

# COMPUTING EGO-MOTION AND STRUCTURE FROM POINT FEATURES

eingereichtes  
PROJEKTPRAKTIKUM  
von

Bertrand Bourguignon

geb. am 04.08.1985

wohnhaft in:

Georgenstrasse 99  
80798 München

Tel.: 0176 86234261

Stefan Ehrlich

geb. am 21.06.1984

wohnhaft in:

Connollystr. 11/H18  
80809 München

Tel.: 0172 7467102

Carrhel Kouam Kamwa

geb. am 04.02.1986

wohnhaft in:

Hiltenspergerstraße 84  
80796 München

Tel.: 089 303639

Lehrstuhl für  
STEUERUNGS- UND REGELUNGSTECHNIK  
Technische Universität München  
Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss  
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer/-in:

Prof. Dr. sc. nat. Jörg Conradt

Beginn:

27.10.2009

Abgabe:

29.01.2010



## **Abstract**

Orientation within a spatial structure could simply be implemented by tracking point features in images captured by a moving video source. Due to perspective changes, provided by the moving video source, it is possible to infer the motion of the camera, as well as the position of the tracked features.

This work describes the development and evaluation of a system practically using this method. Based on a small mobile robot, the aim was to implement appropriate algorithms in order to prove the practical functionality of “Ego-motion and Structure from Point Features”.

The developed system is equipped with a feature tracking algorithm that detects and tracks coloured objects.

Wheel sensors, attached to the robot are used to do a rough measurement of the robot’s position. Both tracked features and rough robot position are used by a localization algorithm to calculate the feature’s as well as the camera’s position.

The developed system was evaluated under optimized laboratory conditions. It showed a good functionality with limited accuracy. This work gives explanations for the lack of precision and proposes measures to optimize the entire system.



TECHNISCHE UNIVERSITÄT MÜNCHEN  
**LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK**  
 ORDINARIUS: UNIV.-PROF. DR.-ING./UNIV. TOKIO MARTIN BUSS  
 EXTRAORDINARIA: UNIV.-PROF. DR.-ING. SANDRA HIRCHE



27.10.2009

PRACTICAL COURSE  
 for  
 Bertrand Bourguignon, Mat.-Nr. 3209862  
 Stefan Ehrlich, Mat.-Nr. 3613152  
 Carrhel Kouam Kamwa, Mat.-Nr. 3061713

### Computing Ego-Motion and Structure from Point Features

#### Problem description:

When viewing video footage from a non-stationary source, it is possible to infer the motion of the camera, as well as the structure of the viewable scene, simply by tracking point features in the image. Such calculations are commonly referred to as solving the "ego-motion and structure" problem, which has attracted much attention over the years. We have recently developed a very simple and efficient algorithm that is designed to be practical for real-time applications with limited hardware resources that require a continually-maintained estimate of position and/or spatial structure. Good examples of such systems are small and inexpensive mobile robots.

The algorithm performs visual point tracking in a circular 360deg field of view, and updates its feature position estimates with every new observation. In a second pass it relies on the feature position estimates to update its ego motion estimate. So far we have only implemented the algorithm in a simulated environment; however, we believe that the algorithm is sufficiently simple to ultimately run on a standard microprocessor (32bit, 60MHz) on board of small mobile robots.

#### Tasks:

In this "Practical Project" the student(s) will get a mobile robot with an omni-directional circular camera that initially connects to the local computer network through WLAN. In a first stage, the student(s) will implement the algorithm in C on a Linux computer and evaluate its real-world performance. During a second stage the student(s) will convert the algorithm to run on a microcontroller (still in C) on-board of the mobile robot. Such an algorithm running on tiny on-board hardware with low communication latencies will ultimately allow very fast robot motion for small robots equipped with cheap sensors.

#### Bibliography:

- [1] J. Gluckman, and S.K. Nayar. Ego-motion and omnidirectional cameras. ICCV, pages 999-1005, 1998.
- [2] S. Se, D. Lowe, and J. Little. Global localization using distinctive visual features. Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), pages 226:231, 2002.

Supervisor: Jörg Conradt

  
 (J. Conradt)  
 Univ.-Professor

---

# Contents

<b>1</b>	<b>Introduction.....</b>	<b>7</b>
1.1	Detailed Task Description .....	7
1.2	Scientific Motivation .....	8
<b>2</b>	<b>Main Part.....</b>	<b>10</b>
2.1	Development of the entire system .....	10
2.1.1	General Overview .....	10
2.1.2	WIFI-Communication.....	11
2.1.3	Robot Control .....	12
2.1.4	Robot position measurement .....	12
2.1.5	Camera Data Processing .....	13
2.1.6	Camera Data Analysis .....	15
2.1.7	Localization Algorithm.....	18
2.2	Evaluation of the entire system.....	26
2.2.1	Laboratory Setup.....	26
2.2.2	Investigation 1 (robot) .....	27
2.2.3	Investigation 2 (localization algorithm).....	28
2.2.4	Investigation 3 (featuretracking).....	30
2.2.5	Investigation 4 (trajectory).....	32
<b>3</b>	<b>Summary.....</b>	<b>33</b>
3.1	Critique and conclusion of the essential results.....	33
	<b>Appendix.....</b>	<b>37</b>
	<b>List of Figures.....</b>	<b>43</b>
	<b>Bibliography .....</b>	<b>45</b>

## List of symbols and abbreviations

LAN	Local Area Network
WLAN	Wireless Local Area Network
WIFI	“Wireless Fidelity”
mm	Millimeter
m <sup>2</sup>	square-meter
V	Volt
PC	Personal Computer
Hz	Hertz





# 1 Introduction

## 1.1 Detailed Task Description

During this project, we had to implement an algorithm in C on a Linux system to control the robot of Figure 1.1 (a). The robot has a diameter of 200mm and is 260mm high. It consists of 3 multi-directional wheels with incremental encoders, two microcontrollers, a WLAN card, a CMOS camera and a tube on which a spherical mirror is mounted.

The 3 multi-directional wheels are up 120 degrees from each other. This allows a translational movement in the x and y-direction and a rotational movement around the robot center. On the basis of incremental encoders, the wheels rotation can be recorded. The wireless card allows communicating with the robot via WLAN, and sending different commands.

The sensor used in the system is a CMOS color camera (Electronics, VS6524). The camera resolution is 640x480 pixels. Images of size 480x480 pixels can be recorded at a frequency of 9Hz in a memory. The typically applications of this camera can be found in mobile phone and video phone.

The core of the system is a spherical mirror attached to a tube, which is the “eye” of the robot. Namely, the camera is oriented toward the spherical mirror to get a 360 panoramic view from the current robot environment. The camera capture of the spherical mirror is shown in Figure 1.1: mobile robot (a); spherical mirror (b) (b).

The aim of the introduced project is –based only on this mobile robot with cheap camera sensor- to implement an algorithm that allows the robot to analyze its environment. Namely, in a scene that includes features (obstacles or landmarks) with different colors and in different ratio, the robot will move and take pictures of its environment. The function of the algorithm is to track this features and to estimate their position. The image sequence of the spherical mirror of the moving robot will be use to extract information about the features. Then using this information, the algorithm will compute the features positions. With the help of this features positions, the robot can orient itself in its environment and, for example, return to its start position.

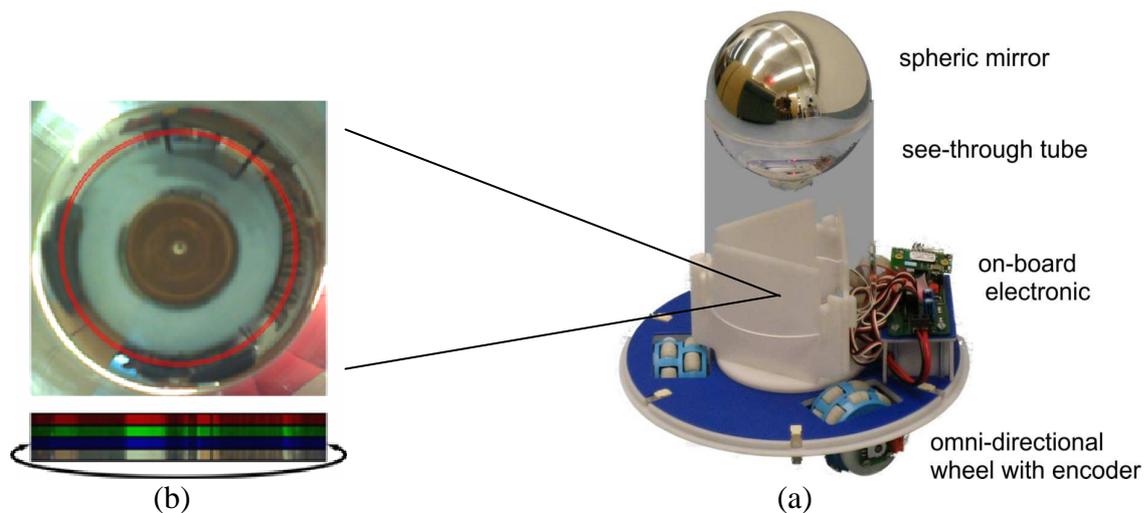


Figure 1.1: mobile robot (a); spherical mirror (b)

## 1.2 Scientific Motivation

Analyzing the environment to get spatial information is one of the most important tasks of autonomous systems. The autonomy of robot vehicles, for example cars or flying objects, depends on the collected spatial information and its interpretation. Almost all types of service robots, the smallest household robot as well as complex humanoid robots need spatial information to orientate and to fulfil their tasks. Industrial robots as well can manage by far more difficult task with help of spatial information. Acquisition and interpretation of the spatial environment is the basis of adaptive robotic systems that work together with humans.

Today, various sensor systems are used to capture spatial information. The most important types are listed below:

- Laser Range Scanner
- Ultrasonic Distance Sensor
- Stereo Camera Systems

It depends on the application which sensor systems or combination of several sensor systems fits best to get the required quality of spatial information.

Laser Range Scanners afford a big range (up to  $\sim 18\text{m}$ ) and a high precision (error:  $\leq 20\text{mm}$ ) for a relative high costs ( $\sim 5000\text{€}$ ). A range of  $270^\circ$  around the sensor system can be captured quasi-simultaneously. The laser light source as well as the mechanical components inside the sensor system leads to a high energy consumption (typically  $\sim 10\text{-}20\text{W}$ ). [1]

Ultrasonic Distance Sensors are cheaper but less precise than Laser Range Scanner. The echo of the emitted spherical sonic waves allows getting information about the distance but not about the precise angle of the detected object. Environmental influences like humidity and air turbulences can strongly influence the measurement results.

Stereo Camera Systems are relative cheap and afford a high precision of the captured spatial information. However, the precision depends on the camera's resolutions and the complexity of the used algorithms to interpret the pictures. The limitations of this sensor system strongly depend on available computing power, which is limited by the application.

The method that is described in this thesis offers a new opportunity to capture and interpret the spatial environment. The method is characterized by its simplicity and provides several advantages compared to methods using typical sensor systems.

The camera and the emissive sphere allow the capturing of 360° around the sensor system (robot). Hence, a changing of the sensor system's position in order to capture other important parts of the spatial environment is not necessary. The picture coming from the camera is being reduced to its essence by cutting the most important part. The reduced picture is used by an interpreting algorithm in order to detect and track distinctive objects (features) around the robot. Changing of the robot's perspective due to its movement provides sequential data that is used by a localization algorithm to calculate the feature's and its own (the camera's) position within the spatial environment.

Due to the camera data reduction the algorithms to interpret the picture and calculate the localization can be designed very simple. An algorithm that does not use high computing power can be integrated on a simple microcontroller. This feature leads to several advantages like cost savings and the opportunity to miniaturize the whole sensor system. Another advantage is the reduced consumption when using a simple microcontroller. Less power consumption leads to an increase in the sensor system's mobility and its autonomy.

In terms of the data reduction and the simplification of the interpreting algorithm the precision of the method is limited and not as high as the precision of Laser Range Scanners. The method is particularly interesting to be used in applications where high precise sensor systems are too expensive and typical cheap sensor systems are too imprecise.

Applications are possible in the context of simple household robots. Household robots that are used for example to clean the floor are typically equipped with so called "bumpers" that helps them to orientate roughly. The sensor system, described in this thesis could be compared with the "bumpers" in terms of manufacturing costs. Upgrading a household robot with the opportunity to use spatial orientation would highly improve its efficiency.

## 2 Main Part

### 2.1 Development of the entire system

#### 2.1.1 General Overview

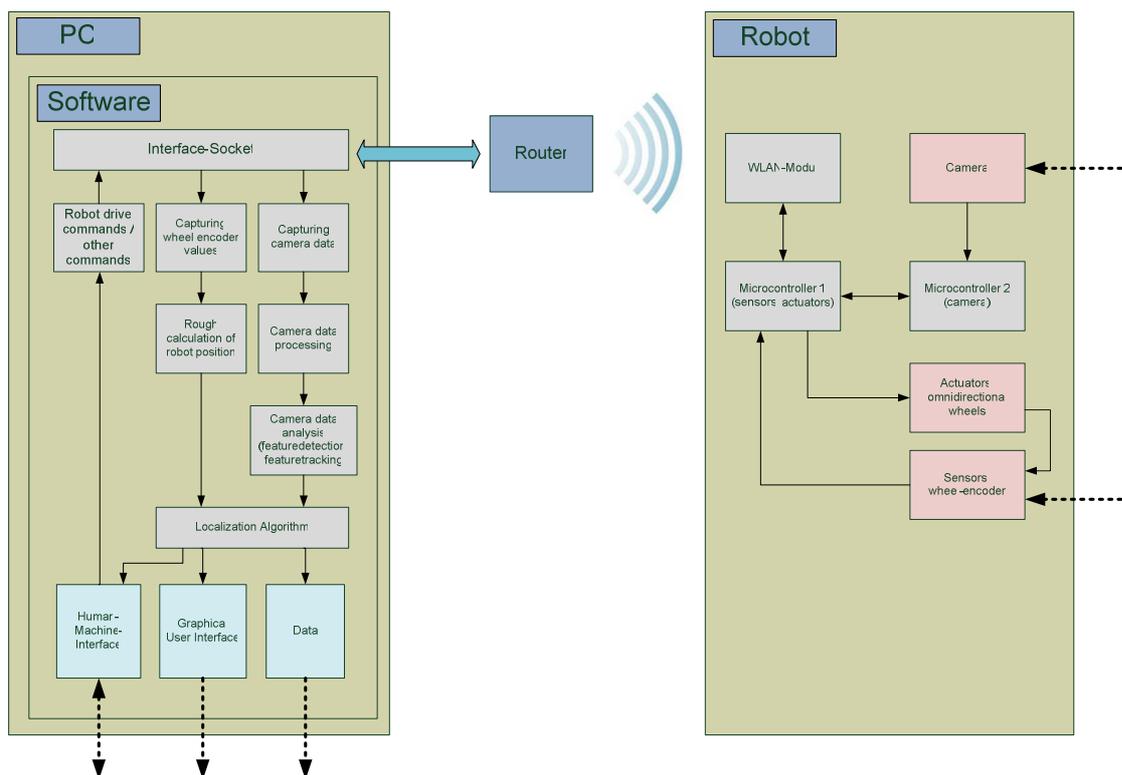


Figure 2.1: Functional diagram of the entire system

The entire system consists of two main components: The robot and the Software to control the robot. The software is embedded on an external PC that communicates with the robot via Local Area Network (LAN).

The robot is equipped with two microcontrollers, one microcontroller for the sensors and actuators one microcontroller for the camera. The robot is further equipped with a WLAN-Module to be able to communicate via LAN. The central unit is microcontroller 1. A communication with microcontroller 2 is only possible via microcontroller 1. The robot is only able to accept several commands afforded by the implemented protocols on microcontroller 1 and 2. In order to use the mobile robot as a sensor system additional software is required.

The external software consists of several subcomponents. The interface socket is the unit that afford data exchanges with the robot via LAN. It is used to send different commands to the robot in order to control its behaviour. Furthermore the interface socket is used to receive the requested data from the robot. The software requests for the wheel encoder values and data from the camera. Both incoming data is being processed. The wheel encoder values are being used to calculate the current robot position roughly. The camera data is used for detecting and tracking distinctive features around the robot. Both processed data is used by the algorithm to estimate the position of the tracked distinctive features and the current position of the robot. Important data is being showed by the human-machine-interface and the graphical user interface. The computed data is also being exported as a text-file. The human-machine-interface is further usable to control the robot via abstract commands.

### 2.1.2 WIFI-Communication

To make the communication between the robot and the computer easier, it has been necessary to develop a basic Software. The basic Software consists of all the functions implemented in the project. These functions can be divided into 3 categories with different objectives: first, to establish a wireless connection between computer and robot; they are grouped under a single socket. Another purpose is to control the robot wheels (wheel control function). The third objective is to measure the robot position.

#### Socket

As shown in Figure 2.1 the Socket ensures the information flow between the robot and the computer via a wireless router. The Socket works according to the client- server principle. The client is here the computer and the server the robot. A simple example of the communication between them is shown in Figure 2.2, too. The socket is implemented in C and represents a quantity of functions that are used to establish and parameterize the connection between the robot and the computer and if the task was made, will be closed. The commands that are included in these functions are transferred via the wireless router to the robot. Using its WLAN cards, the robot can receive and executes them.



Figure 2.2: Client – Server Communication

### 2.1.3 Robot Control

The control functions send commands to the robot which cause the movement of the robot in a space. This functionality is implemented by several functions (see appendix: Software/Omnimotion/RobotInterface.c).

- **driveforward ()**: This function sends a command to the robot to move forward
- **drivebackward ()**: This function can move the robot backwards
- **rotation ()**: The robot can perform this function to rotate
- **freeze ()**: The freeze () function stops the movement of the robot

### 2.1.4 Robot position measurement

As reminder the aim of this work is to compute the structure from point features with none stationary source (ego-motion) and to estimate their positions within an image sequence. The measurement of the current camera position (robot position) in world coordinates plays a major role in the estimation of these positions. Namely the algorithm uses the recursive least square approximation and the position of the robot is used as start value of this approximation. The robot's position is computed by the **robotposition ()** function. Namely the current value of the robot's encoders is constantly interrogated and incremented while driving the robot. Based on these encoder values and a simple mathematical approach, the position of the camera is evaluated and passed on to the estimation algorithm. The recording of images occurs simultaneously with robot's movement. The algorithm thus receives data input packets consisting of an image sequence together with the corresponding camera position recorded during the trip. An example of an image sequence is shown in Figure 2.3. The points represent the positions where the images and the camera positions were stored.

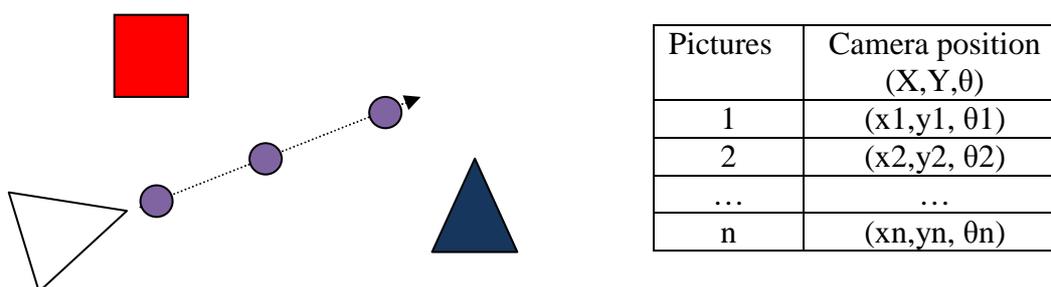


Figure 2.3: Robot position measurement

## 2.1.5 Camera Data Processing

The camera data processing comprises getting pictures from the robot's camera and processing them to be usable for further application. This functionality is implemented by two functions (see attached CD: C-Code (software)/Omnimotion/RobotInterface.c).

- `getCamData()`
- `DataToPic()`

Detailed flow diagrams to both functions are attached in the appendix (see appendix A, B)

### 2.1.5.1 `getCamData()`

The function `getCamData()` is used to get the data of one picture from the robot's camera controller.

Before requesting for one picture several adjustments has to be done: Normally the camera controller sends the data of one whole captured picture (640x480 pixels). As described in Chapter 1.1 (Detailed Task Description) the camera controller is able to extract the most interesting part of the picture through cutting a ring consisting of 256x12 pixels. In order to get the ring-data instead of the whole-picture-data the camera-controller needs to be adjusted.

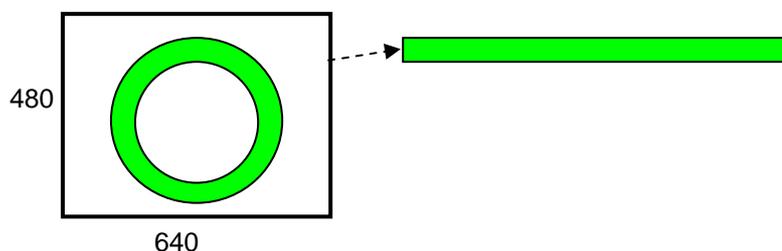


Figure 2.4: Extraction of the ring

Another adjustment that has to be done before requesting for one picture is the camera offset setup. In terms of manufacture tolerances of the camera and the emissive sphere it is most probable that the cut ring is not well-positioned. That leads to a skewed picture and therefore to the problem that interesting information around the robot would not be viewable correctly. An adjustment is done through redefinition of the position where the ring is cut out.

After all adjustments have been done a request for one picture is sent. The function `getCamData()` thereupon waits for incoming data. As soon as it receives one picture is returns the data.

### 2.1.5.2 DataToPic()

The function `DataToPic()` is used to convert the incoming camera data into a “real” picture consisting of pixels. The data format of the raw camera data is arranged as follows:

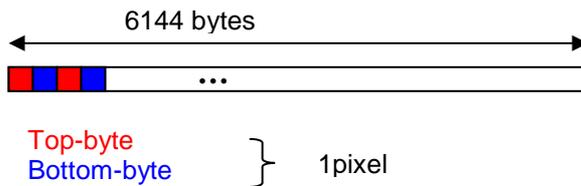


Figure 2.5: raw camera data

In the first step the pixel value is calculated from the top- and the bottom-byte. The pixel value contains information about the pixel’s colour.

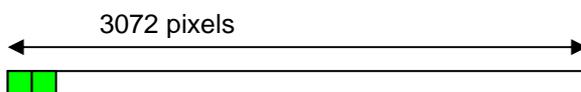


Figure 2.6: calculation of the pixels

Referring to the ring that was cut out by the camera controller the pixels do not have the correct order. In order to use the data as a ring in a spread format a reordering of the pixels has to be done. Therefore two lookup tables, one for the x-position, one for the y-position are used to bring the pixels into the right order.

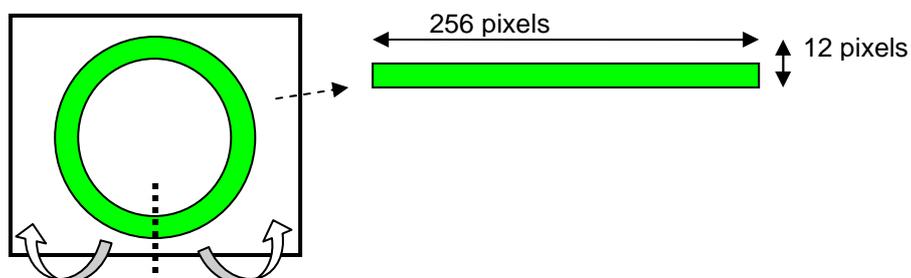


Figure 2.7: Rebuilding and spreading of the ring

The returned picture is usable for instance to show it on the screen or to do further processing.

## 2.1.6 Camera Data Analysis

In order to analyse and interpret the robot's view of the environment it is necessary to introduce some conventions first.

It does not make sense to aspire correct interpretations of the environment around the robot in every possible viewable scene. In contrary it is reasonable to prove the practical functionality of the localization algorithm first under defined optimized artificial laboratory conditions. These definitions allow focusing the development of the interpretation algorithms.

The defined laboratory conditions are described as follows:

- The viewable scene has an expansion of maximal 2 m<sup>2</sup> and is protected from external influences like light sources or irritating objects with the help of walls.
- The scene is well illuminated.
- The obstacles are immobile.
- The obstacles that help the robot to orientate are coloured red, green or blue.
- The obstacles are higher than at least 100mm
- The obstacles has a diameter of at least 50mm and maximum 200mm

The analysis and interpretation of the viewable scene around the robot is being carried out in two steps. The first step is the analysis of the picture in order to find distinctive points or objects. The second step is to assure that a distinctive point or object will be recognized in future pictures. This is implemented by the following two functions (see appendix: C-Code (software)/Omnimotion/Featuretracking.c).

- `featuredetection()`
- `featuretracking()`

Detailed flow diagrams to both functions are attached in the appendix (see appendix C, D)

### 2.1.6.1 Featuredetection()

The function `featuredetection()` is used to analyze the picture in order to find distinctive points or objects. The feature detection that is implemented is specialized in the identification of coloured objects.

In the first step three new pictures are built using the incoming picture and contain the information of the picture's red- green and blue-intensity. The following steps are done for each of the three pictures (red, green and blue).

In order to separate interesting parts of the picture that could be an indication for distinctive objects a segmentation is done. This is carried out by a comparison of each pixel's intensity with the average intensity of the whole picture. A pixel becomes the property "threshold positive" if its intensity exceeds a defined limit which depends on the average intensity.

An additional noise filter could be optionally used in this part of the function to eliminate isolated threshold positive pixels.

After getting regions that could represent an interesting object these regions first have to be classified as valid. The validation is done by measuring the regions expansion in x- and y-direction of the picture. After classification of the region as valid the feature extraction can be done by calculation of the region's center. The calculated value is the angle under which the robot "sees" the detected object.

The angle and the colour of the detected feature are written into a list of features that is returned by the function.

### 2.1.6.2 Featuretracking()

The function `featuretracking()` is used to recognize features that were already detected in past pictures. To handle new detected features and the problem that features could be lost are also important tasks which are implemented in that function. The loss of an object could naturally be effected by changing of the robot's perspective due to its movement. A detected object can disappear (see Figure 2.8: blue object) for a moment because of being covered by another object (see Figure 2.8: red object).

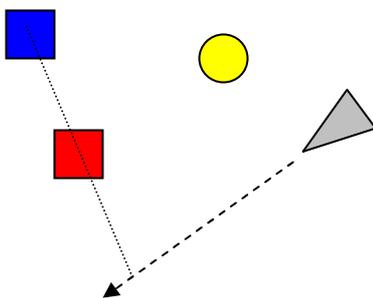


Figure 2.8: Temporary loss of a feature

With respect to that problem a lost feature will be kept “alive” for a short time in order to keep the possibility to redetect it.

In the first call of the functions no tracked features are available. Because of that fact all detected features are written into a new featurelist that contains the tracked features. All copied features get the certainty-level of 1.

In the following calls of the function each detected feature is compared with each tracked feature. If the colour is equal and the position similar the detected feature is considered as recognized. Within the list of tracked features the position of this feature is updated and its certainty-level is incremented.

New detected features are added to the list of tracked features and get a certainty-level of 1.

Old features that are lost for one moment loose one certainty-level. This is done step-by-step until the level attenuated to 0. Then the feature is marked with an illegal position value that shows that it is absolutely lost and could never be recognized.

The tracked features are returned by the function in a list of features.

#### Preparation of the data for the localization algorithm

In order to calculate the positions of the objects around the robot the localization algorithm needs tracked features from a sequence of captured pictures. This sequence is generated through collecting the data of a limited number of pictures all over the way the robot moves through the scene (see Figure 2.9: green dots).

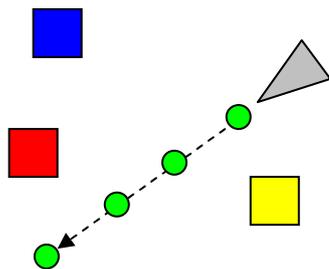


Figure 2.9: Collecting data

The collected data is arranged as follows:

Picture	Feature1	Feature2	Feature3	...	Feature n
1	colour1, $\Phi_1$	colour2, $\Phi_2$	colour3, $\Phi_3$	...	colour n, $\Phi_n$
2	colour1, $\Phi_1$	colour2, $\Phi_2$	colour3, $\Phi_3$	...	colour n, $\Phi_n$
3	colour1, $\Phi_1$	colour2, $\Phi_2$	colour3, $\Phi_3$	...	colour n, $\Phi_n$
...	...	...	...	...	...
m	colour1, $\Phi_1$	colour2, $\Phi_2$	colour3, $\Phi_3$	...	colour n, $\Phi_n$

Figure 2.10: Collected data

## 2.1.7 Localization Algorithm

### 2.1.7.1 Problem description

The problem we have to solve is as follows: our robot moves in an unknown environment and has to determine **positions of obstacles** seen by the feature tracking and the **position of the camera/robot** itself. Indeed, the concept of ego-motion is the “computation of camera motion from a sequence of images” [2]. This sequence is built with pictures taken during the movement of the robot and will be used to deduce the positions of both features and camera.

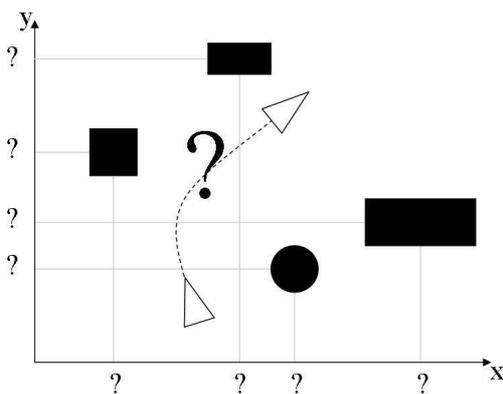


Figure 2.11: Schematic view of a short move in a 4-obstacles environment

The obstacles are supposed immobile, each one bring therefore 2 unknowns (abscissa and ordinate) in the problem. The camera move and we have to be permanently able to estimate its position/orientation, i.e. we must be able to calculate its trajectory (dashed line).

To find a solution we have to use the available measurements. These are the angles under which the features are seen and come as output of the feature tracking. Figure 2.12 shows those measurements in the case where the camera detects 2 features.

Note that sensors on the robot's wheels might be able to give a piece of information about the position of the camera. These data must however be carefully processed: their values depends strongly on the environment (type and inclination of the floor...etc).

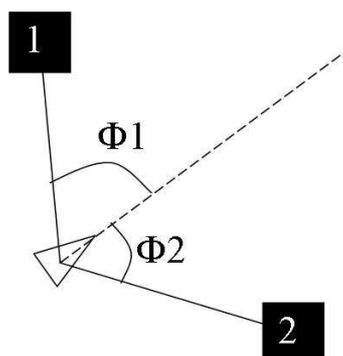


Figure 2.12: Data measured by the feature tracking

### 2.1.7.2 Problem modelling

The coordinates of the camera are stored in a 3-dimensional vector:  $(X\_Cam, Y\_Cam, \theta\_Cam)$ . Those are the abscissa, ordinate and angle to the origin, which is taken at the start point of the camera. These 3 components are together called “pose” of the camera. Each Feature is localized by its abscissa and ordinate:  $(X\_Feat, Y\_Feat)$  in the same coordinate system.

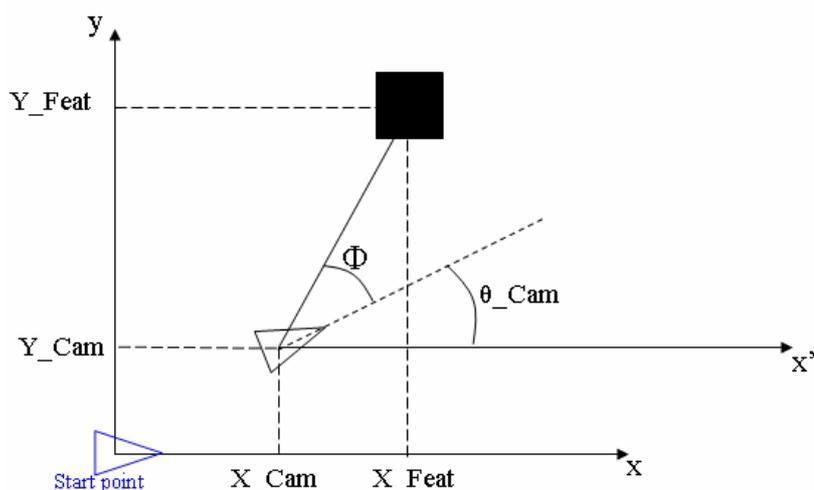


Figure 2.13: Schematic representation of all parameters for a given feature at a certain position

For the k-th feature, at the j-th position (actually the position corresponding to the j-th picture saved in the sequence of the feature tracking), we have the following relation:

$$\Phi_{j,k} = \text{atan2}(Y\_Feat[k]-Y\_Cam[j], X\_Feat[k]-X\_Cam[j]) - \theta\_Cam[j] \quad (2.1)$$

With respectively N and M (valid) features and pictures, our system reaches the size of **M\*N equations** (k=1..N and j=1..M).

The unknown parameters are X\_Feat, Y\_Feat for N features and X\_Cam, Y\_Cam,  $\theta\_Cam$  for M pictures. We store all unknowns in a big unknown vector:

$$p = (X\_Feat[1:N], Y\_Feat[1:N], X\_Cam[1:M], Y\_Cam[1:M], \theta\_Cam[1:M]) \quad (2.2)$$

As the unknown vector's size is **2N+3M**, we will get a unique solution only if

$$2N + 3M \leq MN \quad \text{i.e.} \quad \boxed{M \geq \frac{2N}{N-3}} \quad \text{with} \quad N \geq 4 \quad (2.3)$$

*Note that the condition  $N \geq 4$  limits our algorithm to environment containing 4 features or more*

### 2.1.7.3 Solution of the equation system

We are now facing the following mathematical problem:

$$\underline{f}(p) = \Phi \quad (3.1)$$

where  $\underline{f}$  is a nonlinear function from  $\mathbb{R}^{2N+3M}$  to  $\mathbb{R}^{M \times N}$ .

Solving such a problem with an analytical method is never easy, considering that the minimal number of equations is reached for N=4 (which implies M=8, see (2.1)), and is equal to  $4 \times 8 = 32$ . Moreover, the measured values of  $\Phi$  are subject to errors and uncertainties, which exclude an exact solution. The method to use is therefore an optimization method which can solve **nonlinear multidimensional** problems. Methods based on **least squares approximation** seem to be well adapted to such a problem: even with over-determined systems (happens when  $MN > 2N+3M$ ), these methods will try to find the solution which best fits the data.

However, such methods need a start point for the unknown vector p, otherwise the algorithm could get stuck at a local minimum, far from the real solution. This is where we use the data from the wheels sensors, giving starting values for X\_Cam[1:M], Y\_Cam[1:M] and  $\theta\_Cam[1:M]$ . Starting points for X\_Feat[1:N] and Y\_Feat[1:N] can then be found from these values and  $\Phi$  (see simulation).

### 2.1.7.4 Simulation with Matlab

In order to get acquainted with the resolution of the problem and the behavior of the algorithm in this particular case, we first programmed a simulation with Matlab. We simulated features and several successive positions of the camera. The trajectory, number/positions of features and number of camera poses were completely customizable. To simulate the uncertainties/perturbations of the real system, we added white noise to all components of the pose of the camera.

In Matlab, a function that performs the least squares approximation like we want it to is the *lsqnonlin* function. We let this function run with different configurations, and numerous times with the same configuration to estimate the influence of the white noise (=perturbations). For each simulation, the real trajectory differs from the programmed one because of white noise. We calculate the angle under which feature is seen from the position of the feature and the (real) pose of the camera. We then run *lsqnonlin* with our nonlinear equation system (3.1) as input. Finally, the simulation plots (with animations) the trajectory of the robot, returns the real position of the features, the first evaluation of these positions (see below) and the positions after least squares optimization.

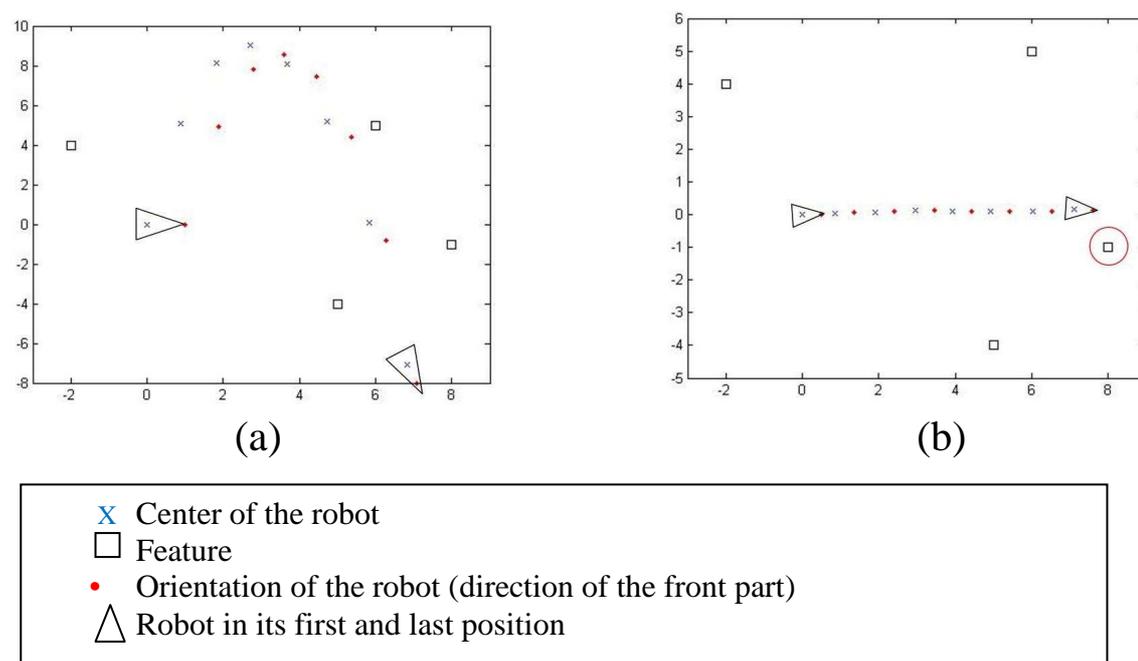


Figure 2.14: Simulation with a “complex” trajectory (a); Simulation with a straight line trajectory (b)

The position of each feature is first evaluated from 2 poses of the camera (these poses are deduced from the wheel sensors in real case). This first evaluation is critical and can lead to aberrations if the algorithm gets stuck at a local minimum. In the previous examples, the results (particularly the position of the red-circled feature of Figure 2.14) were **worse in the straight line case**, no matter which points of the trajectory were used for the evaluation. Least squares approximation is then able to calculate new values of

the position by considering the whole trajectory. Final values present (in almost all cases) an improvement in comparison to the first evaluation.

### How the features positions are first evaluated

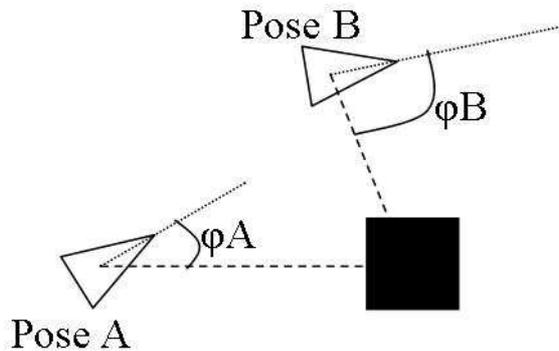


Figure 2.15: First evaluation of the position

To evaluate the position of the feature from 2 poses A and B of the trajectory, we apply the formula:

$$\tan(\theta_A + \varphi_A) = \frac{Y_{Feat} - Y_{Cam\_A}}{X_{Feat} - X_{Cam\_A}}$$

$$\tan(\theta_B + \varphi_B) = \frac{Y_{Feat} - Y_{Cam\_B}}{X_{Feat} - X_{Cam\_B}}$$

We can then deduce the expression of  $X_{Feat}$  and  $Y_{Feat}$  in function of the other parameters.

### How to select a good first-position-evaluation for each feature

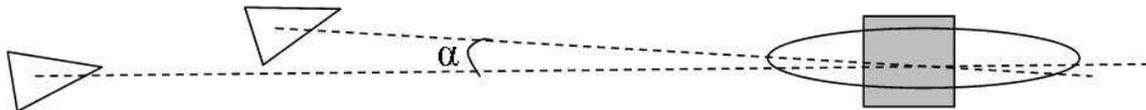


Figure 2.16: Bad configuration for the first evaluation of the position of a feature

Figure 2.16 shows bad conditions to evaluate the position of the feature: a slight error on the orientation of the camera or due to the feature tracking can have a huge influence on the resulting position of the feature.

In the 2<sup>nd</sup> case (Figure 2.17), the conditions are much better:  $\text{abs}(\alpha) \approx 90^\circ$

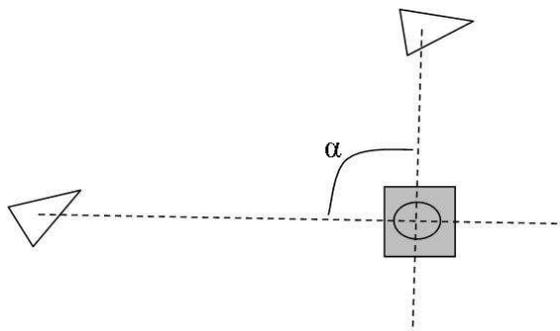


Figure 2.17: Better configuration for the first evaluation

If we use the previous denominations (with poses A and B) the expression of  $\alpha$  is the following:

$$\alpha = (\theta_B + \varphi_B) - (\theta_A + \varphi_A)$$

With a straight line trajectory, the optimal condition ( $\text{abs}(\alpha) \approx 90^\circ$ ) can hardly be reached, particularly when a feature is situated on the axis of this line (see Figure 2.14 (b)), this is why it is strongly recommended to let the algorithm run after a complex trajectory. Once the first evaluation of each feature position can be “correctly” evaluated (the deduced coordinates must have the same sign as the real coordinates and their absolute value must not exceed the one of the real coordinate by more than 200%), then the performance of the least squares algorithm can be judged satisfying (error < 15%). With a more complex trajectory, we can get closer to the criteria  $\text{abs}(\alpha) \approx 90^\circ$  and the error can then easily falls under 10%.

### 2.1.7.5 Implementation in C

#### Global structure of the localization algorithm

In our simulation, it was easy to get data to work with: we just simulated the outputs of feature tracking and wheel sensors in an adapted format. In the real case, we had to integrate the localization algorithm with other components of our system. Three steps of the algorithm are important to understand its global functioning: filter feature tracking data, first estimation of features positions (as we did in the simulation) and least squares

approximation. Figure 2.18 shows these three steps and the inputs/outputs of the localization algorithm.

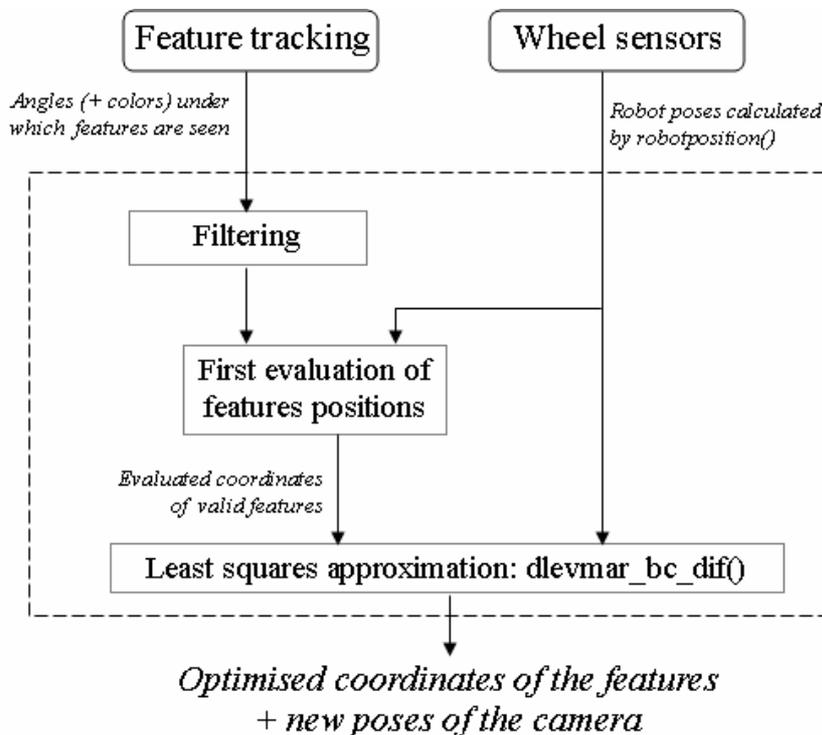


Figure 2.18: Global structure of the localization algorithm

The function performing the least squares approximation was chosen in a library called `levmar` [3]. To compile the code, the BSD-licensed package `LAPACK` is required, as well as `BLAS` (Basic Linear Algebra Subprograms) and eventually `ATLAS` (Automatically Tuned Linear Algebra Software). We selected `dlevmar_bc_dif()` instead of other functions because its format corresponds to what we are trying to solve (like `lsqnonlin` in Matlab) and it lets the possibility to set lower and upper bounds to the solution, preventing the algorithm from converging to absurd values in case of wrong estimation before the least squares approximation.

### Filter data from feature tracking

Although our feature tracking already selects the features which are more likely to correspond to real obstacles, the localization algorithm cannot work with “invalid” data. Indeed, even during a short run, all features are not always visible, feature tracking then returns a corresponding alternative value (777 or 999, see 2.1.6 Camera Data Analysis). This is why we first filter the returned values from feature tracking. Figure 2.19 shows

how we proceed: we first eliminate features which are too often absent (grey cells, features  $k-1$  and  $k+1$ ) and then, given the remaining features, we keep the pictures (= array's lines) containing only valid values (normal/white cells). On Figure 2.19, pictures 1 and 2 have been deleted (strike) because feature  $k$  was not yet identified (bold) by feature tracking, in the same way picture  $M$  has been deleted because of the disappearance of feature 2. At the end of filtering, we send all valid values (normal cells), as a new array (corresponding to the measured values of  $\Phi$ ), to the least squares function.

	Feature 1	Feature 2	...	...	Feature $k-1$	Feature $k$	Feature $k+1$	...
Picture 1	$\Phi_{1,1}$	$\Phi_{1,2}$	...	...	999	<b>999</b>	999	
Picture 2	<del><math>\Phi_{2,1}</math></del>	<del><math>\Phi_{2,2}</math></del>			<del><math>\Phi_{2,k-1}</math></del>	<b>999</b>	999	
...	...	...			...	$\Phi_{3,k}$	999	
...	...	...			...	...	...	...
Picture $j-1$	$\Phi_{j-1,1}$	$\Phi_{j-1,2}$			$\Phi_{j-1,k-1}$	$\Phi_{j-1,k}$	999	
Picture $j$	$\Phi_{j,1}$	$\Phi_{j,2}$			$\Phi_{j,k-1}$	$\Phi_{j,k}$	999	
Picture $j+1$	$\Phi_{j+1,1}$	$\Phi_{j+1,2}$			777	$\Phi_{j+1,k}$	999	
...	...	...			...	...	...	...
Picture $M-1$	$\Phi_{M-1,1}$	$\Phi_{M-1,2}$	...	...	777	$\Phi_{M-1,k}$	999	
Picture $M$	<del><math>\Phi_{M,1}</math></del>	<del>777</del>	...	...	<del>777</del>	<del><math>\Phi_{M,k}</math></del>	<del><math>\Phi_{M,k+1}</math></del>	

Figure 2.19: Filtering applied to an array returned by feature tracking

Nevertheless, deleting invalid features results of arbitrary choices: how many times must a feature be seeable (resp. absent) to consider it as a valid (resp. invalid) feature? By eliminating too many features, the result could be incomplete (real obstacles missing), or the number of seen features could be too low (must be  $> 3M/(M-2)$  according to (2.1)) and prevent least squares function from working. On the contrary, by not eliminating enough features, the number of pictures can be too low (must be  $> 2N/(N-3)$ ).

We created a parameter called `MISSING_FEATURES_COUNTER` as a tolerance limit for each feature, before being regarded as an invalid feature. When the number of invalid pictures for a feature becomes greater than `MISSING_FEATURES_COUNTER`, this feature is deleted from the array. The right value of `MISSING_FEATURES_COUNTER` depends of course on the sensitivity of the feature tracking and on the number of pictures  $M$  returned. In our experiment, we tested different values of `MISSING_FEATURES_COUNTER`,  $M$  and feature tracking sensitivity.

### Least squares approximation

dlevmar\_bc\_dif() uses the **Levenberg-Marquardt** method to minimize the sum of squares of nonlinear functions. Our problem is the following (see (3.1), (2.1) and (2.2)):  $\underline{f}(p) = \Phi$

$$\text{with } \left\{ \begin{array}{l} p = ( \quad X\_Feat[1:N], \quad Y\_Feat[1:N], \quad X\_Cam[1:M], \quad Y\_Cam[1:M], \\ \theta\_Cam[1:M] ) \\ \underline{f}_{j,k}(p) = \text{atan2}(Y\_Feat[k]-Y\_Cam[j], X\_Feat[k]-X\_Cam[j]) - \theta\_Cam[j] \\ \quad = \text{atan2}(p[N+k] - p[2N+M+j], p[k] - p[2N+j]) - p[2N+2M+j] \\ \quad = \Phi_{j,k} \end{array} \right.$$

*Note that after the filter we get new values from  $M$  and  $N$  (smaller than those in Figure 2.19).*

In our case, we call dlevmar\_bc\_dif() with the following parameters (see appendix E for prototype):

```
dlevmar_bc_dif( f, p, Φ, M, N, lb, ub, MaxIter, opts, info, NULL, NULL, NULL);
```

The given  $p$  contains the first evaluation of all unknowns.

$lb$  is the lower bound for vector  $p$

$ub$  is the upper bound for vector  $p$

MaxIter = maximal number of iterations

opts contains the minimum options for the approximation of the Jacobian of  $\underline{f}$

Note:  $lb$  and  $ub$  can be define in relationship with the first approximation of  $p$

After the function has run,  $p$  contains the calculated solution.

## 2.2 Evaluation of the entire system

### 2.2.1 Laboratory Setup

In order to test the algorithm in the real world, an experiment environment was set up. This environment has to offer optimal conditions in order to observe the system behavior. Indeed, before testing the system in extreme conditions, it has to prove its efficiency in a favorable environment. Our experiment environment is shown on Figure 2.20. Boxes (a) are used as walls/limits and protect the scene from external influences like light sources or moving objects. The green, blue or red objects (b) are the obstacles

(=features) that the robot will have to localize. These features are placed on green points. These green points (c), placed every 20cm on the floor, are used as marks to estimate the coordinates of objects. The algorithm depends heavily on the scene lighting, therefore light sources (d) were set up at the scene's corners. An experiment consists of 10 up to 13 testruns. In a testrun, the robot moves on a predefined path under constant parameters. The parameters are the changes for other experiments to test the algorithm's accuracy, robustness and find hints to optimize the system robustness and the stability of algorithm.

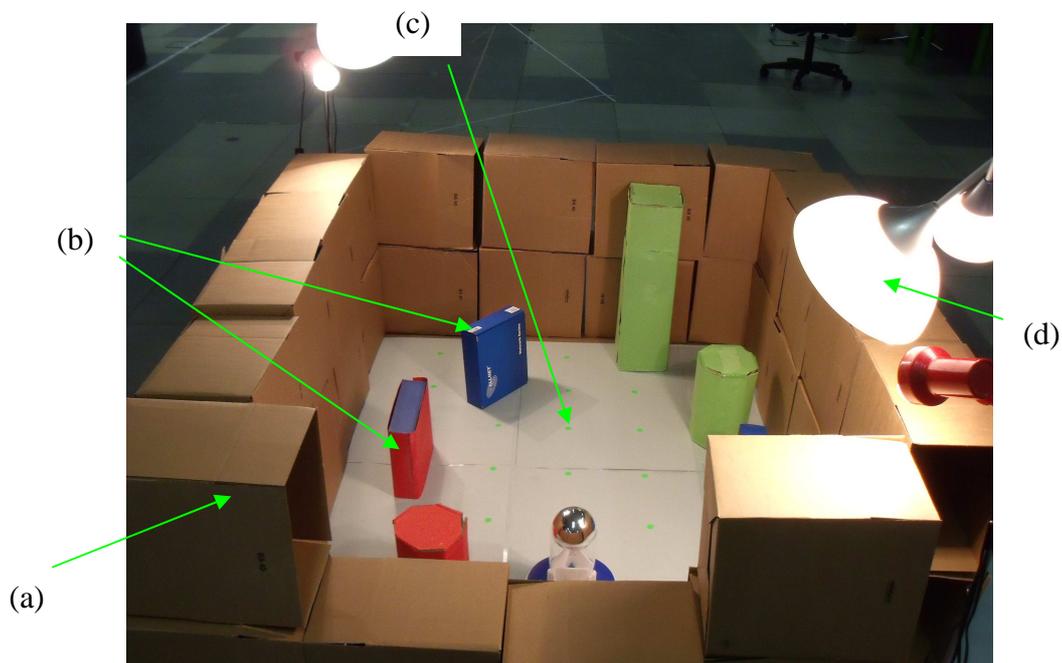


Figure 2.20: The experimental laboratory

## 2.2.2 Investigation 1 (robot)

### Description

The first investigation consists in observing the behavior of the algorithm by different robots. There are two robots: a blue one and a red one. The two robots have to move forward while tracking the features and estimate their positions. The first and the sixth experiments consist of 5 Features, the ninth and the tenth of 6 Features.

<i>Experiment</i>	<i>Parameter(robot)</i>
1	Blue
6	Red
9	Blue
10	Red

Figure 2.21: Configurations of investigation 1

### Results and Interpretation

The following data is an extraction from the whole test evaluation. The detailed results of investigation 1 are documented in an excel sheet that is archived on the attached CD (see folder: Investigations)

<i>Experiment</i>	<i>Parameter(robot)</i>	<i>average error x (abs. value)</i>	<i>average error y (abs. value)</i>	<i>Number of the none tracked</i>
1	Blue	0.05	0.06	0
6	Red	0.10	0.05	1
9	Blue	0.10	0.06	0
10	Red	0.03	0.05	2

Figure 2.22: Results of investigation 1

This data show significant deviation by the x estimation. This is not because of the robot, but the reason can be, that the number of features has been augmented. In the analysis of the result it was also observed that the red robot has difficulty to track green, blue and red features. This lower sensitivity of the red robot could be explained by the camera quality. Although both robots are equipped with the same camera model, two different copies can present slight differences; the camera should therefore be calibrated for each robot. An amelioration of the red robot results can be achieved with a specific calibration of sensitivity.

### **2.2.3 Investigation 2 (localization algorithm)**

#### Description

The localization algorithm has two essential parameters which could influence the quality of the localization results. The first parameter is the number of pictures that are used by the localization algorithm to do its calculations (picture means in this context: features extracted from the pictures and the measured robot position). The second

parameter is the accepted maximum value of pictures that could not be used for calculation in terms of invalid or lost features.

In investigation 2 two different parameter setups are tested:

- 12 pictures and thereof max. 3 invalid pictures
- 16 pictures and thereof max. 4 invalid pictures

In the first setup min. 8 pictures and in the second setup min. 12 pictures are used to calculate the feature's and camera's position.

The investigation includes testruns with two different robots, a red one and a blue one. This should avoid interpreting influences that are effected by differences of the robots. The following table shows the different configurations of the investigation:

<i>Configuration</i>	<i>Robot</i>	<i>Parameter-Setup</i>	<i>Number of testruns</i>
1	Red	12 / 3	10
2	Blue	12 / 3	10
3	Red	16 / 4	10
4	Blue	16 / 4	10

Figure 2.23: Configurations of investigation 2

### Results and Interpretation

The following data is an extraction from the whole test evaluation. The detailed results of investigation 2 are documented in an excel sheet that is archived on the attached CD (see folder: Investigations).

<i>Configuration</i>	<i>Parameter-Setup</i>	<i>average error x (abs. value) [m]</i>	<i>average error y (abs. value) [m]</i>	<i>Number of failed testruns</i>
1	12 / 3	0.05	0.06	2
2	12 / 3	0.10	0.05	5
3	16 / 4	0.13	0.05	0
4	16 / 4	0.09	0.06	1

Figure 2.24: Results of investigation 2

Data analysis show that an increasing of the number of pictures that are used by the localization algorithm to calculate the feature's and the camera's position does not influence the localization results. The variety of the localization results is neither identifiable nor significant.

However, the data shows a significant positive influence on the number of failed testruns. This can be considered as an opportunity to optimize the system in terms of robustness. But note that an increasing of the number of pictures used to calculate the localization has a negative impact on the required computing power.

## 2.2.4 Investigation 3 (featuretracking)

### Description

The featuredetection algorithm has several parameters which could influence the quality of the localization results. The parameters can be used to adjust the featuredetection in terms of its sensitivity. The parameters that are varied within this investigation are the sensitivity for red, green and blue objects.

- LIMIT\_RED
- LIMIT\_GREEN
- LIMIT\_BLUE

In the first setup the featuredetection is adjusted low sensitive; in the second setup it is adjusted high sensitive.

The investigation includes testruns with the red and the blue robot. This should avoid interpreting influences that are effected by differences of the robots. In terms of manufacture tolerances of the cameras, the blue and the red robot need different parameter-values for a similar practical behaviour of the featuredetection.

The following table shows the different configurations of the investigation:

<i>Configuration</i>	<i>Robot</i>	<i>Parameter-Setup</i>	<i>Testruns</i>
1	Blue	LIMIT_RED = 180 LIMIT_GREEN = 115 LIMIT_BLUE = 90 (low sensitive)	10
2	Red	LIMIT_RED = 175 LIMIT_GREEN = 110 LIMIT_BLUE = 87 (low sensitive)	10
3	Blue	LIMIT_RED = 175 LIMIT_GREEN = 110 LIMIT_BLUE = 87 (high sensitive)	10
4	Red	LIMIT_RED = 144 LIMIT_GREEN = 108 LIMIT_BLUE = 88 (high sensitive)	10

Figure 2.25: Configurations of investigation 3

## Results and Interpretation

The following data is an extraction from the whole test evaluation. The detailed results of investigation 3 are documented in an excel sheet that is archived on the attached CD (see folder: Investigations).

<i>Configuration</i>	<i>Parameter-Setup</i>	<i>average error x (abs. value) [m]</i>	<i>average error y (abs. value) [m]</i>	<i>Number of failed testruns</i>
1	low sensitive (blue robot)	0.12	0.09	1
2	low sensitive (red robot)	0.03	0.05	1
3	high sensitive (blue robot)	0.14	0.07	0
4	high sensitive (red robot)	0.11	0.06	0

Figure 2.26: Results of investigation 3

Data analysis shows that an increasing of the featur detection's sensitivity does not influence the localization results. The variety of the localization results is neither identifiable nor significant.

However, the data shows interesting information in the testruns 1&9 of configuration 3 and in testrun 9 of configuration 4. One example is showed as follows:

Testrun	x calc. [m]	y calc. [m]	x first aprox. [m]	y first aprox. [m]	feature	Pos x [m]	Pos y [m]
1	0.179931	-0.339105	0.171009	-0.341896	g1	0.20	-0.30
	0.371625	0.301756	0.414314	0.4151	b1	0.40	0.40
	0.77461	-0.258724	0.988464	-0.405153	b2	1.00	-0.40
	0.555982	-0.380741	0.597538	-0.44741	r1	0.60	-0.40
	-0.493508	1.527445	-0.418705	1.292122		0.40	0.40
	0.179931	-0.339105	0.171009	-0.341896	g1	0.20	-0.30

Figure 2.27: Results (detail) of investigation 3

The table shows a feature, detected and tracked in testrun 1 of configuration 3 that does actually not exist. Features that are used by the localization algorithm, although they does not exist, will disturb the calculation of the position of other features and finally also the calculation of the camera position.

The estimated position of feature "b2" may "suffer" from the additional detected feature that does not exist. All other testruns of this configuration show a sufficient estimation of the position of feature "b2".

The localization results cannot be improved by increasing the featur detection's sensitivity. In contrary, this measure effects the detecting and tracking of non-existing features in certain cases. The assumption that these non-existing features disturb the localization results could be verified for several testruns.

### 2.2.5 Investigation 4 (trajectory)

Our 4<sup>th</sup> investigation aimed to compare the results obtained in a same environment with two different trajectories: a straight line (T1) and a little more complex trajectory (T2):

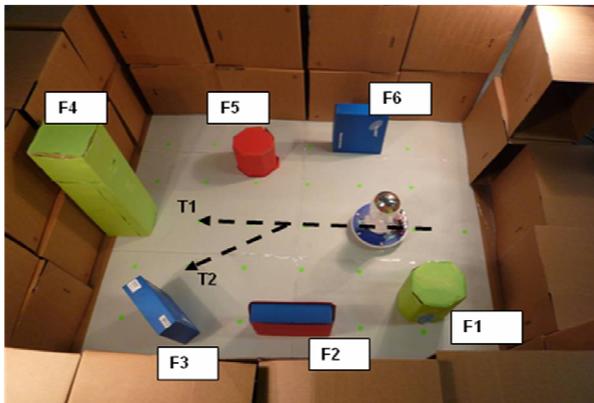


Figure 2.28: Configurations of investigation 4

Our environment consisted of 6 features including one on the axis of the “straight line trajectory” (F4). This was thought to show what we saw with the simulation: the final localization of features should be better with T2, mainly due to the first estimation of F4's position.

We then compared errors of feature localization for a total of 47 testruns (28 for T1 and 19 for T2), with different values of feature tracking sensitivity.

The results do not show any improvement in the feature localization, we even notice a precision degradation for a certain sensitivity of feature tracking. By analyzing all measurements, we saw that too many features were “recognized” (imaginary features were “seen”) although this didn't happen during T1 with the same sensitivity. Of course, those features disturb the global results of least squares approximation: the algorithm tries to optimize a set of parameters from aberrant values.

This experiment shows the critical importance of the set of parameters feature tracking sensitivity/filtering of feature tracking data. In order to correctly evaluate the behavior of our system in such a situation, we should therefore optimize these parameters first, and then retry this experiment.

## 3 Summary

### 3.1 Critique and conclusion of the essential results

The project's objective was about to prove the practical functionality of an algorithm using features from a spatial environment to orientate and to provide the possibility for egomotion. Base of the project is a small mobile robot that is equipped with an omnidirectional driving system, a camera and an emissive sphere, theoretically able to orientate in a spatial environment. In order to control the mobile robot using a localization algorithm, software was developed within this project. This software runs on an external PC and communicates with the robot platform via WLAN.

The detection and tracking of distinctive objects around the robot could be implemented and was evaluated under optimized laboratory conditions.

A rough measurement of the robot's movement could be implemented by capturing the incremental encoders of the omnidirectional wheels. This data is strongly influenced by the environment (type and inclination of the floor) and therefore could only be considered as a support for the calculation of the localization algorithm.

A localization algorithm was implemented and evaluated under optimized laboratory conditions. It uses the angles under which distinctive features are "seen" as well as the rough measurement results of the robot's movement to estimate the feature's and the robot's (the camera's) position within the spatial environment.

The position estimation of the localization algorithm has a limited accuracy, but in the context of the laboratory setup the results can be considered as satisfying. In case of detecting and tracking too few features from the spatial environment, failure measurements may occur.

Investigations on the system within the laboratory setup provide information about opportunities to improve the localization algorithm's accuracy as well as the probability for occurrences of failure measurements. Possible optimizations are described as follows.

### Camera resolution limitations

In terms of saving costs and required computing power the resolution of the camera is limited. That limitation effects an error that cannot be considered as insignificant.

The resolution of possible angles between  $0^\circ$  and  $360^\circ$  is approximately  $1.4^\circ$ . Hence an error of  $1.4^\circ$  has to be expected in the worst case.

The impact on the localization results depends on the distance between the robot and the detected object. A calculation with the law of cosines results that an object that is positioned 1 m afar from the robot could be considered as approximately 2.5 cm away from the actual “real” position due to the camera’s resolution limits.

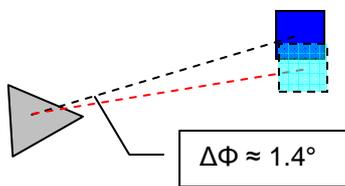


Figure 3.1: Camera resolution error

### Featuretracking

Imprecise localization results can be effected by imprecise featuredetection and featuretracking results.

Errors could for example result due to a miscalculation of the detected object’s center: Not every object is well illuminated from every possible direction the robot may “see” it. This can effect that only half of the object is detected and hence the object’s center may be “shifted” to one direction (see Figure 3.2: Problems caused by partial illumination). The impact on the localization results depends on the distance between the robot and the detected object. Near objects that have temporary the described properties will effect a bigger error than objects that are far away.

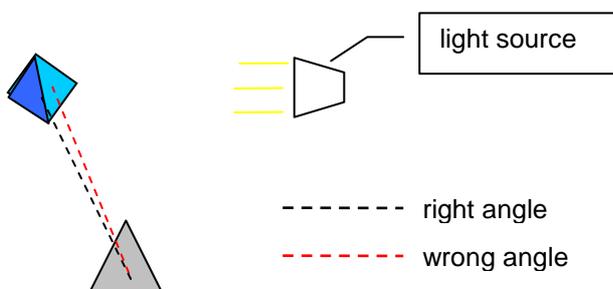


Figure 3.2: Problems caused by partial illumination

Errors can also occur because of feature losses. The classification of an assumed object as valid depends on how big it appears in the picture. Hence an object that is too far away could not be detected and will be lost or never “seen”. In addition it is possible that the featuretracking gets confuse by the occurrence of too many features, because new features could be considered as already seen. The localization results become imprecise if the algorithm gets too less or wrong information about the spatial environment around the robot.

In order to optimize the featuredetection and the featuretracking in terms of robustness several measures are possible. Using filters to brighten the picture or increase its contrast could effect a better initial situation for the implemented featuredetection. These filters can also be implemented adaptive to changes of the environment.

An improvement is also possible through changing of the segmentation algorithm (see Chapter 2.1.6 Camera Data Analysis). Instead of using the whole picture to calculate the threshold-status of every pixel it is also possible to calculate the threshold-status in the context of a 3x3-Array that is shifted through the whole picture (Figure 3.3: ). This could provide an improvement of the algorithm in terms of problems effected by partial illumination.



Figure 3.3: Partial segmentation

Instead of using the colour intensity, other properties of the spatial environment could be used as well. Distinctive objects can also be detected by their edges or the type of their shape. This could also improve the interpretation algorithm in terms of problems effected by partial illumination or the independence from illumination itself.

The optimization opportunities of the featuredetection and the featuretracking are manifold. Further investigation has to be accomplished to predict which optimization opportunities might have the most positive impact on the system’s robustness.

#### Least squares approximation

The localization algorithm, as it is now implemented, is not devoid of intrinsic imperfections. Before implementing it on a master controller, we would have to reduce

its needs in calculation power. Indeed, during our experiments, we measured the time needed to process data with two different platforms:

- PC from CCRL: CPU = AMD Phenom 9850 (Quad-Core, 2.5GHz), 4GB RAM
- Netbook: CPU = Intel Atom N270 (Single-Core, 1.6GHz), 1GB RAM

With the first configuration, the average processing time was around 400ms and with the second one, around 1.8s. By extrapolating this data, we can easily realize that it cannot be adapted to an embedded microcontroller.

A combination of solutions might make it possible, among those we already listed:

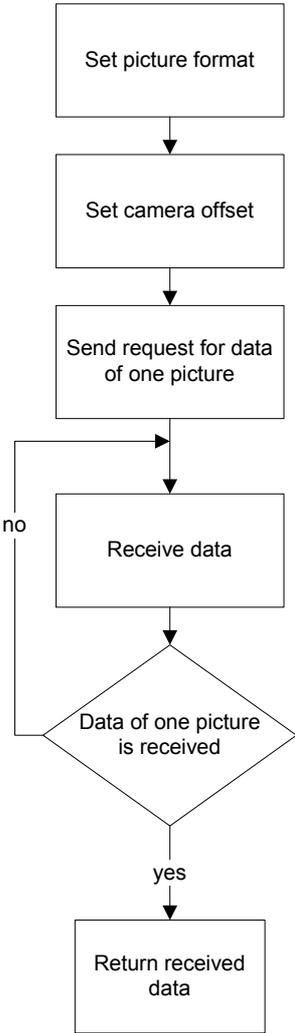
- linearization of atan2 in our nonlinear equation system (if estimations are good, the algorithm always works around a same point)
- change parameters of dlevmar\_bc\_dif() (reducing iterations limit, enhancing tolerance on result...)

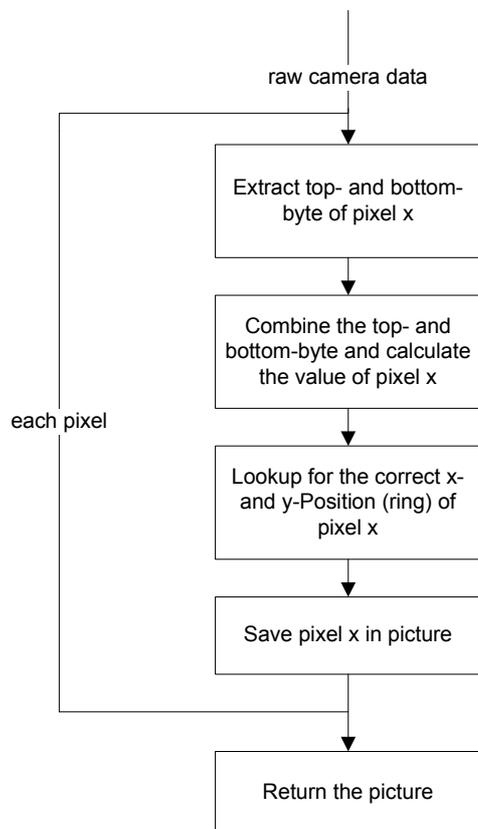
#### Filtering of feature tracking data

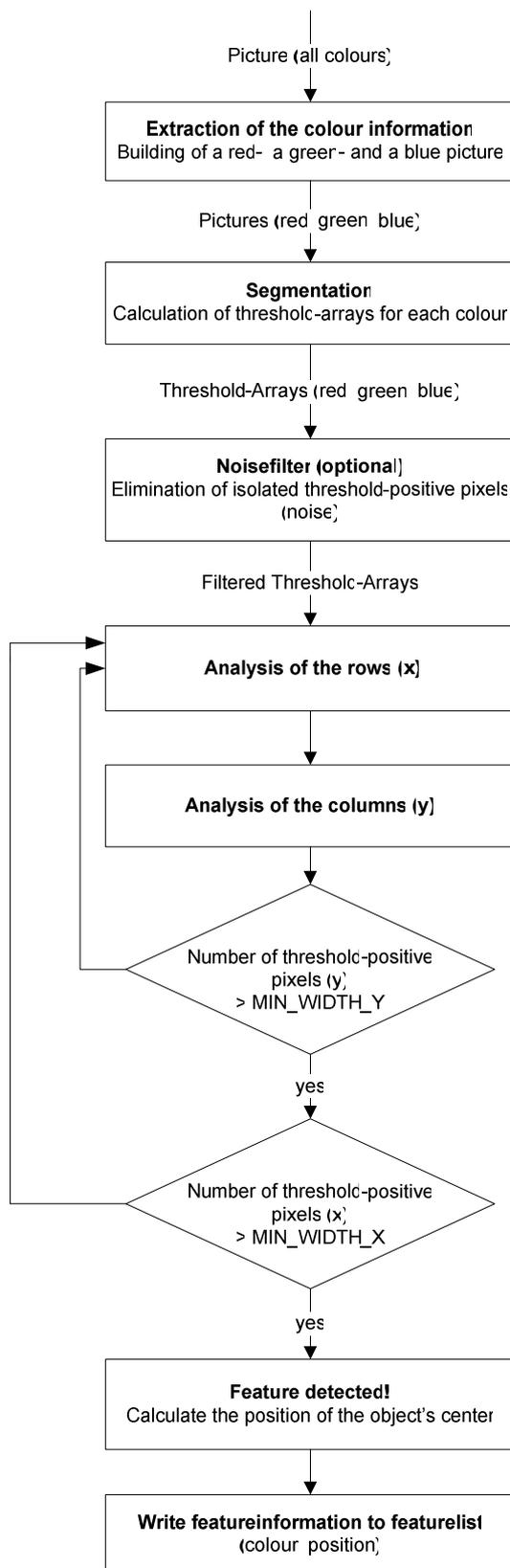
In real case, the biggest problem we had was to recognize features; we therefore tried to optimize parameters related to the filtering of feature tracking data. We pointed out the crucial importance to harmonize the filtering parameters to the feature tracking sensitivity: when the sensitivity is too high and the filtering not strict enough, imaginary features can be “seen”, which affects all results (localization of features/camera); and when the sensitivity is too low and the filtering too strict, real feature can be “unseen” which can lead to collisions with obstacles.

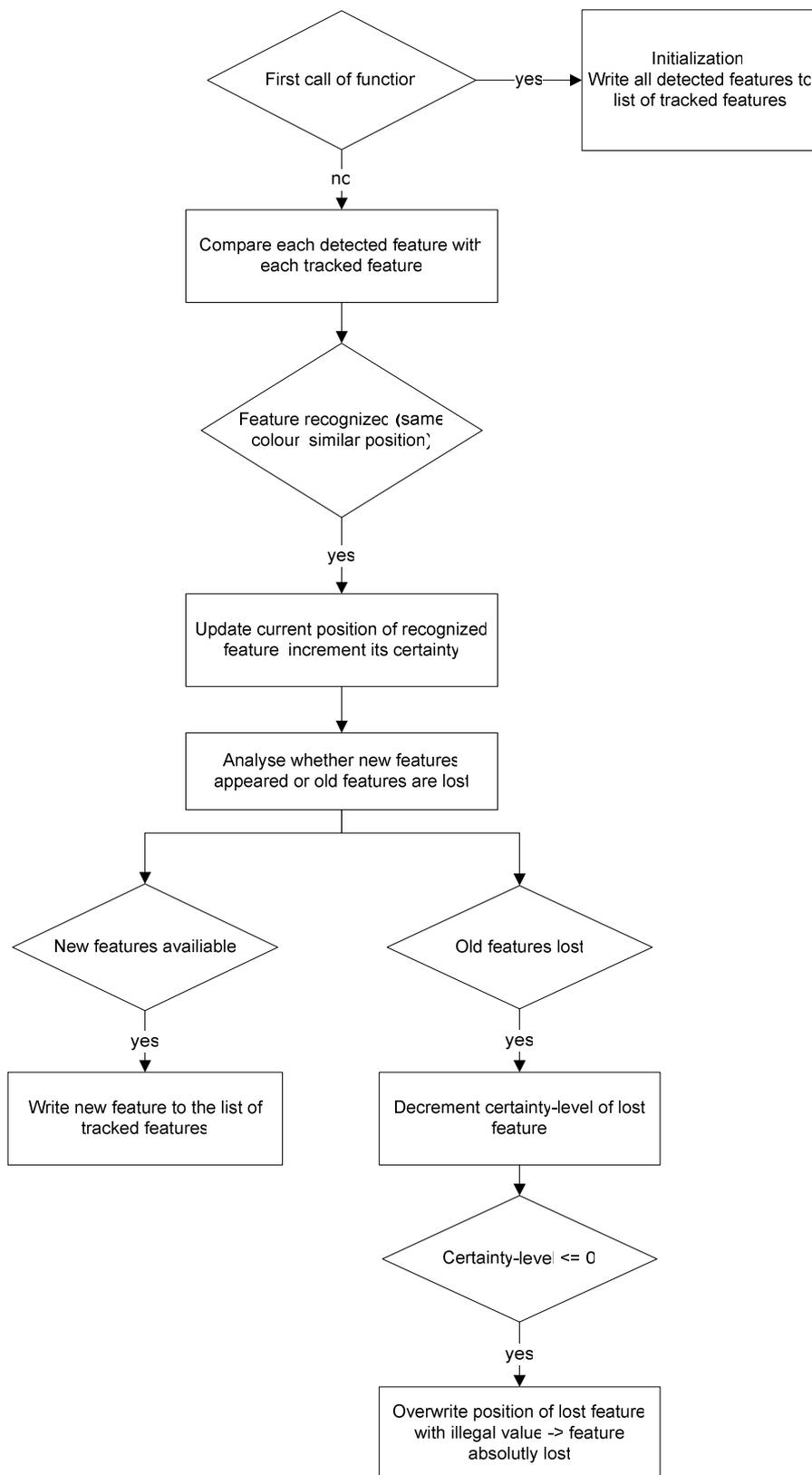
# Appendix

**Appendix A: Flow Diagram of function getCamData()**



**Appendix B: Flow Diagram of function DataToPic()**

**Appendix C: Flow Diagram of function featuredetection()**

**Appendix D: Flow Diagram of function featuretracking()**

**Appendix E: Prototype of `dlevmar_bc_dif()` (see [3] for more details):**

```
int dlevmar_bc_dif(  
    void (*func)(double *p, double *hx, int m, int n, void *adata),  
    double *p, double *x, int m, int n, double *lb, double *ub,  
    int itmax, double *opts, double *info, double *work, double *covar, void *adata);
```

## List of Figures

Figure 1.1: mobile robot (a); spherical mirror (b) .....	8
Figure 2.1: Functional diagram of the entire system .....	10
Figure 2.2: Client – Server Communication .....	11
Figure 2.3: Robot position measurement .....	12
Figure 2.4: Extraction of the ring.....	13
Figure 2.5: raw camera data.....	14
Figure 2.6: calculation of the pixels.....	14
Figure 2.7: Rebuilding and spreading of the ring .....	14
Figure 2.8: Temporary loss of a feature.....	16
Figure 2.9: Collecting data.....	17
Figure 2.10: Collected data.....	18
Figure 2.11: Schematic view of a short move in a 4-obstacles environment .....	18
Figure 2.12: Data measured by the feature tracking.....	19
Figure 2.13: Schematic representation of all parameters for a given feature at a certain position.....	19
Figure 2.14: Simulation with a “complex” trajectory (a); Simulation with a straight line trajectory (b) .....	21
Figure 2.15: First evaluation of the position.....	22
Figure 2.16: Bad configuration for the first evaluation of the position of a feature .....	22
Figure 2.17: Better configuration for the first evaluation.....	23
Figure 2.18: Global structure of the localization algorithm.....	24
Figure 2.19: Filtering applied to an array returned by feature tracking.....	25
Figure 2.20: The experimental laboratory .....	27
Figure 2.21: Configurations of investigation 1 .....	28
Figure 2.22: Results of investigation 1 .....	28
Figure 2.23: Configurations of investigation 2 .....	29
Figure 2.24: Results of investigation 2 .....	29
Figure 2.25: Configurations of investigation 3 .....	30
Figure 2.26: Results of investigation 3 .....	31
Figure 2.24: Results (detail) of investigation 3.....	31
Figure 2.23: Configurations of investigation 4.....	32
Figure 3.1: Camera resolution error.....	34
Figure 3.2: Problems caused by partial illumination .....	34
Figure 3.3: Partial segmentation .....	35



## Bibliography

- [1] [botzeit.de (2009). Robotik-Blog. Blog zu „SICK LMS-100“. URL: [http://botzeit.de/blog/2009\\_03\\_04\\_sick-lms-100.html](http://botzeit.de/blog/2009_03_04_sick-lms-100.html) (Abruf: 29.01.2009)
- [2] J. Gluckman, and S.K. Nayar. Ego-motion and omnidirectional cameras. ICCV, pages 999-1005, 1998.
- [3] M.I.A. Lourakis, “levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++”, <http://www.ics.forth.gr/~lourakis/levmar/>, 2004