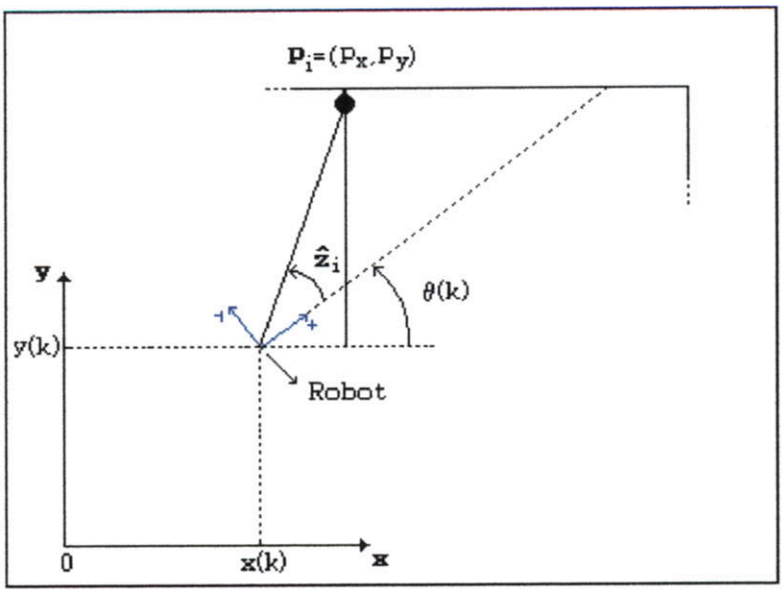

Figure 3.2: Calculation of $\hat{\mathbf{z}}_i(k+1)$

for each prediction.

3.3.4 Matching

The goal of the matching procedure is to produce an assigment from measurements $\mathbf{z}_j(k)$ to landmarks $\mathbf{p}_i$. For each prediction and observation we first compute the innovation ν_{ij}.

$$\begin{aligned} \nu_{ij}(k+1) &= [\mathbf{z}_j(k+1) - \hat{\mathbf{z}}_i(k+1)] \\ &= [\mathbf{z}_j(k+1) - \mathbf{h}_i(\hat{\mathbf{x}}(k+1|k), \mathbf{p}_i)]. \end{aligned} \tag{3.13}$$

The innovation covariance is then calcultated by

$$\mathbf{S}_{ij}(k+1) = \nabla\mathbf{h}_i\mathbf{P}(k+1|k)\nabla\mathbf{h}_i^T + \mathbf{R}_i(k+1). \tag{3.14}$$

A *validation gate* is used to determine the correspondece between predictions and observations:

$$\nu_{ij}(k+1) \leq G. \tag{3.15}$$

This equation is used to test each sensor observation $\mathbf{z}_j(k+1)$ with each predicted measurement $\hat{\mathbf{z}}_i(k+1)$. When a single observation falls in the validation gate, we get a successfull match. Measurements which do not fall in this gate are ignored for localization. The same occurs if a measurement falls in the gate for more then one prediction, or vice-versa.

3.3.5 Estimation

The final step is to use successfully matched predictions and observations to compute $\hat{\mathbf{x}}(k+1|k+1)$, the updated vehicle position estimate. First we build a new vector $\mathbf{z}(k+1)$ containing all the matched observations for time $k+1$ and calculate the composite innovation $\nu(k+1)$. Then we build another vector $\nabla\mathbf{h}$ with all the validated predictions. Using the composite noise vector $\mathbf{R}(k+1)$ we compute the composite innovation covariance $\mathbf{S}(k+1)$ as in Equation 3.14. We then calculate the Kalman filter gain

$$\mathbf{W}(k+1) = \mathbf{P}(k+1|k)\Delta\mathbf{h}^T\mathbf{S}^{-1}(k+1) \tag{3.16}$$

to compute the updated vehicle position estimate

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{W}(k+1)\nu(k+1) \tag{3.17}$$

with associated variance

$$\mathbf{P}(k+1|k+1) = \mathbf{P}(k+1|k) - \mathbf{W}(k+1)\mathbf{S}(k+1)\mathbf{W}^T(k+1). \tag{3.18}$$

3.3.6 Summary

We can summarize the localization cycle by the following steps:

1. $\hat{\mathbf{x}}(k+1|k), \mathbf{P}(k+1|k) \leftarrow$ `vehicle-position-prediction`$(\hat{\mathbf{x}}(k|k), \mathbf{P}(k|k), \mathbf{u}(k), \mathbf{Q}(k))$
2. $Z(k+1) \leftarrow$ `observation`

3. $\hat{Z}(k+1), \nabla\mathbf{h}(k+1) \leftarrow$ `measurement-prediction`$(\hat{\mathbf{x}}(k+1|k), map)$

4. $\nu(k+1), \mathbf{S}(k+1) \leftarrow$ `matching`$(Z(k+1), \hat{Z}(k+1), \mathbf{P}(k+1|k), \nabla\mathbf{h}(k+1), \mathbf{R}(k+1))$

5. $\mathbf{W}(k+1), \hat{\mathbf{x}}(k+1|k+1), \mathbf{P}(k+1|k+1) \leftarrow$ `estimation`$(\hat{\mathbf{x}}(k+1|k), \mathbf{P}(k+1|k), \nu(k+1), \nabla\mathbf{h}(k+1), \mathbf{S}(k+1))$

3.4 System Implementation

We implemented our Kalman filter algorithm for localization using XOberon. In XOberon each program is called a *module* and each module is divided into *procedures*. Our system was mainly consisted of two modules. The first one, called SRLocalization.Mod implements the Kalman filter algorithm. The second module, called ExtractLandmarks.Mod implements the vision procedures to capture images and identify the angles corresponding to the extracted landmarks. Additional modules containing I/O procedures were also implemented. The source code is presented in Appendix A.

To facilitate our description, in this section we are going to follow the sequence presented in the summary of the last section.

The first *a priori* information defined in SRLocalization is the position of our landmarks with respect to our global coordinate system:

$$Map = \begin{cases} p_1 = (1.203m, 3.880m); \\ p_2 = (2.100m, 3.880m); \\ p_3 = (2.733m, 2.870m); \end{cases} \tag{3.19}$$

With respect to the Vehicle Position Prediction, $\hat{\mathbf{x}}(k+1|k)$ is obtained directly from the odometry calculation. This value is accessible in the SmartROB database by means of a predefined object.

With $\hat{\mathbf{x}}(k+1|k)$ we can obtain $dx = \Delta s \cos\theta(k)$, $dy = \Delta s \sin\theta(k)$ and $\Delta\theta$ by just subtracting from $\hat{\mathbf{x}}(k|k)$. We can then update $\nabla\mathbf{f}$.

$\mathbf{Q}(k)$ for each time step k is defined by:

$$\mathbf{Q}(k) = \begin{bmatrix} K_{ss}dx & 0 & 0 \\ 0 & K_{ss}dy & 0 \\ 0 & 0 & K_{s\theta}\Delta s + K_{\theta\theta}\Delta\theta \end{bmatrix} \tag{3.20}$$

where K_{ss}, $K_{s\theta}$ and $K_{\theta\theta}$ are drifting coefficients presented by Crowley in [CC92]. The values of these coefficients were empirically set to be:

$$K_{ss} = 0.01; \quad K_{s\theta} = 0.005; \quad K_{\theta\theta} = 0.01; \tag{3.21}$$

We then calculate $\mathbf{P}(k+1|k)$ using Equation 3.8.

The next step is the Observation. Now we call the module ExtractLandmarks which performs all the image manipulation. For the time being, what we need to know is that it will return a set $Z(k+1)$ of n_O observed angles $\mathbf{z}_j(k+1)$ corresponding to the extracted landmarks. The module ExtractLandmarks will be described in more detail in Chapter 4.

Now comes the Measurement Prediction step. First, with $\hat{\mathbf{x}}(k+1|k)$ and the map, we calculate the n_E expected angles using Equation 3.10 and also $\nabla\mathbf{h}_i$ for $1 \leq i \leq n_E$ from Equation 3.12.

In the Matching step we calculate all the innovations ν_{ij} using Equation 3.13 and apply Equation 3.15 to match the right angles. The validation gate G was empirically set to $G = 0.02rad$.

In the Estimation step, we update the position. If no angles are matched in the previous step then the updated vehicle position $\hat{\mathbf{x}}(k+1|k+1)$ is set to be $\hat{\mathbf{x}}(k+1|k)$ and $\mathbf{P}(k+1|k+1) = \mathbf{P}(k+1|k)$. This means that the position was updated just using information from odometry.

Now it is important to emphasize that our innovation $\nu(k+1)$ and consequently $\mathbf{S}(k+1)$ are one-dimensional. This fact made the calculations much easier because no matrix inversion procedure ($\mathbf{S}^{-1}(k+1)$) is needed.

To calculate $\mathbf{S}_i(k+1)$ we use Equation 3.14 with

$$\mathbf{R}_i(k+1) = [r_{ii}] \tag{3.22}$$

where r_{ii} is the covariance associated with the landmark i.

Then for a matched angle we can calculate the Kalman filter by

$$\mathbf{W}_i(k+1) = \mathbf{P}(k+1|k)\Delta\mathbf{h}_i^T \frac{1}{\mathbf{S}_i(k+1)} \tag{3.23}$$

and also

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{W}_i(k+1)\nu_i(k+1) \tag{3.24}$$

In case of more than one matched angle this procedure is repeated for each one of them separately.

Now we just have to update the robot's position. To do that we just change the attributes of the object odometry for the new values.

We start the algorithm with a $\mathbf{P}(0|0)$, whose elements correspond to the initial uncertainty in the robot's position.

Chapter 4

Vision System

You can observe a lot by just watching.

Yogi Berra

In this chapter we explain in more details our vision system. As seen before, it is composed by a frame grabber and a CCD camera. The algorithms for image processing were developed in XOberon.

4.1 Identifying Landmarks

The first task that our system has to do is to capture an image from the environment. This task is done using predetermined procedures. But before we can fully use them we must perform a preliminary configuration in the frame grabber.

We choose one of its 4 inputs and also select the desired image format. We have chosen the PAL image format with a dimension of 768×288 pixels and an 8-bit grayscale mode which generates 256 levels of brightness.

An image is first captured and stored in the DMA memory. We then transfer it to

the RAM memory for later manipulation. Figure 4.1 shows an example of a raw image captured by the camera.

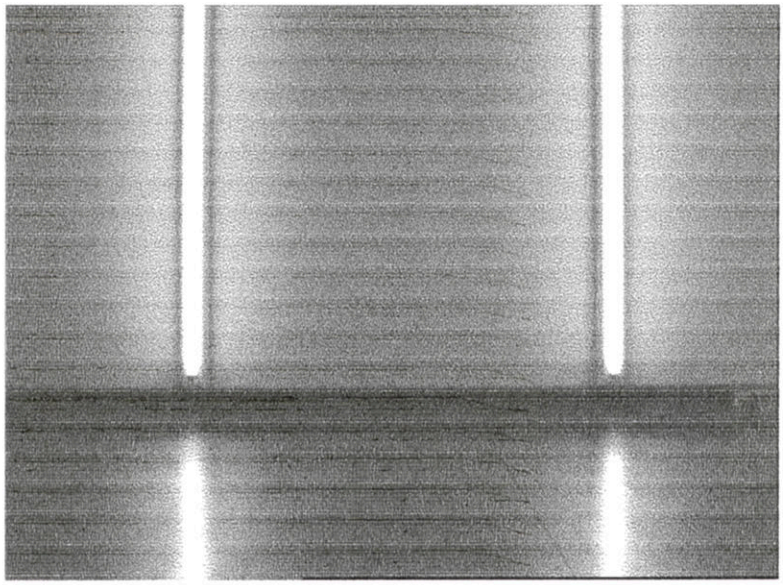

Figure 4.1: Raw Image from Camera

Now that we have the image, we identify in pixel level where the lamps are located. Lamps are sources of light and provide regions of high brightness that characterize them. All we have to do is detect these regions and find the peaks of brightness. To do that we define a *pixel threshold* value. Because we are dealing with sources of light, this value can be considerably high ($e.g\ Th \approx 220$ in a 0-255 scale of brightness).

The algorithm we developed condenses the image information in a single vector with dimension equal to the number of columns of the image. We initialize this vector with zeros. Then for each column of our image, we test the level of brightness of each element. If it is greater than the threshold we add 1 to the corresponding element in our vector. In the end of this process our vector contains all the information we need to detect the peaks of brightness. Figure 4.2 illustrates this vector graphically.

Now we "walk" through this vector and analyze each set of 5 subsequent elements,

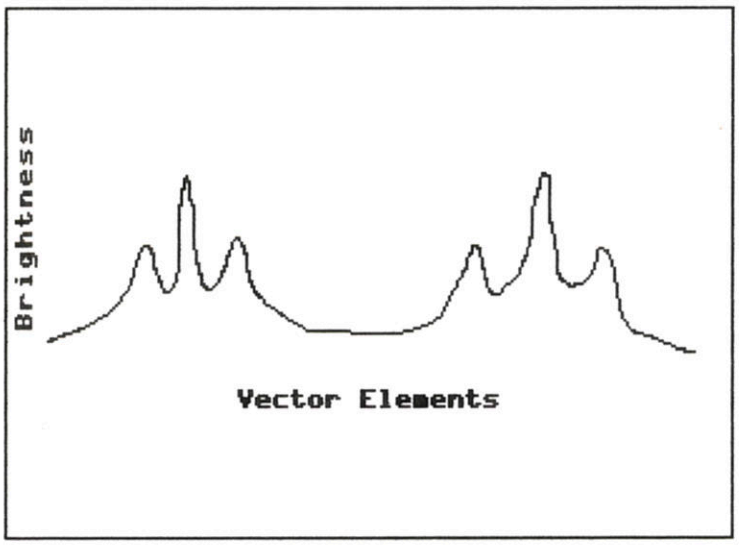

Figure 4.2: Vectorized Image

summing their values. If the result is greater than a *sum threshold* then we define the middle element to be an extracted landmark. Figure 4.3 illustrates this fact.

Recovering the image we obtain the result shown in Figure 4.4.

The image is also saved in GIF format for later use.

Finished the image processing procedures, we still need to convert our eventual extracted landmarks from positions in a vector into angles with respect to the robot's coordinate system. This process is done after a calibration of the camera, where we build a conversion table that associates an angle to each position in our vector.

Summarizing we have the following steps:

1. Capture an image;

2. Define a pixel threshold;

3. Scan the image and convert it into a vector;

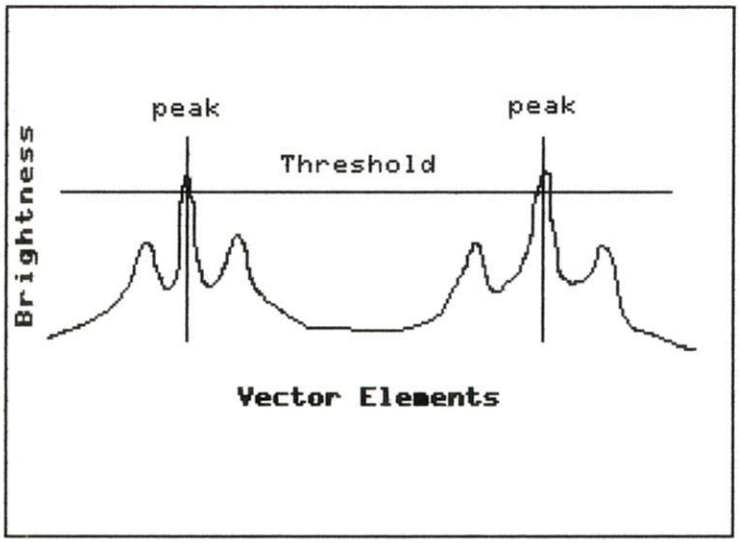

Figure 4.3: Extracted Peaks

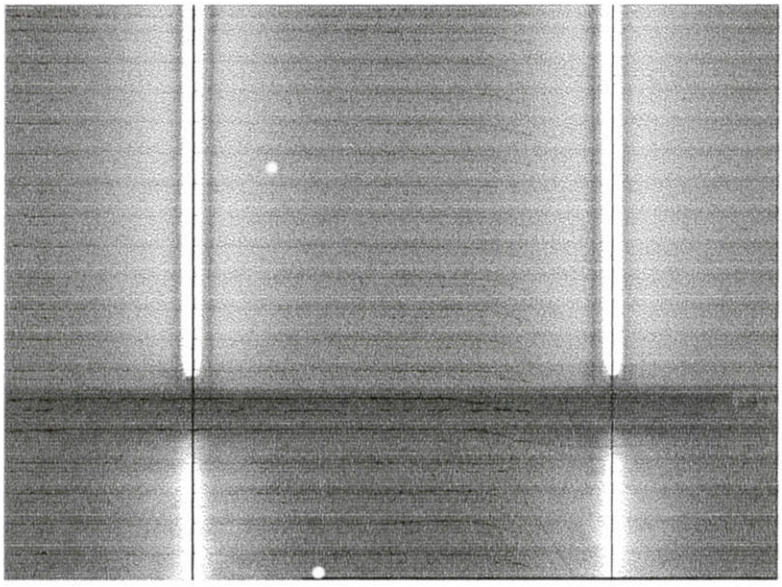

Figure 4.4: Extracted Landmarks

4. Define a sum threshold;

5. Scan the vector and find peaks;

6. Convert eventual peaks into corresponding angles;

Although it seems quite easy to perform this task, when we work with vision we realize that many factors influence our results, such as the reflection of light on objects, the amount of artificial or natural light in the environment, image distortion, the characteristics of the camera and how the camera is tuned. All these conditions make the process of tuning the threshold values something really hard.

Ideally, we would want to have some dynamic procedure that could perform this task in real-time taking into account the instantaneous conditions of the environment. This is not a trivial task, so we have decided to define these values once in the beginning of the process. To be able to do that we assumed that the environmental conditions wouldn't change too much, something that could be questionable in a real situation.

This fact brings some drawbacks. Figure 4.5, for example, shows an image where the reflection of light on the wall generates two regions of high brightness around the lamp. In this case, if we tune our thresholds inadequately (see Figure 4.6), we end up having false observations as shown in Figure 4.7.

Because of these problems we need a good Matching procedure to be able to distinguish correct from false landmarks. In the next section we discuss this point in more detail.

4.2 The Matching Procedure

In the previous chapter we superficially presented a matching procedure algorithm using a validation gate. In our algorithm, after the Measurement Prediction and Observation

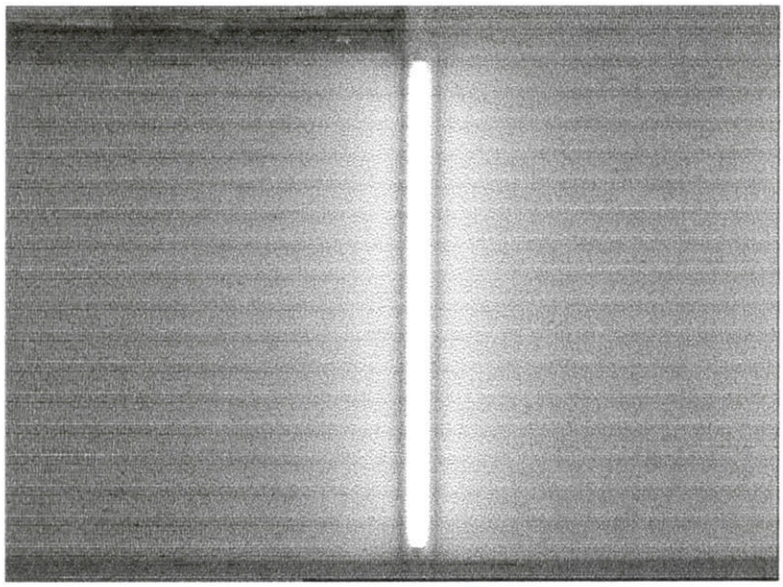

Figure 4.5: Regions of Reflection

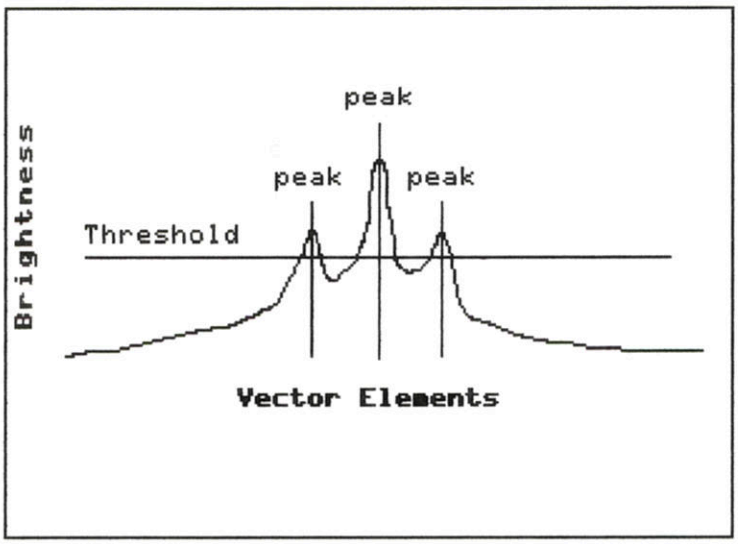

Figure 4.6: Inadequate Threshold Value

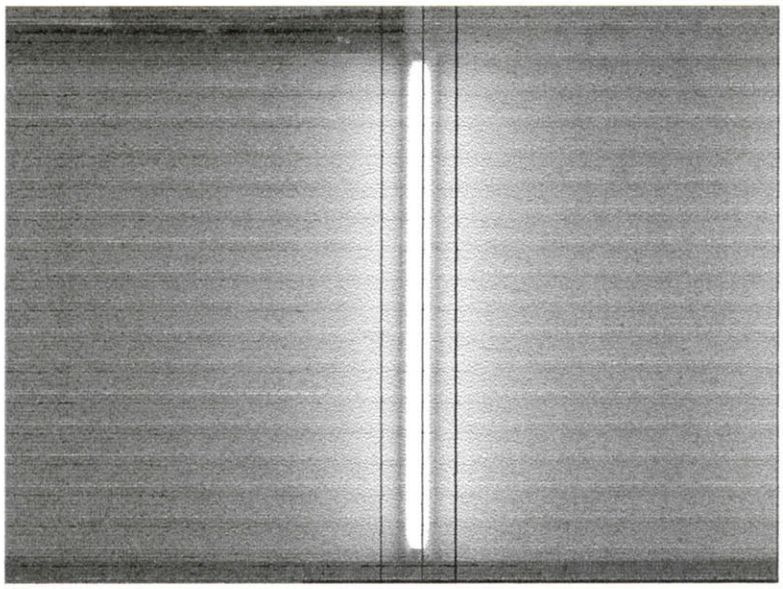

Figure 4.7: False Extraction

steps, we have a set $\hat{Z}(k+1)$ of n_E predicted angles and also a set $Z(k+1)$ of n_O observed landmarks.

The number of elements in these two sets (n_E and n_O) is not necessarily the same, *e.g.* the case we described in the previous section where reflection areas on the wall were identified as landmarks. The following question arises: How do we distinguish correct observations from false ones?

The answer to this question is also a little bit complex. Many authors have already discussed this problem and solutions point to the use of filters that separate false observations from correct ones [BSF88, AG92, LDW92, GA93]. These more elaborated approaches are beyond the scope of our work. Our implementation was simply based on applying Equation 3.15.

The choice of G depends on the accuracy of our odometry and also our vision system, and tells us how far, at most, an extracted landmark can be from a predicted one. If the odometry is precise enough we can make G small, otherwise we have to relax it. This

assumption is quite reasonable because if we set G to a small value we are assuming that an observation has to be close enough to a prediction to be matched. If our odometry is not so precise, we can loose important information just because of this strict choice of G. On the other hand, if we relax G we allow the observations to be in a wider range around our prediction, consequently we have a greater chance of detecting a false observation. Clearly there's always a trade-off between these two things.

4.3 A Word about Camera Calibration

The CCD camera used in our experiment had an objective angle of about 20 degrees. Some calibration procedures developed by Nicola Tomatis at EPFL (Swiss Federal Institute of Technology Lausanne) showed that for this camera the distortion in the images was low enough not to require a correction algorithm. This fact considerably reduced our amount of work.

Chapter 5

Experiment Results

I don't have any solution, but I certainly admire the problem.

Ashleigh Brilliant

When all else fails... read the instructions !

LaTeX help message

In Chapter 3 we presented the theoretical aspects about localization. Having understood the theory behind it we can start talking about a feasible implementation. In this chapter we focus on this practical part. We start with an evaluation of the odometry in the SmartROB and then show results from some experiments. We compare localization only based on odometry with localization based on odometry and vision.

5.1 Evaluating our Odometry

The first step toward testing our system is to evaluate the accuracy in our odometry. We used the UMBmark (University of Michigan Benchmark) test proposed by Borenstein

and Feng [BEF96] to illustrate qualitatively the characteristics of the odometry in the SmartROB.

This test was developed to detect systematic errors like unequal wheel diameters, misalignment of wheels or finite encoder resolution. It is consisted of an experiment where the robot is programmed to run through the four borders of a square path, both in the clockwise and counterclockwise directions. The path will return the vehicle to the starting area but, because of odometry and controller errors, not precisely to the starting point.

Figure 5.1 and Figure 5.2 show the results obtained. The starting point is indicated by an arrow. The blue balls represent the position given by the SmartROB's odometry. The red crosses indicate where the SmartROB really is. We can notice that there's a distortion in the robot's path. Our measurements showed that for a 10-meter run the error in the position was approximately 10 centimeters. Although it seems quite accurate (1% error), it's not precise enough if for example, the robot will be used to perform a task that requires millimeter precision.

Our intention in running this test was just to have some feeling about our odometry, not to improve its characteristics. The reason for that is the fact that our ultimate goal is to make a comparison between the use of odometry alone and its combination with vision. If our odometry were too precise, it would be more difficult to illustrate the improvements added by a vision system or any other sensor.

5.2 Localization using only Odometry

When we perform localization using only odometry, at each time step we estimate the robot's position based on information from the encoders and than associate a correspondent uncertainty to it. With this approach each estimated position is surrounded by a

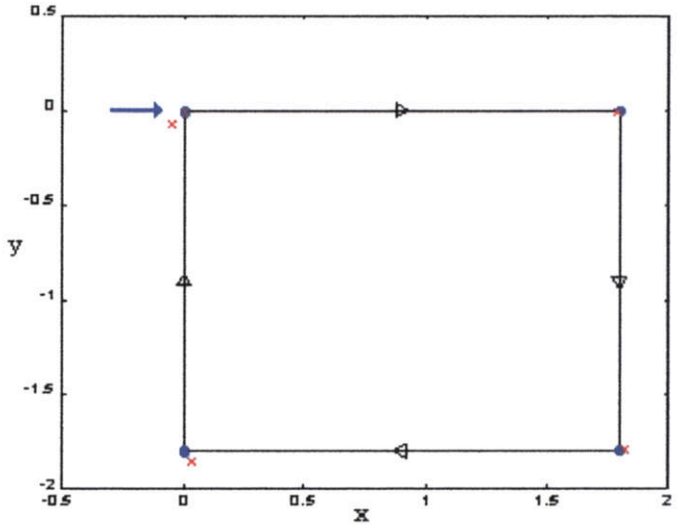

Figure 5.1: UMBmark Test - Clockwise

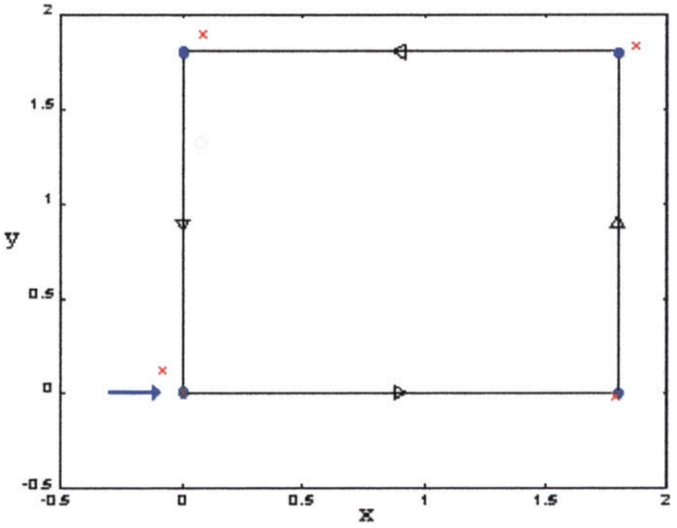

Figure 5.2: UMBmark Test - Counterclockwise

characteristic "error ellipse" which indicates a region of uncertainty for the robot's actual position. Typically, these ellipses grow with travel distance, due to odometry errors. To construct them we used the first four elements of $\mathbf{P}(k|k)$.

Figure 5.3 shows the result obtained with the SmartROB when only odometry is used for localization. The blue crosses represent the position given by the SmartROB's odometry. The red balls indicate the real position of the SmartROB. The theory is confirmed and the robot looses precision with travel distance. In the long-term it will surely get lost.

Another sensor is then required to overcome this problem. In the next section we present the results obtained with our vision system.

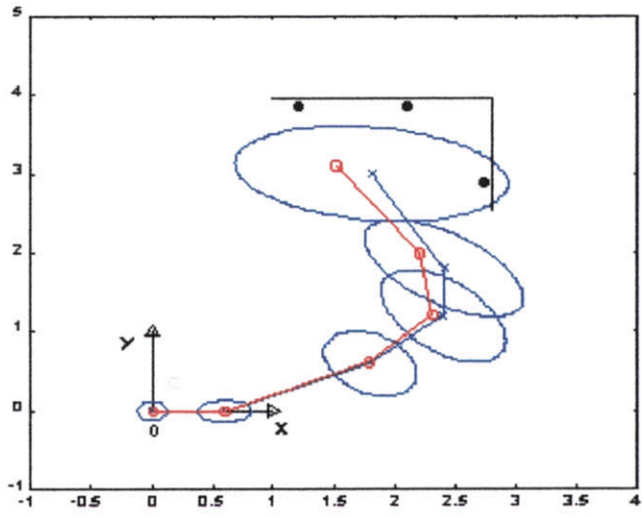

Figure 5.3: Localization using only Odometry

5.3 Localization using Odometry and Vision

In the second run of experiments our vision system was used. To better illustrate the improvements achieved in localization we defined a sequence of steps for each run. The

robot starts at the origin with an initial uncertainty defined by $\mathbf{P}(0|0)$. In each subsequent step we move the robot to a new position and than run the localization cycle to update the position. Figure 5.4 shows the result.

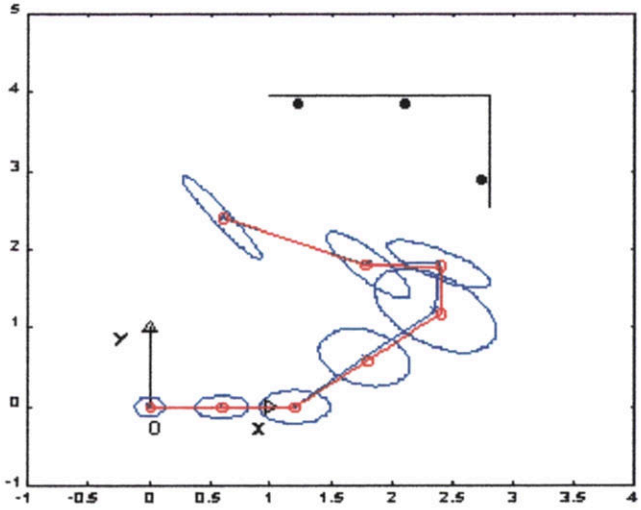

Figure 5.4: Localization using Odometry and Vision

In the first five steps of this path the SmartROB was intentionally placed with such orientation that no landmarks were seen. The result is that only the odometry is used to update the robot's position, therefore the associated uncertainty starts growing.

When the robot first sees a landmark in the sixth step we notice that the position is corrected and the uncertainty is reduced. The same occurs in the next two steps. It means that every time the robot sees a landmark we have more confidence in our position. This confidence is determined by the state covariance matrix $\mathbf{P}(k|k)$ that is used to construct the error ellipses. Our task is to keep $\mathbf{P}(k|k)$ as small as possible. In a real situation the localization algorithm has to be executed continuously and the idea is that the robot be able to see at least one landmark at a time to keep his position accurate.

In our implementation the camera used was fixed on the robot, so if the robot's orientation is such that no landmark can be seen we loose precision in the position. To

overcome this problem a servomotor could be installed in the camera. Then, at each cycle, the camera is rotated to a position where at least one landmark can be seen, therefore the position can be updated.

In a system where we have a precise odometry and also a precise vision system, the uncertainty associated to the actual position should be reduced to very small values. This allows very precise navigation as shown in Figure 5.5.

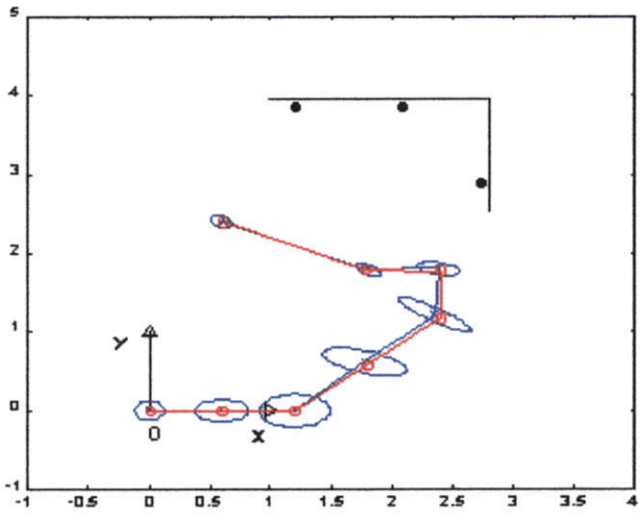

Figure 5.5: Precise Navigation

In our case (Figure 5.4) we can notice that when the robot sees a landmark the uncertainty does not grow and is even a little bit reduced. However, it is not comparable to what we can see in Figure 5.5.

Many reasons led us to these results. The first one is related to our camera. Its fixation on the robot was not mechanically stable, *i.e.* it had a backlash that introduces inaccuracies in the measurements. It means that we have to reduce our confidence with respect to our vision system. Besides that, our Kalman filter used a simple Matching procedure. As we discussed previously, when we relax the value of G we have a greater chance of detecting false landmarks. If we just use Equation 3.15 without any other

criterion of selection, when more then one observation fall in the validation gate we loose precious information.

Besides all the problems in our implementation we were able to navigate the SmartROB quite precisely keeping bounded the uncertainty around the position. The results were satisfactory if we take in account the simplifications we made. It follows that it's possible to perform accurate navigation of mobile robots with a simple approach.

Chapter 6

Conclusion and Outlook

Nothing more than education advances the prosperity,
the power, and the happiness of a nation.

- THOMAS JEFFERSON, (1743-1826)

In this work we have developed a simple localization algorithm for mobile robot navigation. This algorithm was based on odometry and vision. Fluorescent lamps were used as landmarks. A Kalman filter was used to update the robot's position.

The results showed that when we use only odometry for localization, the uncertainty correspondent to the robot's position grows with distance travel. On the other hand when we combine odometry with vision we can keep the uncertainty bounded, allowing precise navigation.

Due to some problems already described our results were not optimal, however we were able to achieve our ultimate goals.

Improvements in our system could be addressed in the following directions:

1. Use a more stable fixation for the camera to increase precision;

2. Develop a more sophisticated matching procedure in the localization cycle;

3. Develop algorithms to dynamically calculate the threshold values;

4. Optimize the code;

An interesting study would be the performance evaluation of the SmartROB navigation system.

Suggestions of new approaches include the use of landmarks that naturally occur in the environment, such as edges of a table, corners of the room, doors, etc. The main idea is that we don't need to change the environment just to be able to have precise localization.

Another suggestion would be the implementation of map building (*i.e.* update and keep an accurate map of the environment) and obstacle avoidance (*i.e* avoid objects in the environment whose location are not predictable). These two problem are also part of a navigation system.

Emphasizing, our implementation was far from being optimal. In fact, it was very simple. But its simplicity could clearly illustrate how a localization algorithm works. Because of its limitations we could see where the problems are and how we should approach them. Implementing a system like the one we did, involves not only a theoretical background but also a practical experience and feeling. The former we can learn by reading books, but the latter we only learn by doing.

Appendix A

List of source programs in XOberon

A.1 MODULE SRLocalization.Mod

```
MODULE SRLocalization; (* last updated 01.03.99 by Hugo *)
(* This module implements a localization algorithm for the SmartROB System, based on Vision *)
(* and Odometry. The Kalman filter notation is taken from [Leonard & Durrant-Whyte, 1992]. *)
(* This color (brown) is used for general description of Procedures, black is used for normal code, *)
(* blue is used for explanations and green is used for commented calculations that were optional. *)
(* Author: Hugo L. Gosmann (gosmann@usa.net). February, 1999. *)

IMPORT
    XT:=XTexts,
    MA:=MathL,
    OdometryDD,
    SRPosCtrl,
    XOberon,
    Base,
    ExtractLandmarks,
    MyOutputs;

CONST
    GATE = 0.04;(* Validation Gate: Maximum distance between observed and expected Landmarks *)

    (* ----- Position of Landmarks ----- *)
    (* ----- My Map ----- *)
    LM1x = 1.203; LM1y = 3.880;
    LM2x = 2.100; LM2y = 3.880;
    LM3x = 2.733; LM3y = 2.870;

    (* ----- Covariances ----- *)
    SigmaZ1 = 0.02; (* 0.02 rad = 1.15 degrees *)
    SigmaZ2 = 0.02;
    SigmaZ3 = 0.02;

    (* Drifting coeficients from [Chenavier & Crowley, 1992] *)
```

```
    Kss = 0.01;
    Kst = 0.005;
    Ktt = 0.01;

VAR
    w: XT.Writer;

    (* Kalman Filter-related variables *)
    x: ARRAY 3 OF LONGREAL; (* State vector of the system *)
    F, (* Jacobian of f, transition function of the system *)
    H, (* Jacobian of h, the measurement model *)
    P: ARRAY 3,3 OF LONGREAL; (* Position Covariance Matrix *)
    W, (* Kalam Filter Gain Matrix *)
    S, (* Innovation Covariance *)
    R: ARRAY 3 OF LONGREAL; (* Measurements Covariance Matrix *)
    innovation: LONGREAL; (* The innovation *)

    (* Vision-related variables *)
    nExpectedLM, (* Number of expected Landmarks *)
    nExtractedLM, (* Number of extracted Landmarks *)
    nMatchedLM: INTEGER; (* Number of matched Landmarks *)
    expectedLM: ARRAY 3 OF LONGREAL; (* Vector containing the expected Landmarks *)
    extractedLM: ARRAY 12 OF LONGREAL; (* Vector containing the extracted Landmarks *)
    matchedLM: ARRAY 3,2 OF INTEGER; (* Vector containing the position of matched Landmarks *)
    matchingMatrix: ARRAY 3,12 OF LONGREAL; (* Cross-Matrix used for matching *)

    (* Odometry and Position Control-related variables *)
    obj: Base.Object;
    myOdometry: OdometryDD.Odom;
    myPosCtrl: SRPosCtrl.Ctrl;

PROCEDURE ArcTan2 (y,x : LONGREAL) : LONGREAL;
CONST
    ARCTANEPSILON = 0.000000001;
VAR
    Value : LONGREAL;
BEGIN
    IFABS(x)<ARCTANEPSILON THEN
      IFy > 0.0 THEN
        RETURN(MA.pi /2.0);
      ELSIF y < 0.0 THEN
        RETURN(-MA.pi/2.0);
      ELSE
        RETURN(0.0);
      END;
    ELSE
      Value := MA.arctan(y/x);
      IF (x < 0.0) & (y > 0.0) THEN
        RETURN(Value+ MA.pi);
      ELSIF (x < 0.0 ) & (y <= 0.0) THEN
        RETURN(Value-MA.pi);
      ELSE
        RETURN(Value);
      END;
```

```
    END;
END ArcTan2;

(* ———————— OUTPUTS ———————— *)
PROCEDURE WriteMatrix(M: ARRAY OF ARRAY OF LONGREAL; dimension: INTEGER);
VAR
    i, j: INTEGER;
BEGIN
    FOR i := 0 TO dimension-1 DO
      FOR j := 0 TO dimension-1 DO
        XT.WriteLongRealFix(w, M[i, j], 10, 10, 0);
      END;
      XT.WriteLn(w);
    END;
    XT.Append(XOberon.Log(), w.buf);
END WriteMatrix;

PROCEDURE WriteVector(v: ARRAY OF LONGREAL; dimension: INTEGER);
VAR
    i: INTEGER;
BEGIN
    FOR i := 0 TO dimension-1 DO
      XT.WriteLongRealFix(w, v[i], 10, 10, 0); XT.WriteLn(w);
    END;
    XT.Append(XOberon.Log(), w.buf);
END WriteVector;

PROCEDURE WriteP*;
VAR
    i, j: INTEGER;
BEGIN
    XT.WriteString(w, "% Covariance Matrix P:"); XT.WriteLn(w);
    WriteMatrix(P,3);
END WriteP;

PROCEDURE WriteX*;
BEGIN
    XT.WriteString(w, "% Robot Current Position (x, y, theta):"); XT.WriteLn(w);
    WriteVector(x,3);
    XT.Append(XOberon.Log(), w.buf);
END WriteX;

(* ———————— KALMAN FILTER ———————— *)
PROCEDURE PositionPrediction*;
(* This procedure performs the vehicle position prediction based on Odometry *)
VAR
    currentX, currentY, currentTheta,
    dx, dy, dtheta,
    accu1, accu2: LONGREAL;
BEGIN
    (* "Calculate" (get from Odometry) the estimate of the current stante *)
    (* x(k+1|k) = f(x(k|k), u(k)) *)
    myOdometry.Get(currentX, currentY, currentTheta);
```

```
    dx := currentX - x[0];
    dy := currentY - x[1];
    dtheta := currentTheta - x[2]; (* currentTheta is allways in the interval [-pi, pi] *)

    (* But dtheta can lie in the interval [-2pi, 2pi], so we have to check it and correct if necessary *)
    IF dtheta > MA.pi THEN dtheta := dtheta - 2*MA.pi;
    ELSIF dtheta < -MA.pi THEN dtheta := dtheta + 2*MA.pi;
    END;

    (* Update the Odometry's model *)
    (* F is the Jacobian of the state transition function f *)
    F[0][2] := -dy; (* -delta y = -T(k) * sin(theta(k)) *)
    F[1][2] := dx; (* delta x = T(k) * cos(theta(k)) *)

    (* Update the Position *)
    x[0] := currentX;
    x[1] := currentY;
    x[2] := currentTheta;

    (* Calculate the estimated P *)
    (* P(k+1|k) = F(k)P(k|k)F'(k) + Q(k) *)
    (* Q describes the uncertanty in the model of the system, taken from [Chenavier & Crowley, 1992] *)
    accu1 := P[0][2] + F[0][2] * P[2][2]; (* Accumulators to speed up the calculation *)
    accu2 := P[1][2] + F[1][2] * P[2][2];

    (* P[1][0], P[2][0] and P[2][1] are not calculated because P is simmetric *)
    P[0][0] := P[0][0] + F[0][2] *(P[0][2] + accu1) + Kss*ABS(dx);
    P[0][1] := P[0][1] + F[0][2] * P[1][2] + accu1 * F[1][2];
    P[0][2] := accu1;
    (* P[1][0] := P[0][1] + P[0][2] * F[1][2] + accu2 * F[0][2]; *)
    P[1][1] := P[1][1] + F[1][2] *(P[1][2] + accu2) + Kss*ABS(dy);
    P[1][2] := accu2;
    (* P[2][0] := accu1; *)
    (* P[2][1] := accu1]; *)
    P[2][2] := P[2][2] + Kst*ABS(MA.sqrt(MA.power(ABS(dx), 2)+MA.power(ABS(dy), 2))) + Ktt*ABS(dtheta);
END PositionPrediction;

PROCEDURE Observation*;
(* This procedure calls the Vision module which performs the extraction *)
(* of the angles corresponding to the Landmarks. *)
BEGIN
    nExtractedLM := 0;
    ExtractLandmarks.FindLightSource(extractedLM, nExtractedLM);
    XT.WriteString(w,"% Angular Position of Extracted Landmarks"); XT.WriteLn(w);
    WriteVector(extractedLM, nExtractedLM);
END Observation;

PROCEDURE MeasurementPrediction*;
VAR
    i, j: INTEGER;
BEGIN
    nExpectedLM := 3; (* Total number of Landmarks *)

    (* Measurement Prediction *)
```

```
    (* z(k+1)=h(x(k+1|k), LM) *)

    (* Calculate the expected observations *)
    expectedLM[0] := ArcTan2(LM1y-x[1], LM1x-x[0]) - x[2];
    expectedLM[1] := ArcTan2(LM2y-x[1], LM2x-x[0]) - x[2];
    expectedLM[2] := ArcTan2(LM3y-x[1], LM3x-x[0]) - x[2];

    XT.WriteString(w,"% Position of Expected Landmarks"); XT.WriteLn(w);
    WriteVector(expectedLM, 3);

    (* Calculate H(k+1) that is the Jacobian of h *)
    H[0][0] := (LM1y - x[1])/(MA.power(ABS(LM1x - x[0]),2) + MA.power(ABS(LM1y - x[1]),2));
    H[0][1] := (-LM1x + x[0])/(MA.power(ABS(LM1x - x[0]),2) + MA.power(ABS(LM1y - x[1]),2));
    H[0][2] := -1;
    H[1][0] := (LM2y - x[1])/(MA.power(ABS(LM2x - x[0]),2) + MA.power(ABS(LM2y - x[1]),2));
    H[1][1] := (-LM2x + x[0])/(MA.power(ABS(LM2x - x[0]),2) + MA.power(ABS(LM2y - x[1]),2));
    H[1][2] := -1;
    H[2][0] := (LM3y - x[1])/(MA.power(ABS(LM3x - x[0]),2) + MA.power(ABS(LM3y - x[1]),2));
    H[2][1] := (-LM3x + x[0])/(MA.power(ABS(LM3x - x[0]),2) + MA.power(ABS(LM3y - x[1]),2));
    H[2][2] := -1;
END MeasurementPrediction;

PROCEDURE Matching*;
VAR
    i, j, noOfMatches, match: INTEGER;
BEGIN
    (* Calculate the distances between the expected and the extracted landmakrs *)
    FOR i := 0 TO nExpectedLM-1 DO
      FOR j := 0 TO nExtractedLM-1 DO
        matchingMatrix[i, j] := ABS(expectedLM[i] - extractedLM[j]);
      END;
    END;
    (* Match Observations with Predicted Landmarks *)
    nMatchedLM := 0;
    (* Check, for each expected LM, all observations, to see if one of them satisfies the condition! *)
    FOR i:=0 TO nExpectedLM-1 DO
      noOfMatches:=0;
      match:=-1;
      FOR j:=0 TO nExtractedLM-1 DO
        IFmatchingMatrix[i,j] < GATE THEN
          INC(noOfMatches);
          match:=j;
        END;
      END;
      (* If exactly one does satisfy then check if this observation does not satisfy for another expected LM *)
      IF noOfMatches=1 THEN
        FOR j:=0 TO nExpectedLM-1 DO
          IF matchingMatrix[j,match] < GATE THEN
            INC(noOfMatches);
          END;
        END;
        (* If noOfMatches equals exactly 2, then the match is valid *)
        IF noOfMatches=2 THEN
          matchedLM[nMatchedLM][0]:=i;
```

```
          matchedLM[nMatchedLM][1]:=match;
          INC(nMatchedLM);
        END;
      END;
    END;
    XT.WriteString(w,"% Number of Matched Landmarks"); XT.WriteLn(w);
    XT.WriteInt(w, LONG(nMatchedLM), 10); XT.WriteLn(w);
    XT.Append(XOberon.Log(), w.buf);
END Matching;

PROCEDURE Estimation*;
VAR
    i: INTEGER;
    accu0, accu1, accu2: LONGREAL;
    rv: LONGINT;
BEGIN
    (* If there are no matches than the position won't be updated *)
    IF nMatchedLM=0 THEN
      XT.WriteString(w, "Position Update Cycle will be bypassed"); XT.WriteLn(w);
      XT.Append(XOberon.Log(), w.buf);
    END;
    FOR i := 0 TO nMatchedLM-1 DO
      * Calculate the innovation *)
      innovation := expectedLM[matchedLM[i][0]] - extractedLM[matchedLM[i][1]];

      (* Calculate P(k+1|k)Hi'(k+1) separately to speed up *)
      accu0 := P[0][0] * H[i][0] + P[0][1] * H[i][1] + P[0][2] * H[i][2];
      accu1 := P[0][1] * H[i][0] + P[1][1] * H[i][1] + P[1][2] * H[i][2];
      accu2 := P[0][2] * H[i][0] + P[1][2] * H[i][1] + P[2][2] * H[i][2];

      (* Calculate Si(k+1) = Hi(k+1)P(k+1|k)Hi'(k+1) + Ri(k+1) - a scalar! *)
      S[i] := H[i][0] * accu0 + H[i][1] * accu1 + H[i][2] * accu2 + R[i];

      IF S[i] < 1.0E-11 THEN
        XT.WriteString(w, "Overflow S["); XT.WriteInt(w,i,1); XT.WriteString(w, "]");
        XT.WriteLn(w); XT.Append(XOberon.Log(), w.buf);
      ELSE
        (* Calculate the Kalman Filter Gain Matrix *)
        W[0] := accu0 / S[i];
        W[1] := accu1 / S[i];
        W[2] := accu2 / S[i];
        (* Calculate P(k+1|k+1) = P(k+1|k) - W(k+1)Si(k+1)W'(k+1) *)
        P[0][0] := P[0][0] - W[0] * accu0;
        P[0][1] := P[0][1] - W[0] * accu1;
        P[0][2] := P[0][2] - W[0] * accu2;
        P[1][0] := P[1][0] - W[1] * accu0;
        P[1][1] := P[1][1] - W[1] * accu1;
        P[1][2] := P[1][2] - W[1] * accu2;
        P[2][0] := P[2][0] - W[2] * accu0;
        P[2][1] := P[2][1] - W[2] * accu1;
        P[2][2] := P[2][2] - W[2] * accu2;
        (* Update the Position *)
        x[0] := x[0] + W[0]*innovation;
        x[1] := x[1] + W[1]*innovation;
```

```
        x[2] := x[2] + W[2]*innovation;
        (* Update Odometry *)
        myOdometry.Set(x[0], x[1], x[2]);
        rv := myPosCtrl.SetNewTarget(x[0], x[1], x[2], 0.0);
      END;
    END;
END Estimation;

PROCEDURE Localization*;
BEGIN
    PositionPrediction;
    WriteX;
    MyOutputs.WritePosition(888, x, P);
    Observation;
    MeasurementPrediction;
    Matching;
    Estimation;
    WriteX;
    MyOutputs.WritePosition(999, x, P);
END Localization;

PROCEDURE SavePosition999*;
BEGIN
    MyOutputs.WritePosition(999, x, P);
END SavePosition999;

PROCEDURE Go*;
VAR
    rv: LONGINT;
BEGIN
    rv := myPosCtrl.SetNewTarget(1.0, 1.0, 0.0, 0.0);
END Go;

BEGIN
    XT.OpenWriter(w);

    (* F is the Jacobian of f, state transition function of the system *)
    F[0][0] := 1.0; F[0][1] := 0.0; F[0][2] := 0.0; (* F[0][2] := -dy *)
    F[1][0] := 0.0; F[1][1] := 1.0; F[1][2] := 0.0; (* F[1][2] = dx *)
    F[2][0] := 0.0; F[2][1] := 0.0; F[2][2] := 1.0;

    (* H is the Jacobian of h, the mesurement model *)
    H[0][0] := 0.0; H[0][1] := 0.0; H[0][2] := 0.0;
    H[1][0] := 0.0; H[1][1] := 0.0; H[1][2] := 0.0;
    H[2][0] := 0.0; H[2][1] := 0.0; H[2][2] := 0.0;

    (* P (simmetric matrix) is zero if initial position of robot is perfect. This is definitely not the case *)
    P[0][0] := 0.05*0.05;
    P[0][1] := 0;
    P[0][2] := 0;
    P[1][0] := 0;
    P[1][1] := 0.05*0.05;
    P1][2] := 0;
    P[2][0] := 0;
```

```
    P[2][1] := 0;
    P[2][2] := 0.05*0.05;

    (* R describes the uncertanty in the measurements *)
    R[0] := SigmaZ1*SigmaZ1;
    R[1] := SigmaZ2*SigmaZ2;
    R[2] := SigmaZ3*SigmaZ3;

    (* S is the Innovation Covariance Matrix *)
    S[0] :=0;
    S[1] :=0;
    S[2] :=0;

    (* Kalman Filter Gain Matrix *)
    W[0] := 0.0;
    W[1] := 0.0;
    W[2] := 0.0;

    (* x is the state vector of the system *)
    x[0] := 0.0;
    x[1] := 0.0;
    x[2] := 0.0;

    (* Grab the object "OdomSR" from the database into "obj" and test if it is *)
    (* of the type OdometryDD.Odom, if so, asign it to MyOdometry *)
    Base.GetObj("OdomSR", obj);
    IF obj IS OdometryDD.Odom THEN myOdometry:=obj(OdometryDD.Odom) ELSE HALT(99) END;

    (* Grab the object "SRPCTRL" from the database into "obj" and test if it is *)
    (* of the type SRPosCtrl.Ctrl, if so, asign it to MyPosCtrl *)
    Base.GetObj("SRPCTRL", obj);
    IF obj IS SRPosCtrl.Ctrl THEN myPosCtrl:=obj(SRPosCtrl.Ctrl) ELSE HALT(99) END;
END SRLocalization.
```

A.2 MODULE ExtractLandmarks.Mod

```
MODULE ExtractLandmarks; (* last updated 12.02.99 by Hugo *)
(* This module implements the extraction of landmarks using the Vision System. *)
(* Author: Hugo L. Gosmann (gosmann@usa.net). February, 1999. *)

IMPORT
    XTexts,XOberon,XWebServer,DMAImages,Bt848,VisionDMA,FastVision,GreyscaleGIF,Files;

CONST
    Columns = Bt848.PALColumns;
    Lines = Bt848.PALLines DIV 2;

VAR
    w: XTexts.Writer;
    server: XWebServer.Server;
    Result: INTEGER;
```

```
    Image1, Image2: ARRAY Columns*Lines OF INTEGER;
    Sum: ARRAY Columns OF INTEGER;

PROCEDURE VisionProblem;
BEGIN
    XTexts.WriteString(w,"Vision Problem: "); XTexts.WriteInt(w,Result,0);
    XTexts.WriteString(w,"(for details: ET.Open Bt848Errors.txt)"); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(),w.buf);
    HALT(99);
END VisionProblem;

PROCEDURE ResetFrameGraber*;
BEGIN
    Bt848.Reset;
    Bt848.SetInput(Bt848.IformMux1, Result);
      IF Result # 0 THEN VisionProblem END;
    Bt848.SetVideoFormat(Bt848.PALVideoFormat, Result);
      IF Result # 0 THEN VisionProblem END;
    Bt848.SetMode(Bt848.ForceMode, Result);
      IF Result # 0 THEN VisionProblem END;
    Bt848.SetCaptureField(Bt848.CodeA,Bt848.ANYField,Bt848.Gray8Color,DMAImages.ImageAdrA, Result);
      IF Result # 0 THEN VisionProblem END;
    Bt848.SetCaptureField(Bt848.CodeB,Bt848.ANYField,Bt848.Gray8Color,DMAImages.ImageAdrB, Result);
      IF Result # 0 THEN VisionProblem END;
END ResetFrameGraber;

PROCEDURE SaveGIF*(VAR image: ARRAY OF INTEGER; name: ARRAY OF CHAR);
VAR
    f: Files.File;
    w: Files.Writer;
BEGIN
    f:=Files.New();
    IF f # NIL THEN
      Files.OpenWriter(w,f,0);
      GreyscaleGIF.Make(w,image,Columns,Lines,8);
      Files.CloseWriter(w);
      Files.Register(f,name);
      (* RETURN TRUE *)
    END;
    (* RETURN FALSE *)
END SaveGIF;

PROCEDURE FindThreshold(X,Y: INTEGER; VAR Image: ARRAY OF INTEGER):INTEGER;
BEGIN
    RETURN 220
END FindThreshold;

PROCEDURE ThresholdingAndSum(X, Y, Th: INTEGER; VAR InOutImage: ARRAY OF INTEGER);
    VAR i: LONGINT;
BEGIN
    i := X*Y-1;
    WHILE i >= 0 DO
        IF InOutImage[i] > Th THEN INC(Sum[i MOD X]) END;
        DEC(i)
```

```
    END
END ThresholdingAndSum;

PROCEDURE VisualizePeak(X, Y, y: INTEGER; VAR InOutImage: ARRAY OF INTEGER);
    VAR i: LONGINT;
BEGIN
    i := y;
    WHILE i < X*Y DO
        InOutImage[i] := 0;
        i := i + X
    END;
END VisualizePeak;

PROCEDURE FindLightSource*(VAR ExtractedLM: ARRAY OF LONGREAL; VAR nExtractedLM: INTEGER);
CONST
    PeakTh = 1000;
VAR
    th, i, oldpeak, newpeak: INTEGER;
    peak: BOOLEAN;
BEGIN
    XTexts.WriteString(w, "Start capture..."); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(), w.buf);
    Bt848.StartCapture(Bt848.CodeA, Result); IF Result # 0 THEN VisionProblem END;

    XTexts.WriteString(w, "Wait end capture..."); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(), w.buf);
    Bt848.WaitEndCapture(Bt848.CodeA, Result); IF Result # 0 THEN VisionProblem END;

    XTexts.WriteString(w, "Copy image from DMA to RAM..."); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(), w.buf);
    VisionDMA.DMAtoRAM(Columns, Lines, DMAImages.ImageArrayA, Image1);

    XTexts.WriteString(w, "Save input image as debug1.gif..."); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(), w.buf);
    SaveGIF(Image1, "debug1.gif");

    XTexts.WriteString(w, "Find threshold for the image and apply it..."); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(), w.buf);
    th := FindThreshold(Columns, Lines, Image1);
    ThresholdingAndSum(Columns, Lines, th, Image1);

    peak := FALSE;
    nExtractedLM := 0;
    i := 2;
    newpeak := Sum[0]+Sum[1]+Sum[2]+Sum[3]+Sum[4];
    WHILE i < Columns-4 DO
        INC(i);
        oldpeak := newpeak;
        newpeak := newpeak - Sum[i-3] + Sum[i+2];
        (* IF newpeak > 0 THEN XTexts.WriteInt(w, newpeak,0); XTexts.WriteLn(w);*)
        (* XTexts.Append(XOberon.Log(), w.buf) END; *)
        IF newpeak > PeakTh THEN
          IF (oldpeak > newpeak) &  peak THEN
            peak := TRUE;
```

```
            XTexts.WriteString(w, "Peak at y = "); XTexts.WriteInt(w, i-1,0); XTexts.WriteLn(w);
            XTexts.Append(XOberon.Log(), w.buf);
            VisualizePeak(Columns, Lines, i-1, Image1);
            nExtractedLM := nExtractedLM+1;
            ExtractedLM[nExtractedLM-1] := 0.0003205*(384-(i-1));
          ELSIF (oldpeak < newpeak) & peak THEN
            peak := FALSE
          END
        END
    END;

    XTexts.WriteString(w, "Save result as debug2.gif..."); XTexts.WriteLn(w);
    XTexts.Append(XOberon.Log(), w.buf);
    SaveGIF(Image1, "debug2.gif");

    (* XTexts.WriteString(w, "Save thresholded image as debug3.gif..."); XTexts.WriteLn(w);*)
    (*XTexts.Append(XOberon.Log(), w.buf); *)
    (* FastVision.Thresholding(Columns, Lines, th, Image1); *)
    (* SaveGIF(Image1, "debug3.gif"); *)

    XTexts.WriteString(w, "...Ok!"); XTexts.WriteLn(w); XTexts.Append(XOberon.Log(), w.buf);
END FindLightSource;

PROCEDURE Extraction*;
VAR
    ExtractedLM: ARRAY 12 OF LONGREAL;
    i,nExtractedLM: INTEGER;
BEGIN
    FindLightSource(ExtractedLM, nExtractedLM);
    FOR i := 0 TO nExtractedLM-1 DO
        XTexts.WriteLongRealFix(w, ExtractedLM[i], 10, 5, 0); XTexts.WriteLn(w);
        XTexts.Append(XOberon.Log(), w.buf);
    END;
END Extraction;

BEGIN
    XTexts.OpenWriter(w);
    server := XWebServer.FindServer(80);
    XWebServer.RegisterType(server, "admin", "BMP", "image-x/bmp");
    ResetFrameGraber;
END ExtractLandmarks.
```

Bibliography

[Ada99] M. D. Adams. *Sensor Modelling, Design and Data Processing for Autonomous Navigation.* World Scientific Publishing, 1999.

[AG92] M. A. Abidi and R. C. Gonzalez. *Data Fusion in Robotics and Machine Intelligence.* Academic Press, 1992.

[Arr96] K. O. Arras. Map building. Diploma Thesis, Swiss Federal Institute of Technology Zurich, February 1996.

[BEF96] J. Borenstein, H. R. Everett, and L. Feng. *Navigating Mobile Robots.* Wellesley, Massachusetts, 1996.

[Bre98] R. Brega. A real-time operating system designed for predictability and runtime safety. In *The Fourth International Conference on Motion and Vibration Control - MOVIC'98*, volume 1, pages 379–384, Zurich, Switzerland, August 1998.

[BSF88] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association.* Academic Press, 1988.

[CC92] F. Chenavier and J. L. Crowley. Position estimation for a mobile robot using vision and odometry. In *IEEE International Conference on Robotics and Automation*, pages 2588–2593, Nice, France, May 1992.

[CK97] K. S. Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *IEEE International Conference on Robotics and Automation*, pages 2783–2788, Albuquerque, New Mexico, April 1997.

[CSCD97] M. Cicerone, E. Stella, L. Caponetti, and A. Distante. Visual landmarks recognition for autonomous robot navigation. In *Intelligent Robots and Computer Vision XVI*, pages 133–139, Pittsburgh, Pennsylvania, October 1997.

[GA93] M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice.* Prentice-Hall, 1993.

[GBFK98] J-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 736–743, Victoria, B.C., Canada, October 1998.

[HGB97] H. Hu, D. Gu, and M. Brady. Outdoor navigation of a mobile robot with multiple sensors. In *SPIE's International Symposium on Intelligent Systems Advanced Manufacturing*, pages 14–17, Pittsburgh, Pennsylvania, USA, October 1997.

[Hig93] N. J. Higham. *Handbook of Writing for the Mathematical Sciences.* Society for Industrial and Applied Mathematics, 1993.

[Hor86] B. K. P. Horn. *Robot Vision.* MIT Electrical Engineering and Computer Science Series. MIT Press, 1986.

[HS92] R. M. Haralick and L. G. Sharipo. *Computer and Robot Vision*, volume I. Addison-Wesley Publishing Company, 1992.

[HS93] R. M. Haralick and L. G. Sharipo. *Computer and Robot Vision*, volume II. Addison-Wesley Publishing Company, 1993.

[Kro89] E. Krotkov. Mobile robot localization using a single image. In *IEEE International Conference on Robotics and Automation*, pages 978–983, 1989.

[Lam94] L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley Publishing Company, second edition, 1994.

[LDW92] J. J. Leonard and H. F. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation.* Kluwer Academic Publishers, London, 1992.

[May90] P. S. Maybeck. The Kalman filter: An introduction to concepts. In I. J. Cox and G. T. Wifong, editors, *Autonomous Robot Vehicles*, pages 194–204. Springer-Verlag, 1990.

[Mös93] H. Mössenböck. *Object-Oriented Programming in Oberon-2.* Springer-Verlag, 1993.

[MW92] H. Mössenböck and N. Wirth. The programming language Oberon-2. Technical report, Swiss Federal Institute of Technology Zurich, May 1992.

[Pap98] I. P. Pappas. *Adaptive Visual Control for Microrobots.* PhD thesis, Swiss Federal Institute of Technology Zurich, 1998.

[Rei91] M. Reiser. *The OBERON System.* Addison Wesley, New York, 1991.

[RW96] G. X. Ritter and J. N. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra.* CRC Press, 1996.

[SBB98] R. Y. Siegwart, R. Büchi, and P. Bühler. Mechatronics education at ETH Zurich based on 'hands on experience'. In *The 6th Mechatronics Forum International Conference - Mechatronics'98*, Skövde, Sweden, September 1998.

[SD98] R. Sim and G. Dudek. Mobile robot localization from learned landmarks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1060–1065, Victoria, B.C., Canada, October 1998.

[Str86] G. Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, 1986.

[Str88] G. Strang. *Linear Algebra and its Applications.* Harcourt Brace Jovanovich, third edition, 1988.

[Tom98a] N. Tomatis. The PMC-FG framegrabber: A bt848-based capture device for the XOberon/PowerPC operating system. Technical report, Swiss Federal Institute of Technology Lausanne, 1998.

[Tom98b] N. Tomatis. Vision feedback for mobile robots. Diploma Thesis, Swiss Federal Institute of Technology Zurich, February 1998.

[Ves95] S. J. Vestli. *Fast, accurate and robust estimation of mobile robot position and orientation.* PhD thesis, Swiss Federal Institute of Technology Zurich, 1995.

[Ves98] S. J. Vestli. Introduction to basic XOberon services. Swiss Federal Institute of Technology Zurich, October 1998.