

# **VISION-BASED AUTONOMOUS NAVIGATOR**

by

Ceyda Nur ÖZTÜRK

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science


in

Computer Engineering


July 2012  
Istanbul, Turkey

## APPROVAL PAGE

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

  
Assist. Prof. Dr. Tuğrul YANIK  
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

  
Prof. Dr. Erkan İMAL  
Supervisor

  
Assoc. Prof. Dr. Veli HAKKOYMAZ  
Co-Supervisor

Examining Committee Members

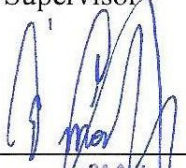




Prof. Dr. Erkan İMAL

Prof. Dr. Onur TOKER

Assoc. Prof. Dr. Veli HAKKOYMAZ

Assist. Prof. Dr. Nahit EMANET

Assist. Prof. Dr. Tuğrul YANIK

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assoc.Prof.Dr. Nurullah ARSLAN  
Director

Date  
July 2012

# VISION-BASED AUTONOMOUS NAVIGATOR

Ceyda Nur ÖZTÜRK

M. S. Thesis - Computer Engineering  
July 2012

Supervisor: Prof. Dr. Erkan İMAL

Co-Supervisor: Assoc. Prof. Dr. Veli HAKKOYMAZ

## ABSTRACT

During past 20 years, much of the research work has been in the area of computer vision for mobile robot navigation. Despite of being computationally more expensive and less accurate compared to sonar or laser range sensors due to their indirect storage of the world's geometry, visionary sensors are promising since they are compact, noninvasive, ubiquitous, cheap, full of information and well-understood because of their similarity to humanoid vision systems.

In this thesis, literature on vision-based navigation is reviewed by emphasizing the robust feature extraction and visionary simultaneous localization and mapping (SLAM) studies. Based on the findings obtained, a similar SLAM system using single camera is proposed. For the implementation of the system, while a mobile robot navigates according to a provided trajectory, some robust features belonging to natural landmarks in the environment are extracted from the camera frames of the robot and tracked through the frames to perceive the environment. In order to extract those robust features, some popular algorithms are adapted. By matching the features between camera frames, relative coordinates of the landmarks in 3D world can be obtained approximately by depending on a prior calibration process. Integrating the robot position estimates based on odometry of robot with landmark coordinate estimates in extended Kalman filter (EKF), relative positions of the robot and the environment landmarks can be computed with some amount of uncertainty. This enables the building of a representative map of the environment simultaneously. Thus, a monocular SLAM system has been investigated, which may also be the basis for a learning mobile robot with stochastic actions that is capable of operating in unknown environments.

**Keywords:** Mobile robots, Single camera, Robust features, Feature projections, SLAM.

# GÖRÜNTÜ TABANLI OTONOM NAVİGATÖR

Ceyda Nur ÖZTÜRK

Yüksek Lisans Tezi – Bilgisayar Mühendisliği  
Temmuz 2012

Tez Danışmanı: Prof. Dr. Erkan İMAL

Tez Eşdanışmanı: Doç. Dr. Veli HAKKOYMAZ

## ÖZ

Son 20 yılda gezgin robot navigasyonu amaçlı bilgisayarlı görme sahasında birçok araştırma yapılmıştır. Gerektirdiği hesaplama yüküne ve sonar ya da lazerli mesafe algılayıcılarla karıştırıldığında ortamın geometrisini dolaylı bir şekilde içeriyor olmasının sonucunda daha fazla hataya açık olmasına rağmen, kameralar az yer kapladığı, ortamlarla etkileşmediği, sık ve ucuz bulunduğu, bilgi dolu olduğu ve insana ait görme sistemine benzediği için rahat anlaşılabilirliğinden gelecek vaatmektedir.

Bu tezde, sağlam öznitelik çıkarma ve görmeye bağlı eşzamanlı konumlandırma ve haritalandırma (SLAM) çalışmalarına vurgu yapılarak görme tabanlı navigasyon üzerine yazın taranmaktadır. Elde edilen bulgulara binaen benzer bir tek kameralı SLAM sistemi önerilmektedir. Sistemin uygulaması için, gezgin bir robot sağlanan bir yörüngeye göre hareket ederken, robotun kamera çerçevelerinden ortamı algılayabilmek için ortamdaki işaretlere ait bazı sağlam öznitelikler çıkarılmakta ve çerçeveler boyunca takip edilmektedir. Bu sağlam öznitelikleri çıkarabilmek için bazı popüler algoritmalar uyarlanmaktadır. Öznitelikleri kamera çerçeveleri arasında eşleyerek, 3B dünyadaki işaretlerin izafi koordinatları evvelki bir kalibrasyon işlemi sayesinde yaklaşık olarak elde edilebilir. Genişletilmiş Kalman filtresinde (EKF) robotun odometresine bağlı robot konumu tahminleri, işaretlerin koordinat tahminleriyle bir araya getirilerek robotun ve ortamdaki işaretlerin konumları bir miktar belirsizlikle hesaplanabilir. Bu eşzamanlı olarak temsili bir ortam haritası inşasına olanak sağlamaktadır. Böylece, bilinmeyen ortamlarda çalışabilecek, öğrenen ve rastgele davranışlı bir gezgin robot için de temel olabilecek tekgözlü SLAM sistemi araştırılmaktadır.

**Anahtar Kelimeler:** Gezgin robotlar, Tek kamera, Sağlam öznitelikler, Öznitelik izdüşümleri, SLAM.

## **DEDICATION**

*To meritorious people who directed me towards  
education and science at the crossroads.*

## **ACKNOWLEDGEMENT**

I would like to thank Prof. Dr. Erkan İMAL who has admirably striven to provide the facilities that could make this research feasible and tried to keep the motivation high throughout the study. Besides, I express my appreciations for the contributions and suggestions of Assoc. Prof. Dr. Veli HAKKOYMAZ. I should also emphasize that initial briefing of Cihan ULAŞ on the issue had a considerable effect to settle the direction of the research.

In particular, I am greatly indebted to my family and to my friends whose understanding and assistance heartened and guided me at the hard times.

Finally, I gratefully acknowledge that this thesis has been supported by the Scientific Research Fund of Fatih University under the project number P50061202\_B.

## TABLE OF CONTENTS

ABSTRACT.....	iii
ÖZ.....	iv
DEDICATION.....	v
ACKNOWLEDGEMENT.....	vi
TABLE OF CONTENTS.....	vii
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
LIST OF SYMBOLS AND ABBREVIATIONS.....	xiii
CHAPTER 1 INTRODUCTION.....	1
1.1 THE NAVIGATION PROBLEM.....	2
1.1.1 Self-Localization.....	2
1.1.2 Map-Building and Map-Interpretation.....	4
1.1.3 Path Planning.....	5
1.2 VISION-BASED NAVIGATION.....	6
1.3 CURRENT SYSTEMS.....	8
1.4 BACKGROUND AND MOTIVATIONS.....	10
CHAPTER 2 LITERATURE REVIEW.....	11
2.1 TYPES OF VISIONARY NAVIGATION.....	14
2.1.1 Map-based Navigation.....	14
2.1.2 Map-building Navigation.....	18
2.1.3 Mapless Navigation.....	21
2.2 ROBUST FEATURE EXTRACTION.....	24
2.2.1 Scale Invariant Feature Transform (SIFT).....	26
2.2.2 Speeded-Up Robust Features (SURF).....	28
2.3 VISIONARY SLAM.....	29
2.3.1 The SLAM Process.....	29
2.3.2 Visionary SLAM.....	30
CHAPTER 3 DESIGN AND IMPLEMENTATION.....	38
3.1 PROPOSED SYSTEM.....	38

3.1.1	Agent Description .....	38
3.1.2	System Design .....	39
3.1.2.1	Robust Feature Extractor .....	40
3.1.2.2	Feature Matcher .....	40
3.1.2.3	Landmark Parameter Estimator .....	41
3.1.2.4	Extended Kalman Filter (EKF).....	41
3.1.2.5	Map Builder .....	41
3.1.2.6	Path Planner .....	41
3.1.2.7	Controller.....	42
3.1.2.8	Navigator .....	42
3.1.2.9	Graphical User Interface (GUI).....	42
3.2	ROBUST FEATURE EXTRACTION AND MATCHING .....	43
3.2.1	Scale Space Extrema Detection .....	43
3.2.2	Keypoint Localization .....	46
3.2.3	Orientation Assignment.....	46
3.2.4	Keypoint Descriptor .....	47
3.2.5	Matching Keypoints .....	49
3.3	LANDMARK PARAMETER ESTIMATION .....	50
3.3.1	Camera Calibration .....	51
3.3.2	Projection to 3D Coordinates.....	55
3.4	EKF AND MAP BUILDING .....	57
3.4.1	EKF Matrices .....	57
3.4.2	EKF Steps.....	62
3.4.2.1	State Update of Odometry .....	62
3.4.2.2	State Update of Re-observed Landmarks .....	62
3.4.2.3	Adding New Landmarks .....	63
3.5	MATLAB INTERFACE .....	64
3.6	SYSTEM CONTROL AND NAVIGATION .....	65
CHAPTER 4	EXPERIMENTS AND RESULTS .....	67
4.1	EXPERIMENTAL PLATFORMS .....	67
4.1.1	E-Puck .....	68
4.1.1.1	Camera Module.....	68
4.1.2	Robotino .....	69
4.1.2.1	Data Acquisition .....	70
4.1.2.2	Odometry Accuracy .....	71
4.2	FEATURE EXTRACTION AND MATCHING .....	73
4.2.1	Evaluation of the SIFT Algorithm .....	73



4.2.2	Evaluation of the SURF Algorithm .....	76
4.3	CALIBRATION AND PROJECTION TO 3D .....	77
4.4	LOCALIZATION OF THE ROBOT AND THE LANDMARKS.....	79
CHAPTER 5	CONCLUSIONS AND FUTURE WORK.....	84
5.1	CONCLUSIONS .....	84
5.2	FUTURE WORK.....	85
REFERENCES.....		86
APPENDIX A	ROBOTINO.....	89
APPENDIX B	E-PUCK.....	91
APPENDIX C	MATLAB CODE OF CONTROLLER.....	93

## LIST OF TABLES

### TABLE

1.1 Comparison of self-localization methods. ....	3
3.1 Computed scales in an octave. ....	44
4.1 Controls generating a sample rectangular path for Robotino.....	69
4.2 Frequencies of frame retrieval.....	70
4.3 Frequencies of odometry retrieval.....	70
4.4 Odometry accuracy experiments. ....	72
4.5 Odometry uncertainties. ....	73
4.6 Performance of the SIFT algorithm on frames at 5 Hz.....	74
4.7 Performance of the SIFT descriptor matching stage on frames at 5 Hz.....	75
4.8 Performance of the SIFT algorithm on frames at 0.5 Hz.....	75
4.9 Performance of the SIFT descriptor matching stage on frames at 0.5 Hz. ....	75
4.10 Performance of the SURF algorithm on frames at 0.5 Hz.....	76
4.11 Performance of the SURF descriptor matching stage on frames at 0.5 Hz. ....	76
4.12 Average reprojection errors of the calibration routines. ....	77
4.13 Camera matrix parameters computed. ....	78
4.14 Distortion parameters computed.....	78
4.15 Diagonal values of covariance matrix P and controls computed based on the odometry for the rectangular path. ....	80
4.16 Diagonal values of covariance matrix P and controls computed based on the odometry for the circular path. ....	80
4.17 The true and the estimated positions of the robot for the rectangular path.....	81
4.18 The true and the estimated positions of the robot for the circular path. ....	81
4.19 Evaluations on final robot localization for rectangular and circular paths. ....	83
A.1 Robotino hardware specification. ....	90
B.1 E-puck hardware specification. ....	92

## LIST OF FIGURES

### FIGURE

1.1	Mobile robots deployed in transportation and exploration domains. ....	9
2.1	Processing steps of FINALE system (Kosaka and Kak, 1992). ....	15
2.2	(a) The camera image and the superimposed expectation map (b) Output of the model-guided edge detector (c) Uncertainty regions associated with the ends of the model line features (d) Matching after Kalman filtering (Kosaka and Kak, 1992). ....	16
2.3	(a) Physical structure of a hallway segment showing the doors and alcoves (b) The topological representation of the hallway in (a) (Meng and Kak, 1993). ....	17
2.4	(a) An example of the hallway as seen by the camera (b) The output of the edge detector (c) The relevant floor edges (d) The Hough map (Meng and Kak, 1993). ....	18
2.5	(a) A top view of the reconstructed trajectory and corridor (b) An image with the features and the flow (Bouget and Perona, 1995). ....	20
2.6	Incoming calibrated camera image of the white competition fence (left) and resulting pseudo-LIDAR scan (right) (Schepelmann et al., 2009). ....	24
2.7	Good and poor choices of features (Bradski and Kaehler, 2008a). ....	25
2.8	(a) The training images for two objects (b) Training objects in a cluttered image with extensive occlusion (c) The results of recognition (Lowe, 2004). ....	27
2.9	The recall versus (1-precision) graph for different methods using similarity difference matching (Bay et al., 2006). ....	28
2.10	Overview of the SLAM process (Riisgaard and Blas, 2005). ....	30
2.11	Bird's-eye view of SIFT feature database (Se et al., 2002). ....	32
2.12	Image search in successive frames during feature initialisation (Davison, 2003). ....	33
2.13	(a) Early exploration and first turn (b) Mapping back all and greater uncertainty (c) Just before loop close, maximum uncertainty (d) End of circle with closed loop and drift corrected (Davison et al., 2007). ....	36

2.14	Processing of vision data collected in indoor environments (Yang et al., 2012)...	37
3.1	Block diagram of system call sequence.....	40
3.2	Scale space of an image.....	44
3.3	Computation of DoG images (Lowe, 2004).....	45
3.4	Maxima and minima of the DoG images (Lowe, 2004).....	45
3.5	Histogram of orientations to determine peaks (Sinha, 2010).....	47
3.6	A 2x2 descriptor array computed from an 8x8 set of samples (Lowe, 2004).....	48
3.7	The keypoints extracted and matched in 2 consecutive frames with (a) the SIFT and (b) the SURF algorithms.....	50
3.8	Point projection onto the image plane (Bradski and Kaehler, 2008a).....	52
3.9	Different views of calibration object.....	53
3.10	Corner locations on a chessboard view.....	54
3.11	Triangulation process for row-aligned stereo images (Bradski and Kaehler, 2008a).....	57
3.12	Matrices (a) X (b) P (c) K, and (d) JH used in EKF (Riisgaard and Blas, 2005)...	59
3.13	Odometry controls generated in the reference frame of robot.....	60
3.14	Rows and columns appended to matrix P for a new landmark (Riisgaard and Blas, 2005).....	64
3.15	Matlab interface of the system so-called VBAN.....	65
4.1	Educational robots (a) E-puck (b) Robotino.....	67
4.2	320x240 images retrieved from E-puck camera module.....	68
4.3	Degree of freedom of Robotino.....	70
4.4	Extracted and matched SIFT features (a) before and (b) after threshold update. ..	74
4.5	Trajectories drawn for the robot navigation along a rectangular path.....	82
4.6	Trajectories drawn for the robot navigation along a circular path.....	82
B.1	Outline of electronic circuitry of the E-puck.....	91

## LIST OF SYMBOLS AND ABBREVIATIONS

### SYMBOL/ABBREVIATION

ADC	Analog to Digital Converter
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
AUV	Autonomous Underwater Vehicles
CAD	Computer Aided Design
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CV	Computer Vision
DCI	Data Converter Interface
DoF	Degree of Freedom
DoG	Difference of Gaussians
DSC	Digital Signal Controller
EKF	Extended Kalman Filter
FP	False Positive
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
GPU	General Processing Unit
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
LED	Light Emitting Diodes
LoG	Laplacian of Gaussian
MCL	Monte Carlo Localization
MDP	Markov Decision Process

PCB	Printed Circuit Board
POMDP	Partially Observable Markov Decision Process
RAM	Random Access Memory
RL	Reinforcement Learning
SFM	Structure From Motion
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded Up Robust Feature
SVN	Subversion
TP	True Positive
UART	Universal Asynchronous Receive Transmit
UAV	Unmanned Aerial Vehicle
ULV	Unmanned Land Vehicles
VBAN	Vision-based Autonomous Navigator
VFF	Virtual Force Field
W-LAN	Wireless Local Area Network

# CHAPTER 1

## INTRODUCTION

Any being in nature is mobile and intelligent to some extent, and most of the time the range of things they can achieve are based on their degree of mobility and level of intelligence. Many technologies have been developed by referencing such abilities of human being or some other animals in nature since the industrial revolution in 18<sup>th</sup> century. What motivated the idea of first mechanically powered submarine Plongeur (Diver) by French in 1863 were the lives of underwater creatures. Similarly, birds were the source of inspiration for planes, which lead to Wright Brothers' invention of the world's first successful airplane and realization of the first controlled, powered and sustained human flight in 1903. And in 1959 why robots are employed by Ford factories was to substitute and automate the human work. As such studies have been built on top of each other, it revealed out the fact that the more machines are desired to achieve, the more mobile and intelligent they have to be.

Today, wheels or mechanical legs are mounted to enable robots with mobility. These mobile robots are also equipped with various sensors enabling interaction with the environment. Thus they may have some pseudo-biotic capabilities to see, to touch or to listen. Besides, intelligence providing data integration and behavioral controls is achieved by electronic brains that might employ various artificial intelligence (AI) techniques such as fuzzy logic, neural networks, Bayes inference or reinforcement learning (RL).

Having those functionalities in place, robots can be driven to deliver food in hospitals, to move containers at loading docks, to guide people in buildings, or to mow

grass in gardens. In large scale, they can be employed for autonomous highway navigation, military operations, deep sea or planetary explorations.

## **1.1 THE NAVIGATION PROBLEM**

Mobile robot applications essentially involve navigation to accomplish any task. Navigation is considered as a high level task to achieve for them when it is not human guided or not simply a sequence of actions programmed but involves some autonomy, in other words intelligence.

Autonomous robot navigation mostly means robot's ability to determine its own position in its frame of reference and then to plan a path towards some goal location. In order to achieve these tasks, robots need to have a representation of the environment such as maps and an ability to interpret that representation.

Fundamentally, navigation can be defined as a combination of three competences; self-localization, map-building and map-interpretation, and path planning. Their explanations and various solutions to them including vision are given through the following subsections.

### **1.1.1 Self-Localization**

Self-localization is the ability of a robot to answer the question of "Where am I relative to the world?" in order to increase its accuracy on performing a task. Depending on specific applications, there may be various solutions to this question each having different pros and cons which are summarized in Table 1.1.

GPS has long been used to localize aerial vehicles. Yet GPS signals are not available to rural outdoor environments. Due to line of sight issues, in addition to indoor environments, they are even absent in the presence of heavy vegetation or overhanging canopy in outdoor regions (Yang et al., 2010). Moreover, for outdoor land vehicles it is demonstrated that GPS systems' signal instability may lead to a magnitude of more than a few meters drift in localization which is unsafe for navigation (Velat et al., 2007).



Another means to localize robots is via range measurement to environment obstacles. Range sensors, sonar sensors or cameras can be used to measure depths which can be processed as features representing the environment (Riisgaard and Blas, 2005; Hager, 2008). Later, by performing an existing map or built map correlations with the obtained features, the uncertainties in robot location can be resolved.

Laser scanners are very precise, efficient, and the output does not require much computation to process, but they are very expensive, and when they are looking at certain surfaces including glass, they can give very bad readings. Additionally, they can not be used underwater since the water disrupts the light, and the range is drastically reduced. Sonar sensors are very cheap compared to laser scanners. In contrast, their measurements are not very good, and they often give bad readings. Where laser scanners have a single straight line of measurement emitted from the scanner with a width of as little as 0.25 degrees a sonar can easily have beams up to 30 degrees in width. Though, underwater they are the best choice and resemble the way dolphins navigate.

**Table 1.1** Comparison of self-localization methods.

<b>METHOD</b>		<b>ADVANTAGES</b>	<b>DISADVANTAGES</b>
<b>GPS Signals</b>		<ul style="list-style-type: none"> <li>• can be used to localize aerial vehicles</li> </ul>	<ul style="list-style-type: none"> <li>• not available to indoor or rural outdoor environments</li> <li>• more than a few meters drift in localization</li> </ul>
<b>RANGE MEASUREMENT METHODS</b>	<b>Sonar Sensors</b>	<ul style="list-style-type: none"> <li>• cheap</li> <li>• best for underwater</li> </ul>	<ul style="list-style-type: none"> <li>• often give bad readings</li> </ul>
	<b>Laser Scanners</b>	<ul style="list-style-type: none"> <li>• precise</li> <li>• efficient</li> </ul>	<ul style="list-style-type: none"> <li>• expensive</li> <li>• bad readings for certain surfaces</li> <li>• can not be used underwater</li> </ul>
	<b>Cameras</b>	<ul style="list-style-type: none"> <li>• contain more information</li> <li>• compact</li> <li>• noninvasive</li> <li>• similar to humanoid vision systems</li> <li>• easy to access and cheap</li> </ul>	<ul style="list-style-type: none"> <li>• computationally intensive</li> <li>• error prone due to changes in light</li> <li>• less accurate due to their indirect storage of the world's geometry</li> </ul>
<b>Artificial Landmarks</b>		<ul style="list-style-type: none"> <li>• easy to implement</li> </ul>	<ul style="list-style-type: none"> <li>• not portable easily</li> <li>• may lack autonomy</li> </ul>

Traditionally, it has been very computationally intensive and also error prone due to changes in light to use vision for the range measurement task. Given a room without light, a vision system will most certainly not work. However, in the recent years, there have been some interesting advances within this field. Often, the systems use a stereo or triclops system to measure the distance. Also, there is a lot more information in a picture compared to laser and sonar scans. This used to be the bottleneck, since all this data needed to be processed, but this is becoming less of a problem with advances in algorithms and computation power (Riisgaard and Blas, 2005).

Finally, artificial landmarks that can be sensed by visionary, optic or inductive sensors can aid localization. The idea can be as simple as a robot's guess about its location given an artificial landmark database as the representation of an environment. Though, such applications are usually too environment specific that they are not easily portable to similar problems and lack autonomy.

Problems regarding self localization include classification, recognition and structure from motion (SFM). During motion of robot, features (or landmarks) are extracted from environment depending on sensor type, and they are classified and recognized continuously to construct a representation of the environment which is called SFM or to perform a match with a previously seen case or pre-built maps. Some popular localization algorithms are Monte Carlo localization (MCL) based on particle filters, Kalman filters, and Extended Kalman filters (EKF) which can handle nonlinear dynamics of the robot.

### **1.1.2 Map-Building and Map-Interpretation**

For mobile robots, maps denote any one-to-one mapping of the world onto an internal representation. They can be in the shape of a metric map or any notation describing locations in the robot frame of reference and may contain different degrees of detail generated by CAD models, occupancy maps, virtual force fields (VFF), and simple graph of interconnections between the elements in the environment. Occupancy maps are formed by the 2D projection of volume of each object in the environment onto the horizontal plane. VFF is an occupancy map where each occupied cell exerts a repulsive force to the robot where goal exerts an attractive force, and all forces are then combined using vector addition and subtraction to indicate the new heading of the

robot. However, though occupancy grids are rich in geometrical detail, they are highly dependent on the accuracy of robot odometry and sensor uncertainties. Additionally, for large scale and complex spaces, it may not be computationally efficient for path planning or localization (DeSouza and Kak, 2002).

Robots may be provided with environment map beforehand or be expected to build it online. In the literature, the robot mapping problem is often referred to as simultaneous localization and mapping (SLAM). In SLAM during navigation, robot must map the environment in which it is being operated using its sensors so that it learns what is around and thus avoids the obstacles. While constructing a map, robot must also know where it is. This problem gets much complicated if the environment is allowed to change, in other words dynamic, when the robot moves around. EKF combined with a landmark sensing model is the most widely used method for SLAM. Not only the robot pose and the distinguishable landmark locations but also their uncertainties are maintained (Russell and Norvig, 2003).

### **1.1.3 Path Planning**

If the navigation is to accomplish a goal rather than just roaming, then the challenge of path planning should be dealt with. The path planning problem is to find a path from one coordinate to another in space. So it is effectively an extension of localization, in that it requires the determination of the robot's current position and a position of a goal location, both within the same frame of reference or coordinates. The primary characteristic of path planning is that it involves continuous spaces. To successfully prepare the plan, it is important to be informed about free space including all attainable coordinates and the occupied space which is the space of unattainable coordinates in the environment, and this information is obtained via maps.

Considering that the robot motion is deterministic, and the localization of the robot is exact, different approaches such as cell decomposition and skeletonization can be applied in high dimensional continuous spaces. Cell decomposition methods decompose the free space into finite number of contiguous regions, called cells. Then, the path planning problem becomes a discrete graph search problem that can be solved by algorithms like A\* or value-iteration. As in VFF, a function called potential field can be defined over the state space whose value grows with distance to closest obstacle. By

this way, clearance from the obstacles is maximized while the path length is minimized. Then, the resulting path may be longer but also it is safer. Skeletonization means reducing the robot's free space to a one dimensional representation which is called the skeleton. Some examples to this approach include the Voronoi graphs and probabilistic roadmaps. Voronoi graph of a free space is the set of all points that are equidistant to two or more obstacles. Path planning is achieved by finding the closest points to initial and goal locations on Voronoi graph, and then by following the shortest path on the graph between those two points. Probabilistic roadmaps are constructed by joining any randomly generated large number of coordinates in free space only if it is easy to reach one node from the other. Later on, a discrete graph search can be performed from start location to target.

Alternative path planning algorithms may handle uncertainty arising from partial observability of the environment or stochastic effects of the robot's actions. If uncertainty is small enough to ignore, maximum likelihood estimates for most likely state can be computed. In order to accommodate uncertainty, problem can be modeled as a Markov Decision Process (MDP) in fully observable environments. Solution to MDP is an optimal policy telling the robot what to do in every possible state. However, partial observability makes the problem harder, and turns the robot control problem into partially observable MDP (POMDP), for which robot usually maintains an internal belief state. Solution to POMDP is a policy defined over robot's belief state. Though, techniques solving POMDPs are not applicable to robotics since no techniques are known for continuous spaces, and discretization can not be coped with by the known ones. Consequently, to keep the pose uncertainty of robot to minimum, some heuristics like coastal navigation, which requires the robot to stay near known landmarks to decrease its pose uncertainty, may be imposed. Thus, uncertainty in the mapping of new landmarks nearby is decreased, and this enables the robot to explore more territory (Russell and Norvig, 2003).

## 1.2 VISION-BASED NAVIGATION

Vision brings about much humanoid approach to navigation problem. While moving via their sonar or range sensors in an environment, robots are similar to blind

people who try to understand the surrounding with a feeling of touch. However, in spite of being more computationally expensive and less accurate compared to sonar or range sensors, visionary sensors like cameras can also be used instead of other sensors enabling safe interaction with the environment (Riisgaard and Blas, 2005; Hager, 2008).

Furthermore, they provide much more consciousness for robots about the environment so that robots can follow some objects including humans (Jia et al, 2006), determine far-range drivable path when driving on highways (Bradski and Kaehler, 2008b), or classify objects before performing related tasks on them such as treating dishes accordingly as unloading a dishwasher (Velat et al., 2007). Despite of the advantages of performing such advanced tasks, computational and accuracy deficiencies of visionary applications have directed some researchers towards fusion approaches, which combine camera input with other sensors rather than employing pure vision-based approaches (Jia et al, 2006; Bradski and Kaehler, 2008b).

There are three different approaches to vision based localization: absolute localization, incremental localization, and localization derived from landmark tracking. In absolute localization, the robot's initial pose is unknown so the navigation system must construct a match between the observations and expectations as derived from database. Due to uncertainties with the observations, it is possible for the same set of observations to match multiple expectations. The resulting ambiguities may be solved by methods such as Markov localization, partially observable Markov processes, Monte Carlo localization, multiple hypotheses Kalman filtering based on mixture of Gaussians, using intervals for representing uncertainties or by deterministic triangulation. Absolute localization is to be contrasted with incremental localization in which it is assumed that location of the robot is known approximately at the beginning of the navigation session, and the goal of the vision system is to refine the location coordinates. Yet in landmark tracking localization of a map-based navigation, correlations are used to keep track of the landmarks in the consecutive images that are recorded as the robot moves (DeSouza and Kak, 2002).

Computations involved in the vision based localization that is the first key to solving navigation problem can be divided into the following four steps. The third one is the most challenging of all since it requires a search that can usually be constrained by prior knowledge of the landmarks and by any bounds that can be placed on the

uncertainties in the position of the robot. First step is acquiring and digitizing camera images. Second one is detecting landmarks, which involves extracting edges, smoothing, filtering and segmenting regions. In third step, matches between observation and expectation are established by trying to identify observed landmarks in the database for possible matches. Finally, system needs to calculate its position as a function of the observed landmarks (DeSouza and Kak, 2002).

### **1.3 CURRENT SYSTEMS**

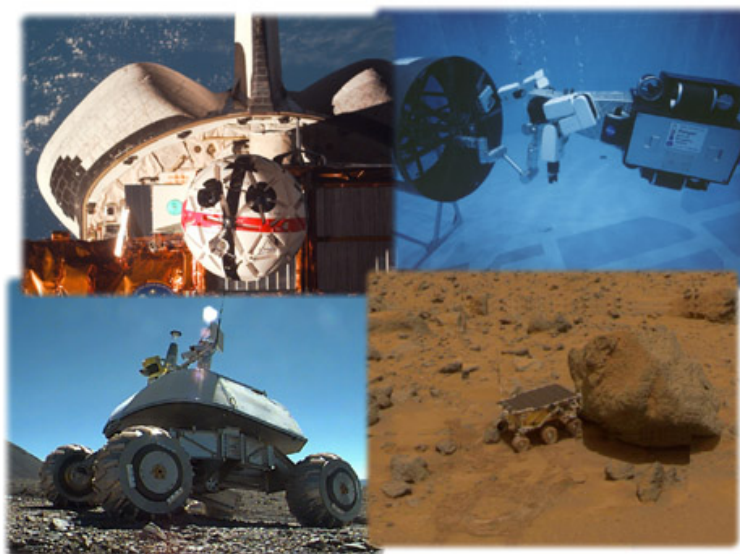
Systems based on mobile robot navigation are applied in different domains such as industry, human assistance, transportation and exploration or entertainment. For any of those systems, vision can be used as a useful means to carry out the designated missions properly.

In industry, autonomous mining robots have been found to be faster and more precise than people in transporting ore in underground mines. Mobile robots have also been used to generate high-precision maps of abandoned mines and sewer systems, or to detect malfunctions of utility poles.

Assisting robots are beneficial especially in hazardous environments for humans such as cleaning up nuclear waste, detecting mines in minefields and entering collapsed structures as members of human search and rescue crews. Besides, robots as vacuum cleaners, lawn mowers or golf caddies to perform daily tasks autonomously are already commercially available. In addition to these, service robots which require unpredictable and dynamic environment handling, and human interaction have been put to operate in public places like museums, shopping malls, or trade fairs as tour guides.

Robotic transportation applications vary from unmanned aerial vehicles (UAVs) that deliver objects to locations that would be hard to access by other means, to automatic wheelchairs, or legged walking robots carrying handicapped or elderly people who are unable to control the system by themselves. A primary example to indoor applications is robots deployed in hospitals to transport food and other items, and car-like robotic systems navigating autonomously on highways or across terrains

(ULVs) are some instances of outdoor transportation applications. The NAVLAB project initially developed by Thorpe et al. (1987), its neural network equipped version ALVINN (1989) and Darpa Grand Challenge winner Stanley of Stanford University by Thrun et al. (Bradski and Kaehler, 2008b) are some of vision-aided autonomous car navigation systems that have been developed so far. Furthermore, mobile robots have been built up to explore places hard to access for people. Planetary exploration robots or undersea exploration vehicles (AUVs) are cases in point. Figure 1.1 shows some exploration and transportation robots that have been designed. The lower right figure patch is the rover of Mars Pathfinder mission of NASA first landed in July 1997 whose navigation was carried out in corporation with a lander unit and consisted of four different functions that are goal designation, path selection, rover localization, and hazard detection.



**Figure 1.1** Mobile robots deployed in transportation and exploration domains.

Finally, human-like robot Asimo of Honda and dog-like robot toy AIBO of Sony are two sample applications in entertainment domain. The latter one is also used as a research platform in most of AI labs around the world.

## 1.4 BACKGROUND AND MOTIVATIONS

Mobile robotics is a multi-disciplinary research area involving machine, computer and electronic engineerings, cognitive psychology, recognition, and neurology. The fields that are open to research are moving, control and directing, human-robot interaction, learning, and adaptation.

From computer engineering perspective, computer science with many of its subfields can contribute to the various phases of mobile robot navigation problem. Computer vision techniques can enhance tracking and obstacle avoidance capabilities of the robots via employment of cameras. Many AI and machine learning algorithms can be adapted to decision making and path planning procedures to combine internal knowledge with environmental observations. Human-computer interaction can aid determining the ways for robots to more naturally interact with humans, and embedded systems can be designed for more efficient and reliable handling of actuators and sensors. Together with the support of other subfields such as Computer Graphics or Robotics, computer science is a crucial contributor to make such a robotic system up, and promises great deal of research opportunities in this area.

Even though the current systems may outperform fully human-guided systems, most of those systems are still semi-automatic or teleoperated, which means a human operates them by remote control, in order for increased accuracy. Moreover, they require environmental modifications such as inductive loops on the floor, active beacons, and bar-code tags, or some prior knowledge and representations about the environment for their operation. Consequently, for the sake of increased flexibility the design of robots that can use natural cues, instead of artificial ones in order to navigate is an open challenge in robotics. Addressing the navigation problem based on natural landmark tracking in an environment with a single camera, it is believed that this study may contribute to the realization of more flexible visionary robotic systems in the future.



## **CHAPTER 2**

### **LITERATURE REVIEW**

A great deal of research has been made on visionary navigation problem, and important improvements and findings have been obtained so far. Even though understanding the way the studies are implemented in detail is fairly hard at a time, it was useful to be aware of the basic ideas the studies depend on in order for providing this research with a reasonable direction. Consequently, first of all some general information about the studies is given through the following paragraphs. Section 2.1 classifies the approaches of the studies to vision based navigation problem, and tries to explain them together with some examples. Being parallel to the approach of this research to the problem, Section 2.2 explains some of the methods to reliably extract robust features from video frames, and Section 2.3 mentions about the visionary SLAM studies depending on those extracted features.

Bradski and Kaehler (2008b) identified robot-vision signal processing primitives and their associated classes of methods and presented a robot-vision example so-called the Stanley robot racing car. They indicated that main signal processing primitive classes are filtering, shape analysis, density modeling, clustering, and tracking. Filtering methods aim removing less important data from image to help subsequent processing stages such as edge detection and tracking. Some filtering methods include convolution in the spatial or frequency domains, smoothing using Gaussian pyramids, patch matching via cross correlation, and despeckling. Shape analysis methods aim at identifying image entities for further processing. They can be either boundary based to determine edges or contours of the object, or region based to determine the area covered by the object. Density modeling means identifying a distribution of image or sub-image

related features such as color or pixel location. Simple histogram binning is often used in robot vision applications. It can be performed by collecting distribution of normalized color values in the image. Once histogram has been obtained the distribution of objects according to the measures can be computed. For clustering two most often used algorithms in robot vision were given as k-means and mean-shift. Also, widely used tracking method was mentioned to be the Lucas and Kanade (1981) tracking based on Harris corner detector, which can be refined by embedding the tracker in a Kalman or particle filter.

Similarly, simple distribution modeling followed by clustering played an essential part in success of the Stanley robot racing car of Stanford University. It enabled the car to use cameras to see out beyond the range of laser range finder, allowing robot to safely drive faster than laser range finders alone allowed. The distribution of the colors in the drivable patch determined by laser range finder was modeled by the vision system using k-means algorithm. A Gaussian color model was fit to each of the k color centers, and then models were normalized to obtain probability of a pixel being part of a road object given the measures in a Bayesian decision model. The vision and laser maps were fused together to obtain drivability map which was passed to the planner to make path and speed decisions.

Jia et al. (2006) surveyed the developments of the last 10 years in the area of vision based target tracking for autonomous vehicles navigation. They emphasized that if the search goal is to send an autonomous vehicle from one coordinate location to another, there is sufficient accumulated expertise in the research community today to design algorithms which could do that in a typical environment; but if the goal is adapting to dynamically changing environment during navigation such as chasing or following moving targets, avoiding unpredictably positioned obstacles, and stopping at a stop sign under varying illumination and background conditions, it is still the central research problem. Most algorithms developed for the navigation of autonomous land vehicles were categorized as visual landmarks tracking algorithms, human following algorithms, target tracking for localization and map building, target tracking with pan-tilt camera platforms, and target tracking for multiple mobile robots cooperation. The first and the third categories, that are most related to this research, are elaborated as

follows, and later how the disadvantages of the existing algorithms can be eliminated is discussed.

For visual landmarks tracking algorithms, visual landmarks were divided into two classes: natural or artificial. Generally, natural landmarks are selected in the scenes in consideration of their particular characteristics. Autonomous vehicles learn those characteristics or keep the features of the landmarks in memory, and recognize them using the neural network or some other matching technique as they move. On the other hand, an artificial landmark is often designed with a specific pattern or color to ease its detection. Simple artificial landmark model can be used for self localization of indoor mobile robots.

Regarding target tracking for localization and map building, localization was defined as determining the position of an object within a reference coordinate system, and tracking was described as constructing a trajectory given a collection of spatially and temporally coherent locations. Localization, mapping, and moving object tracking serve as the basis for scene understanding which is in turn a key prerequisite for making a robot truly autonomous. Among land vehicle localization and map building applications are real-time mobile navigation systems depending on vision based SLAM, localization of indoor mobile robots depending on natural landmark models, or robust tracking algorithms employing edge detectors and Lucas-Kanade algorithm for tracking of the landmarks.

To combine the advantages of the existing methods and to compensate their disadvantages, equipping vehicles with different sensors such as inertial motion sensors, cameras, laser scan, radar or GPS, a fusion of information methodology can be implemented for robust feature extraction and target tracking. But one difficulty of fusion is that because sensors data can not arrive at the same time to the fusion algorithm, different time-stamps may lead to delayed measurements problem.

DeSouza and Kak (2002) surveyed the developments of the last 20 years in the area of vision for mobile robot navigation. They basically separated the progress made during those years as vision based navigation for indoor and vision based navigation for outdoor robots. Indoor applications were further separated into three broad groups: map-based navigation, map-building-based navigation, mapless navigation. Even

though both indoor navigation and outdoor navigation involve obstacle avoidance, landmark detection, map-building or updating, and position estimation; outdoor navigation differs from indoor navigation in that a complete map of the environment is hardly ever known a priori, and the system has to cope with the objects as they appear in the scene without prior information about their expected position. Thus, they divided outdoor navigation applications into two different classes: outdoor navigation in structured and in unstructured environments.

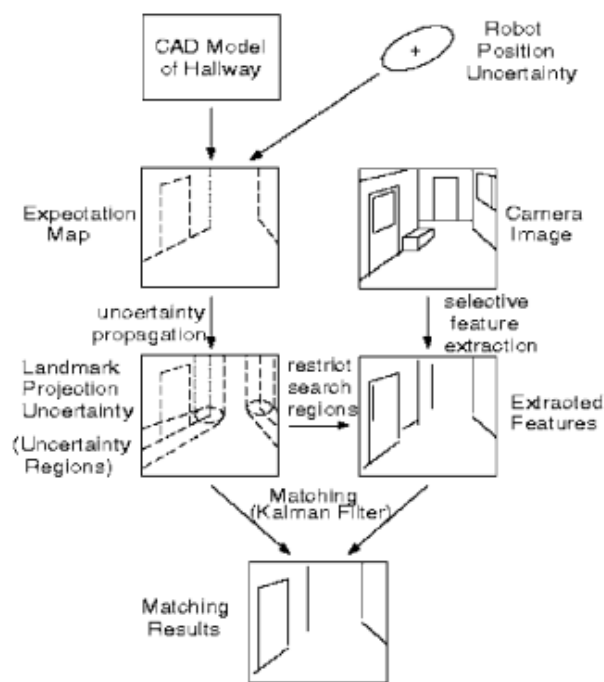
## **2.1 TYPES OF VISIONARY NAVIGATION**

Vision-based indoor navigation applications are considered under map-based, map-building, and mapless navigation in the following subsections. Each type of applications is briefly explained and exemplified mostly again with the help of DeSouza and Kak's survey (2002).

### **2.1.1 Map-based Navigation**

Atiya and Hager (1993) recognized some entities in camera images that stay invariant with respect to the position and orientation of the robot as it travels in its environment. Their absolute localization idea was that given a triple of point landmarks on a wall in the environment, if all three of these points could be identified in each image of a stereo pair, then the length of each side of the triangle, and the angles between the sides would stay invariant as the robot moves to different positions with respect to these three points. So the length and the angle attributes associated with a triple of landmark points would be sufficient to identify triples, and to set up correspondences between the landmark points in the environment and the pixels in camera images. Once such correspondences were established, finding the absolute position of the robot simply became an exercise in triangulation. However, that the coordinates of landmark points may not be known exactly in the world, that the pixel coordinates of the observed image points may be subject to error, and that there may be ambiguity in establishing correspondences between the landmark triples and the observed pixel triples were some of the problems to deal with in this approach.

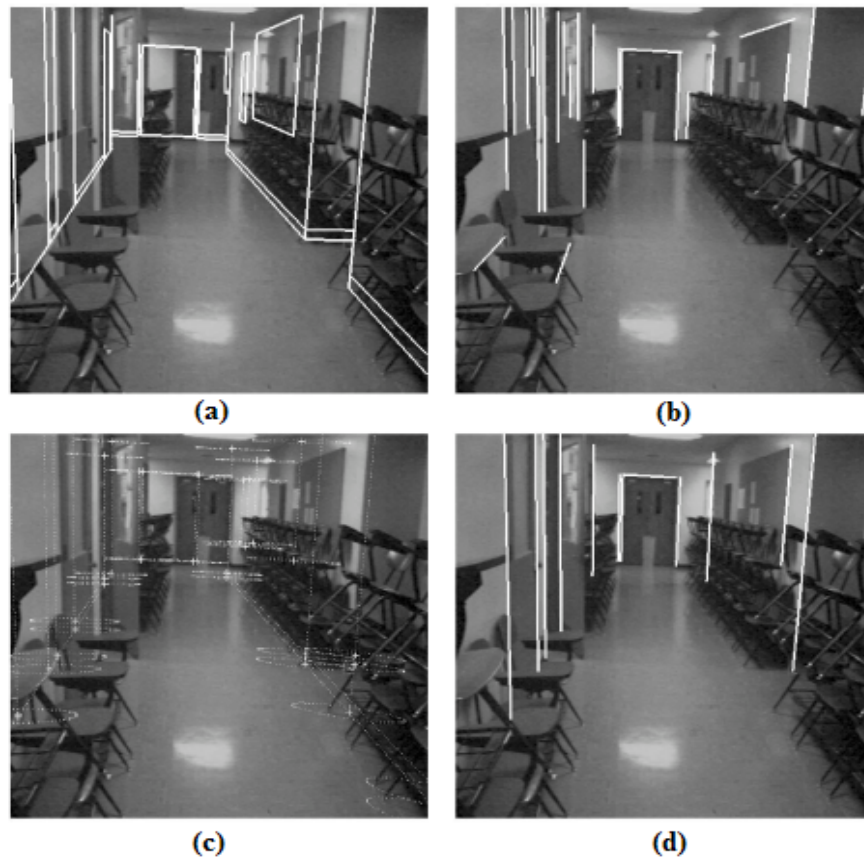
Since in incremental localization for a large number of practical situations the initial position of the robot is known at least approximately, in such cases the localization algorithm must simply keep track of the uncertainties in the robot's position as it executes the motion commands and must use its sensors for a new fix on its position when uncertainties exceed some threshold. One such system achieving incremental localization by using geometrical representation of space and a statistical model of uncertainty in the location of the robot was called FINALE by Kosaka and Kak (1992). Figure 2.1 presents the processing steps of the system.



**Figure 2.1** Processing steps of FINALE system (Kosaka and Kak, 1992).

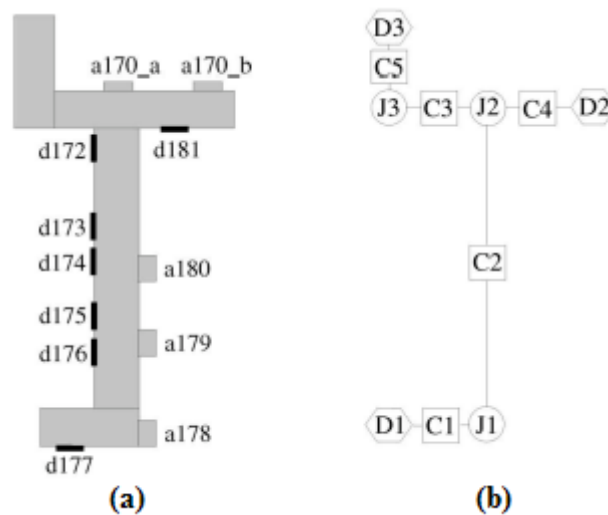
In FINALE system, uncertainty in robot's position  $p(x,y,\theta)$  was represented by a Gaussian distribution, so uncertainty in position was characterized by the mean  $\bar{p}$  and the covariance  $\Sigma_p$ . Translational and rotational motions of the robot in response to commands were also characterized and parameterized to account for the slippage in the wheels so that uncertainty in the robot's position can be identified better. To determine where to look in the camera image given a landmark in the environment, the uncertainty in the position of the robot had to be projected into the camera image through calibration matrix.

Based on the current value of robot position  $\bar{p}$ , an expectation map of the world was prepared. For each end point of a vertical edge in expectation map, using covariance matrix associated with the pixel coordinates of a single point landmark in the scene, one unit of Mahalanobis ellipses were computed to indicate uncertainty regions. Projecting robot's positional uncertainties into camera image were easily extended to mid-level features such as straight edges using Hough space where each straight line feature in the environment was represented by a single point. Finally, edges extracted in the vicinity of model edges in Hough space were matched with the model edges via a model based Kalman filter derived from a linearized version of a constraint equation when the equation was satisfied by the parameters of a straight line in the environment and the Hough space parameters of the corresponding line in the camera image. It was this match through which robot could localize itself updating the statistical parameters of its position. Some of important intermediate steps of the system are shown in Figure 2.2 when robot localizes itself using Kalman filter based approach.



**Figure 2.2** (a) The camera image and the superimposed expectation map (b) Output of the model-guided edge detector (c) Uncertainty regions associated with the ends of the model line features (d) Matching after Kalman filtering (Kosaka and Kak, 1992).

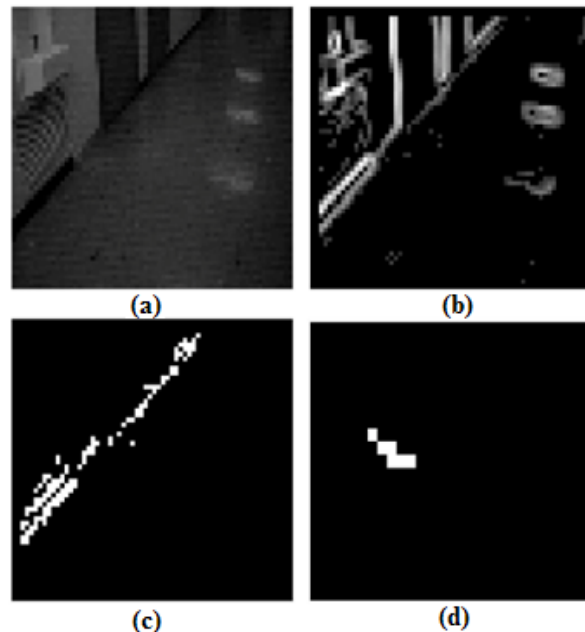
An entirely different approach to incremental localization, NEURO-NAV, by Meng and Kak (1993), utilized a topological representation of the environment. As shown in Figure 2.3, the topological representation employed was a graph data structure using three kinds of nodes in the form of squares, circles and diamonds to represent corridors, junctions and dead ends, respectively. Each node in the graph was attributed. For example, in the figure the main central corridor C2 has an attribute of left landmarks which is a list of pointers to doors d176, d175, d174, d173, and d172, and links of the graph are attributed to contain information regarding the physical distance between the landmarks.



**Figure 2.3** (a) Physical structure of a hallway segment showing the doors and alcoves  
(b) The topological representation of the hallway in (a) (Meng and Kak, 1993).

NEURO-NAV consisted of the two modules: Hallway Follower and Landmark Detector. Each module was implemented using an ensemble of neural networks. As robot executed the commanded motions, Hallway Follower module kept the robot in a parallel path with respect to the walls while Landmark Detector performed searches to keep track of the landmarks contained in the attributed node. Inside Hallway Follower there were neural networks, one of which was called “corridor-left” which could detect from camera image the edge between the floor and the left side wall, and could output the appropriate steering angles keeping the robot approximately parallel with respect to the left wall. To achieve that the camera images were first down sampled from 512x480 pixels matrix to 64x60 pixels matrix to speed up computations, and then Sobel operator

was used for edge detection. Later Hough transform of the edges were taken, and different regions of Hough space were fed into different neural networks. The image in Figure 2.4(a) was taken when the robot was pointed somewhat towards the left wall. So the image is rich in edges that correspond to the junction between the left wall and the floor. In the Hough map these edges occupied cells that are mostly in the left half of the map, and such observations were the ones determining which regions of the Hough space should go to what neural networks. NEURO-NAV's all neural networks were simple three layered feed forward networks using back-propagation algorithm trained by a human operator.



**Figure 2.4** (a) An example of the hallway as seen by the camera (b) The output of the edge detector (c) The relevant floor edges (d) The Hough map (Meng and Kak, 1993).

Since the output nodes of the neural networks were between 0 and 1, and thus were fuzzy in nature, Pan et al. (1995) replaced NEURO-NAV's rule based supervisory controller by real-time fuzzy expert system FUZZY-NAV. It used three linguistic variables distance-to-junction, distance-to-travel and turn-angle.

### 2.1.2 Map-building Navigation

Model descriptions possessed by map-based navigation systems are not always easy to generate, especially the ones providing metrical information. So many



researchers proposed automated or semi-automated robots exploring their environment and building internal representation of it.

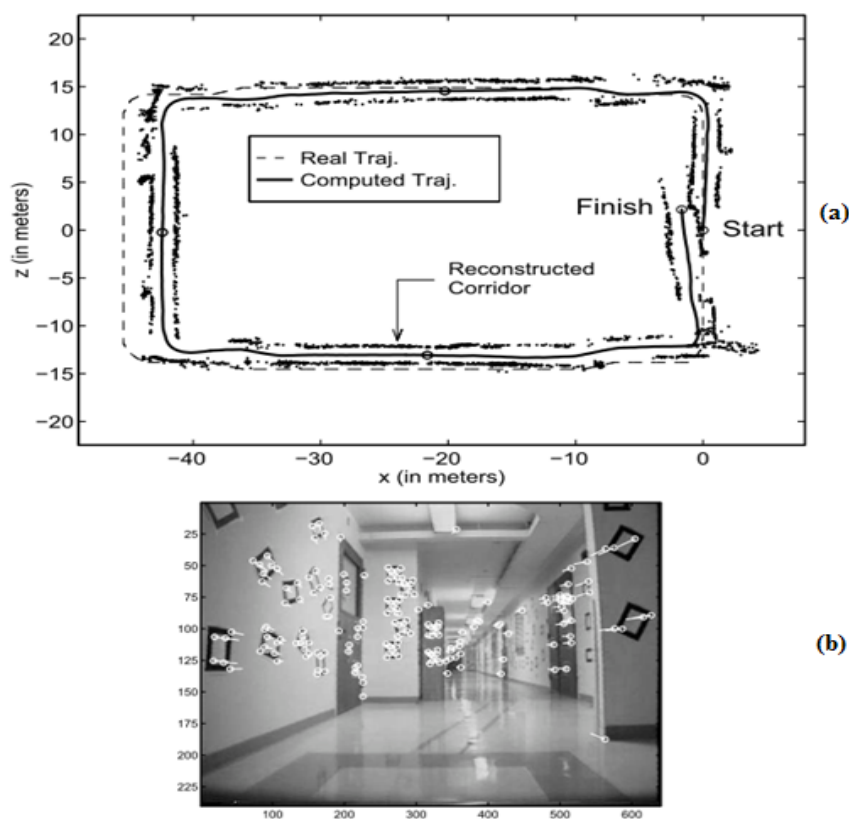
The first attempt at robotic map-building was the Stanford Cart by Moravec (1983). The Stanford Cart used a single camera to take nine images spaced along a 50 cm slider. Next, an interest operator was applied to extract distinctive features in the images. These features were then correlated to generate their 3D coordinates. The world was represented by the 3D coordinates of the features plotted in a grid of two square meter cells. The features were tracked through iterations of the program, and marked in the grid and in the image plane. Although this grid indirectly represented the position of obstacles in the world and was useful for path planning, it did not provide a meaningful model of the environment.

Bouget and Perona (1995) assessed the usefulness of single camera recursive motion estimation techniques for vehicle navigation in the absence of a model for the environment. They extended a recursive motion estimator to handle scale estimation and examined experimentally the accuracy with which the motion and position of the vehicle may be computed on an 8000 frames indoors sequence.

They decomposed the general scheme applied for full recursive rigid motion recovery into 4 successive stages. First stage was automatically extracting some distinguishable feature points from the images and tracking them from frame to frame which gave the image flow information. Using this flow, second stage computed the motion parameters which included a scale factor ambiguity from the norm of translation. This ambiguity was resolved by using scenery information which is also called 3D structure. The third stage was the actual structure reconstruction, and the fourth was the scale factor propagation. For image flow computation a multi-scale version of the Lucas and Kanade algorithm was used.

Experiments were performed on an image sequence taken with a CCD video camera mounted onto a cart moving along a closed corridor to reproduce indoor. The cart was simply pulled by two operators while another operator was sitting on it. The camera was such that it was pointing approximately in the direction of the motion. The velocity of the motion was about 4 km/h, and the corridor was 2 meters wide. To measure the performance of the algorithm, they extracted from the reconstructions two

quantities: the computed angle of turn which was ideally 90 degrees and the final computed vertical deviation which was ideally 0 meter, since the motion was planar. For a whole round-trip experiment Figure 2.5 (a) shows a top view of the complete reconstructed trajectory and corridor. The dashed lines represent the real trajectory, and the solid lines the estimated one. The dots are the reconstructed positions of the features on the walls which have most of the errors at the turns. Figure 2.5 (b) displays an image of the sequence with its attached point features and flow, and the current motion. The two experimental conditions; a time baseline  $k=10$  which means that camera frames are grabbed in 6 Hz and  $N=40$  which is the number points used for motion estimation, were found to be optimal. The results indicated that in addition to the quality of the trajectory the different walls of the corridor were accurately reconstructed.



**Figure 2.5** (a) A top view of the reconstructed trajectory and corridor (b) An image with the features and the flow (Bouget and Perona, 1995).

Thrun (1998) proposed an integrated approach that seeks to combine the best of the occupancy-grid-based and the topology-based approaches. His system first learned a grid-based representation using neural networks and Bayesian integration. The grid-based representation was then transformed into a topological representation.

The SLAM studies to be mentioned in Subsection 2.3 can also be regarded as map-building applications since as the robot moves the algorithm tries to model the world taking the uncertainties in positions of the robot and landmarks into consideration. These studies are given a special focus due to their relevancy to the proposed system.

### **2.1.3 Mapless Navigation**

In mapless navigation systems no map of the environment is formed. Thus no prior information regarding the environment exists. The needed robot motions are determined by observing and extracting relevant information about the elements in the environment without knowing their absolute or relative positions. Consequently, while in map-based systems it is easy to establish meaningful navigation goals for the robot, most robotic systems are limited to just roaming in mapless systems. Because, an internal map representation of a structured environment can be used to conveniently specify different destination points for the robot. On the other hand, for the mapless navigation, most of the time robot only has access to a few sequences of images to get to its destination or some predefined features of the target goals to be tracked.

The prominent mapless navigation techniques are optical flow-based and appearance-based navigations. Optical flow-based systems are developed by being inspired of the visual behavior in insects. Due to insects' extremely narrow binocular field, depth information that can be extracted from the sight is minimal, whereas their sensitivity to motion parallax is high making them much more aware of time-to-crash rather than the distance to environment obstacles. However, appearance based matching is memorizing the environment by storing the images or templates of the environment and associating those images with commands or controls that will lead the robot to its final destination.

Santos-Victor et al. (1993) employed a divergent stereo approach in their robot called robee mimicking the centering reflex of a bee. If the robot was in the center of a corridor, the difference between the velocity of the images seen with the left eye and the right eye was approximately zero, and the robot stayed in the middle of the corridor. However if the velocities were different, the robot moved towards the side whose image changes with smaller velocity. With regard to the robotic implementation the basic idea

was to measure the differences between image velocities computed over a lateral portion of the left and the right images, and to use this information to guide the robot. To compute the average optical flows on each side, the fundamental optical constraint (Equation 2.1) proven by Lucas and Kanade was used. In the equation,  $u$  and  $v$  are the horizontal and the vertical flow components. Having the knowledge that the robot moved on a flat ground plane so that flow along the vertical direction was regarded as zero, Equation 2.2 was obtained where  $I_t$  and  $I_x$  are the time and  $x$ -spatial derivatives of the image respectively.

$$\frac{\delta I}{\delta x} u + \frac{\delta I}{\delta y} v + \frac{\delta I}{\delta t} = 0 \quad (2.1)$$

$$u = -\frac{I_t}{I_x} \quad (2.2)$$

For the algorithm to work, first, the images were smoothed with respect to space and time prior to computation of any derivatives. Then the time derivative was computed simply by subtracting two consecutive smoothed images. At each iteration of the control loop, five 256x256 stereo images were grabbed at video rate and used to compute the time smoothed images. Then last two images on each side were used to find the average optical flow vectors. This average was calculated over a sub-window of 32x64 pixels on each side. Finally, by observing that right and left flows had opposite directions, the comparison was given as in Equation 2.3 where  $T_M$  is the robot forward motion speed, and  $Z_R$ ,  $Z_L$  provide the horizontal projections of these motions into the right and left images. The difference between right and left average optical flows was input to a PID controller to keep the robot centered in a hallway.

$$e = u_L + u_R = T_M \left( \frac{1}{Z_R} - \frac{1}{Z_L} \right) \quad (2.3)$$

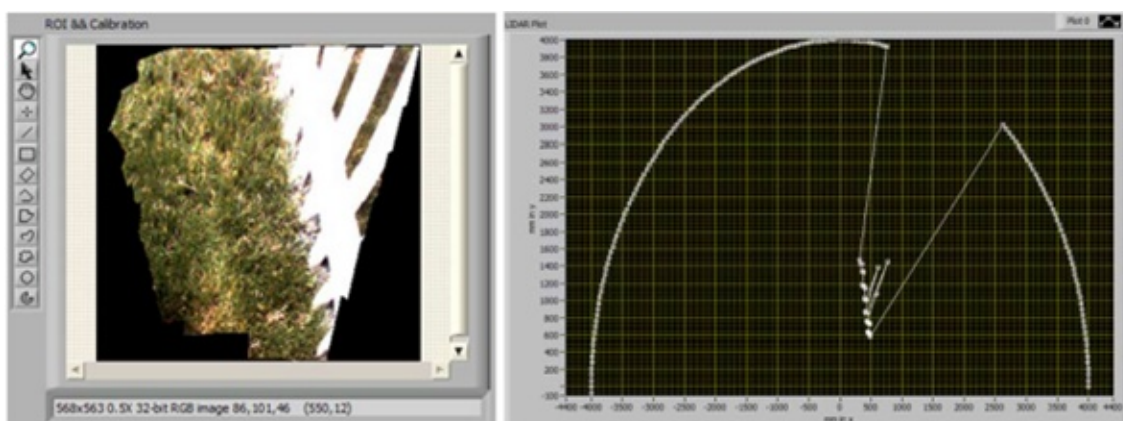
Though, equations given were oversimplified in the sense that they were applied when only two cameras were pointing in symmetrically divergent directions with respect to the direction of motion, and they could not be satisfied during rotational motions. The technique also ran into difficulties if there was insufficient texture on the walls of a corridor since the optical flow calculation depends on the existence of some texture. Apart from this study, Dev et al. (1997) implemented wall following application by extracting depth information from optical flow.

An appearance-based navigation study was implemented by Matsumoto et al. (1996) by using a sequence of images and a template matching procedure to guide robot navigation. Subwindows extracted from down-sampled versions of camera images were used to form the sequence of images that worked as a memory of all the images observed during navigation. Each image in this sequence was associated with the motions required to move from the current position to the final destination. After a sequence of images was stored, and the robot was required to repeat the same trajectory, the system compared the currently observed image with the images in sequence database using correlation processing on a dedicated processor. Once a match occurred, the displacement in pixels between the view image and the template image was computed in order to determine real world displacements and angles to be used in steering commands.

Schepelmann et al. (2009) investigated the use of image hue and intensity to design a robust, real-time vision based obstacle detection system- CWRU Cutter for use during a competition. The competition emulated a backyard environment and had a variety of common obstacles like fence, flower bed lining, and a mobile obstacle inside the contest course. To consistently recognize an object such as a green lawn in an image, using simple RGB color thresholding was problematic due to recognition limitations in illuminated and shaded areas. However unlike RGB planes, the hue plane was found to be relatively insensitive to changing lighting conditions and shadows. Instead of indicating how much red, green and blue are present in a color; hue is an indication of how much of a certain color is present at a pixel. So all green containing pixels could be classified under similar hue values in a partially shaded area, and hue plane could be employed in order to identify colors with non-intersecting hue ranges.

Image processing for obstacle detection on CWRU Cutter was accomplished through a number of steps. First, before mowing, a sample image of the competition field was taken, and the mean hue value of grass on image was calculated for reference. Second, the hue plane was extracted from the image, and a threshold was applied within  $\pm$  one standard deviation of the mean hue value of the grass. The threshold extracted the matching pixels to a binary image with 1 representing passable terrain and 0 representing potential obstacles. Third, binary image dilation and hole filling operations were performed on the array to remove small false positives in the image. Finally, the

resulting array was converted into a range image in polar coordinates. So mowable terrains and obstacles around the lawnmower were detected as  $(r,\theta)$  pairs by considering the middle element of the bottom row of the array the  $(0,0)$  location of the camera. This 1-D range array was referred to as “pseudo-LIDAR” scan since it was generated through images unlike the hardware sensory unit LIDAR for obstacle detection (Figure 2.6).



**Figure 2.6** Incoming calibrated camera image of the white competition fence (left) and resulting pseudo-LIDAR scan (right) (Schepelmann et al., 2009).

## 2.2 ROBUST FEATURE EXTRACTION

Bradski and Kaehler (2008a) pointed out that particular objects are usually identified in still images. In order to understand motions of those objects, two components: identification and modeling are needed. However, tracking things that have not yet been identified is a related problem. Techniques for tracking unidentified objects typically involve tracking visually significant key points or features, rather than extended objects. One method for achieving this is the Lucas-Kanade (1981) technique which is often referred to as sparse optical flow.

Once the feature extraction methods to be mentioned throughout this subsection are applied, the result is an array of pixel locations that are hoped to be found in subsequent frames of video or in images. Matching those features in frames is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking (Lowe, 2004).

In a video frame, there can be many local features to track. Obviously, if a point is picked on a large blank wall, then it is not easy to find that same point in the next frames. On the other hand, if a point or feature, that is unique and parameterizable in such a way that it can be compared to other points in another image, is picked, then it is a pretty good chance of finding that point again. In Figure 2.7, the points in circles are good points to track whereas those in boxes are poor choices even though they are sharply defined edges.



**Figure 2.7** Good and poor choices of features (Bradski and Kaehler, 2008a).

To more powerfully distinguish between ordinary edges and trackable features, points having strong derivatives in two orthogonal directions, which are called corners, are searched. The most commonly used definition of a corner was provided by Harris and Stephens (1988). This definition relied on the matrix of the second-order derivatives of the image intensities ( $\partial^2x$ ,  $\partial^2y$ ,  $\partial x\partial y$ ). Second-order derivatives of all points were computed through Hessian matrix (Equation 2.4), and then those derivatives were combined to form the Hessian image.

$$H(p) = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial y \partial x} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \quad (2.4)$$

For the Harris corner, the autocorrelation matrix of the second derivative images over a small window around each point was defined as in Equation 2.5 where  $w_{i,j}$  is a

weighting term that can be uniform but is often used to generate a circular window, or Gaussian weighting.

$$M(x, y) = \begin{bmatrix} \sum_{-K \leq i, j \leq K} w_{i,j} I_x^2(x+i, y+j) & \sum_{-K \leq i, j \leq K} w_{i,j} I_x(x+i, y+j) I_y(x+i, y+j) \\ \sum_{-K \leq i, j \leq K} w_{i,j} I_x(x+i, y+j) I_y(x+i, y+j) & \sum_{-K \leq i, j \leq K} w_{i,j} I_y^2(x+i, y+j) \end{bmatrix} \quad (2.5)$$

Then the corners were places in the image where the autocorrelation matrix of the second derivatives had two large eigenvalues. In essence, this meant that there is texture (or edges) going in at least two separate directions centered around such a point, just as real corners have at least two edges meeting in a point. Second derivatives were useful because they did not respond to uniform gradients. Another advantage of this definition was that, when considering only the eigenvalues of the autocorrelation matrix, quantities that were invariant also to rotation were considered, which is important because objects being tracked might rotate as well as move. This approach by Harris was later improved by Shi and Tomasi (1994) shedding light on that good corners result as long as the smaller of the two eigenvalues is greater than a minimum threshold.

### 2.2.1 Scale Invariant Feature Transform (SIFT)

Lowe (2004) described image features that are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint, which made them suitable for matching differing images of an object or scene. They were well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. In addition, being highly distinctive the features allowed a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition. This approach has been named as the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale-invariant coordinates relative to local features.

Figure 2.8 demonstrates an object recognition implementation through SIFT. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.





(a)



(b)



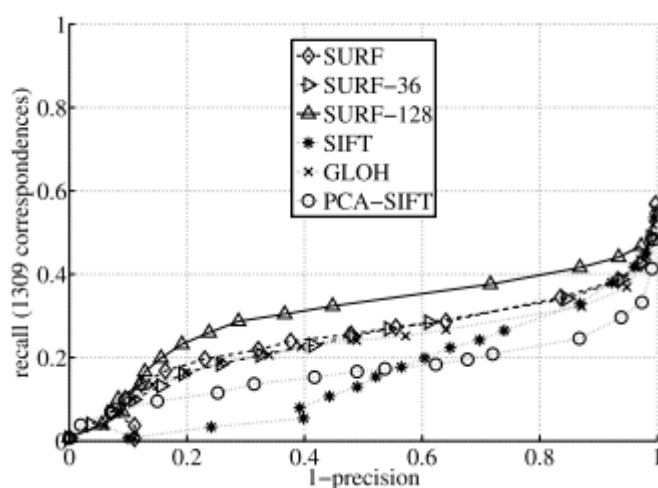
(c)

**Figure 2.8** (a) The training images for two objects (b) Training objects in a cluttered image with extensive occlusion (c) The results of recognition (Lowe, 2004).

### 2.2.2 Speeded-Up Robust Features (SURF)

Bay et al. (2006) presented a scale and rotation invariant interest point detector and descriptor called SURF which approximated or even outperformed previously proposed schemes with respect to repeatability, distinctiveness and robustness yet could be computed and compared much faster. They focused on scale and rotation invariance since perspective effects and skew scaling are second order effects. They stressed that in some cases even rotation invariance could be left out to increase efficiency and discriminative power, especially in mobile robot navigation applications where the camera often rotates about the vertical axis.

Their detector was based on the Hessian matrix but used a very basic approximation. It relied on the integral images to reduce the computation time. The descriptor described the distribution of Haar-wavelet responses within the interest point neighborhood. Only 64 dimensions were used for the descriptors reducing the time for feature computation. Additionally, a new indexing step based on the sign of Laplacian increasing the matching speed and robustness of the descriptor was presented. Figure 2.9 proves that using similarity difference matching technique SURF descriptor outperformed the other descriptors with sometimes more than 10% improvement in recall for the same level of precision where interest points were not affine invariant.



**Figure 2.9** The recall versus (1-precision) graph for different methods using similarity difference matching (Bay et al., 2006).

At the same time, SURF was faster to compute. The accurate version SURF-128 using a descriptor of 128 dimensions showed slightly better results than the regular SURF, but was slower to match. Again, for the same number of points, computations of the detector and the descriptor was 391 ms for SURF-128 whereas it was 1036 ms for SIFT. The timings were evaluated on a standard Linux PC (Pentium IV, 3GHz).

## **2.3 VISIONARY SLAM**

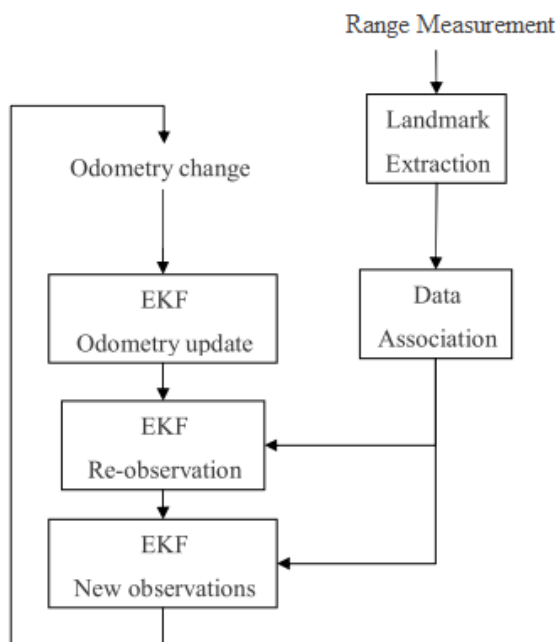
### **2.3.1 The SLAM Process**

SLAM is more like a concept than a single algorithm and is concerned with the problem of building a map of an unknown environment by a mobile robot while at the same time navigating the environment using the map. In order to be realized, it requires a mobile robot and a range measurement device. The SLAM algorithm consists of a number of steps: Landmark extraction, data association, state estimation, state update, and landmark update, and there are different approaches to solve each of the smaller parts (Riisgaard and Blas, 2005).

The goal of the SLAM is to use the environment to update the position of the robot. Since the odometer of the robot that gives the robot's position is often erroneous it can not be directly relied on. The errors in odometry stems from some systematic errors a such as unequal wheel diameters, wheelbase uncertainty, or wheel misalignment and from some non-systematic errors such as traveling over uneven floors or wheel slippage due to slippery floor (Se et al., 2002). Then, the range measurement to the environment obstacles is used to correct the position of the robot. This is accomplished by extracting features that are commonly called landmarks from the environment and by re-observing them when the robot moves around. The EKF is the main algorithm of the SLAM process, and is responsible for localization of the robot based on the observed features. It keeps track of an estimate of the uncertainty in the robot's position and also the uncertainty in these landmarks that the robot has seen in the environment. An outline of the SLAM process is shown in Figure 2.10.

When the robot moves, the odometry changes, and the uncertainty pertaining to the robot's new position is updated in the EKF using odometry update. Then landmarks

are extracted from the environment at the new position of the robot. Later the robot attempts to associate these landmarks to the previous observations of landmarks. Reobserved landmarks are then used to update the robots position in the EKF. Landmarks which have not previously been seen are added to the EKF as new observations so that they can be re-observed later. At any point in these steps the EKF will have an estimate of the robots current position.



**Figure 2.10** Overview of the SLAM process (Riisgaard and Blas, 2005).

### 2.3.2 Visionary SLAM

Visionary SLAM applications employ cameras as the range measurement device. Depending on the number of cameras used they can be designed as monocular (single camera), binocular (two cameras) or trinocular (three cameras) systems. The robust feature extraction methods discussed in Section 2.2 are applied on camera frames in order to determine some landmarks in the environment. Then, these features are matched to each other to measure the landmark parameters, or coordinates, as accurately and efficiently as possible similar to any sonar or range sensor based system. Some of the systems also try to estimate the camera movements from the matches, and this method is known as visual odometry. Those landmark parameters together with

odometry data of robot are filtered in EKF system as mentioned to complete the SLAM cycle.

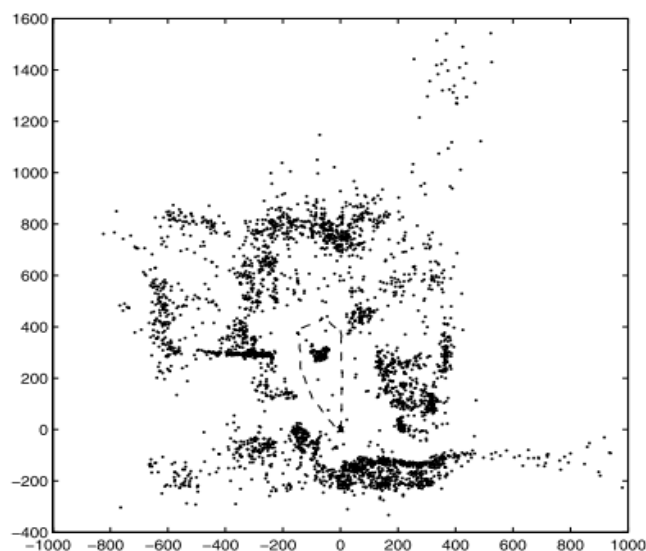
Se et al. (2002) described a vision-based mobile robot localization and mapping algorithm which used SIFT image features as natural landmarks in unmodified environments. With their trinocular stereo vision system, these landmarks were localized and robot egomotion was estimated by least squares minimization of the matched landmarks. They took into account the feature viewpoint variation and occlusion by storing a view direction for each landmark. Maintaining the error in estimates for the landmark positions and the robot pose was accomplished by Kalman filters.

They chose SIFT features over widely used Harris corner detector since the latter method is sensitive to scale of an image and therefore is not suited to be matched from a range of robot positions for building a map. The extracted SIFT features in each of the three images were stereo matched among the images. The constraints when performing the stereo match involved epipolar, disparity, orientation, scale and finally unique match constraints.

To build a map, the knowledge of how the robot had moved between the frames was needed in order to put the landmarks together coherently. Since the robot odometry gives a rough estimate and is prone to errors such as drifting or slipping, the robot odometry estimation of ego-motion was improved by matching SIFT features between frames. To find matches in second view efficiently, the odometry information was used to predict the region to be searched in the image for each match. The matched SIFT features were then used in a least-squares procedure to compute more accurate camera motion.

After SIFT features were matched between frames, a database map containing the SIFT features was maintained (see Figure 2.11), and this database of landmarks was used to match features found in subsequent views. Using the initial camera coordinate frame as a reference, all landmarks were taken relative to this frame. For each SIFT feature that had been stereo matched and localized in 3D coordinates, an entry containing the current 3D position of the SIFT landmark relative to the initial coordinate frame, the scale and orientation of the landmark, and a count indicating over how many consecutive frames a landmark was missed were hold in the database.

Experiments showed that these visual landmarks were robustly matched, robot pose was estimated, and a consistent three-dimensional map was built. Algorithm ran at around 2 Hz for 320x240 images on their mobile robot with a Pentium III 700 MHz processor, and majority of the processing time was spent on SIFT feature extraction.

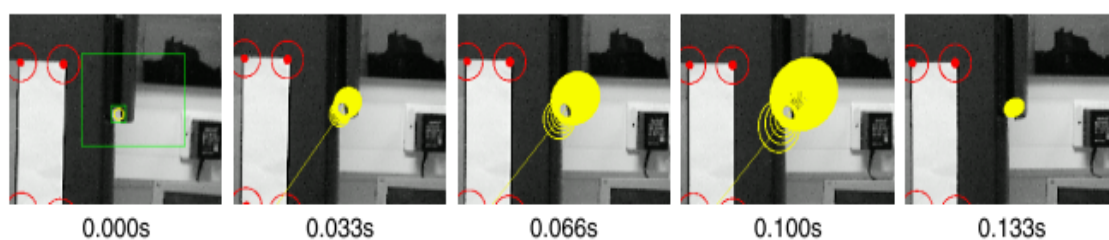


**Figure 2.11** Bird's-eye view of SIFT feature database (Se et al., 2002).

Davison (2003) presented a Bayesian framework for repeatable single-camera localisation via mapping of a sparse set of natural features using motion modelling and an information-guided active measurement strategy. He pointed out the success of EKF systems in SLAM problem but also the restriction of visionary SLAM applications to the smoothly moving robots with known control parameters and stereo vision. He stressed that the difficulty in real-time localization with a single camera since the number of features increase as the robot moves, and tracking all those would result in motion drift. Additionally, he mentioned that the key difference between constructing a motion model for a camera and a wheeled robot moving on a plane was that in the robot case one is in the possession of the control inputs driving the motion whereas no such information is available for a camera moving at a person's hand. Yet, for both cases, the motion models could be generated by using classical physics and probabilistic assumptions.

Visual feature measurements were realized by the approach of Davison and Murray (2002). Features were detected according to the method of Shi and Tomasi (1994) and matched using normalized sum-of-squared-difference correlation. For the features that were already registered in the SLAM map, using the estimates of the robot (or camera) position and feature position, the next measurements could be predicted using the pinhole camera model that is briefly explained in Subsection 3.3.1, and uncertainty in this prediction could be calculated. The knowledge of feature position uncertainty permitted a fully active approach to search the image for finding matches. Then, feature correlation occurring in the limited search regions maximized efficiency and minimized the chance of mismatch.

Feature initialization step in single camera SLAM was a difficult task, because 3D depth of features could not be estimated from one measurement. The approach adopted was to initialize a 3D line into map along which the feature must lie. Along this line a set of discrete depth hypotheses were made analogous to 1D particle distribution with 100 particles reflecting the indoor operation range. At subsequent time steps, these hypotheses were all tested by projecting them onto the image. Feature matching the hypotheses produced a likelihood estimate for each, and their probabilities were reweighted resulting in initialization of points into the map when the ratio of standard deviation of depth to depth estimate dropped below a threshold. A depth prior removed the need to search along the entire epipolar line, and improved the robustness and speed of initialization (see Figure 2.12). On the other hand, it is noted that most of the experiments carried out had involved mostly sideways camera motions, and this initialization approach would have performed more poorly with motions along the optic axis where little parallax was measured.



**Figure 2.12** Image search in successive frames during feature initialisation (Davison, 2003).

Map management involved the decisions when to add features to the map or when to delete them. The number of reliable features visible from any camera location was kept close to a predetermined value which was 6-10. Features were added to map if the number visible in the area the camera was passing was less than this threshold. Features were detected by running the image interest operator to locate the best candidate within a box of limited size (100x50 pixels) placed within the image. Position of the search box was chosen randomly with the constraints that it should not have overlapped with any existing features, and that any features should not have disappeared from the field of view immediately based on the current estimates of camera linear and angular velocities. A feature was deleted from the map if after a predetermined number of detection and matching attempts 50% of the time the feature was invisible. This pruned bad features which were not true 3D points or were often occluded. Problems only arised if mismatches occurred due to a similarity in appearance between clutter and landmarks, and this could potentially lead to catastrophic failure. Correct operation of the system relied on the fact that in most scenes very similar objects did not commonly appear close enough to lie within a single image search region.

Results showed that tracking of such kind was observed to be very repeatable and adaptable within a desktop scenario so that long periods of tracking of several minutes did not present any problem. On a 2.2 GHz Pentium processor, typical breakdown of processing time required at each frame at 30 Hz was computed to be 25 ms: 10 ms for correlation search, 5 ms for Kalman filter, and 10 ms for feature initialization.

Davison et al. (2007) extended the study of Davison (2003) on real-time single camera localization to recover the 3D trajectory of a monocular camera moving through a previously unknown scene, achieving real-time but drift-free performance inaccessible to SFM approaches. They also presented applications of their system named MonoSLAM to real-time 3D localization and mapping for a high-performance full-size humanoid robot and live augmented reality (AR) with a hand-held camera.

Regarding system initialization of the single camera SLAM, it was stated that there was no direct way to measure feature depths or any odometry from visual input. So they started from a target of known size allowing them to assign a precise scale to the estimated map and motion, rather than running with scale as a completely unknown factor. Having some features in the map right from the start meant that normal predict-



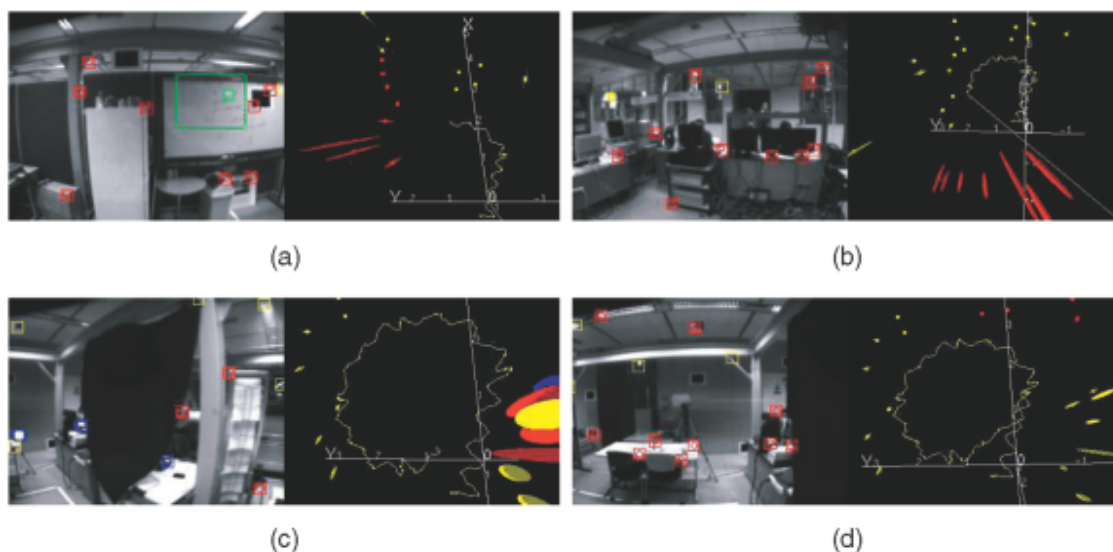
measure-update tracking sequence could be immediately entered in EKF without any special first step. Otherwise, without features to match, algorithm would get stuck to estimate the camera motion from frame one to frame two since features could not be initialized fully into the map after only one measurement using a single camera. To achieve that on the first tracking frame, the camera was held in a certain approximately known location relative to the target for tracking to start. The known features with their measured positions were placed into the map at system start-up with zero uncertainty which defined the world coordinate frame for SLAM. In the state vector the initial camera position was given an initial level of uncertainty corresponding to a few degrees and centimeters.

For natural landmark selection, the value of invariant features such as SIFT were appreciated in providing a high level of performance in matching, in loop-closing, or in localizing a lost robot. Though, SIFT features were found to be less suited to continuous tracking due to the high-computational cost of extracting them. Consequently, in order to increase the invariance of their features to the degree of freedom available to SIFT, each feature was stored as an oriented planar texture. Then, when making measurements of a feature from new camera positions, its patch could be projected from 3D to the image plane to produce a template for matching with the real image. This template was a warped version of the original square template captured when the feature was first detected.

They presented the use of MonoSLAM to provide real-time SLAM for a humanoid robot platform, HRP-2, as it moved around a cluttered indoor workspace. In the humanoid SLAM application, although it had been possible to progress with their vision-only algorithm, the ready availability of the gyro information with the humanoid robot played a role in reducing the rate of growth of uncertainty around looped motions. The gyro was sampled at the 30 Hz rate of vision for use within the SLAM filter, and the standard deviation of each element of the angular velocity measurement was assessed as  $0.01 \text{ rads}^{-1}$ . Then, the measurements were incorporated in the EKF directly as an internal measurement of the robot's own state which constituted an additional Kalman update step before visual processing.

For the experiments, the robot was programmed to walk in a circle of radius 0.75 m which was a fully exploratory motion, involving observation of new areas before

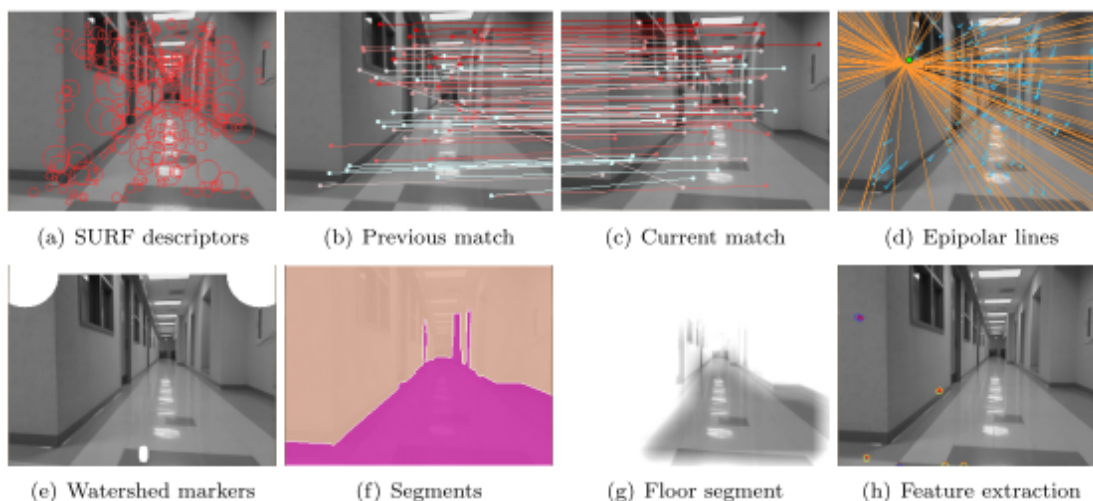
closing one large loop at the end of the motion. For safety and monitoring reasons, the motion was broken into five parts with short stationary pauses between them: First, a forward diagonal motion to the right without rotation in which the robot put itself in position to start the circle, and then four 90 degree arcing turns to the left where the robot followed a circular path, always walking tangentially. The walking was at HRP-2's standard speed, and the total walking time was around 30 seconds but the SLAM system continued to track continuously at 30 Hz even while the robot paused. Figure 2.13 shows the results of this experiment on which classic SLAM behavior can be demonstrated. A steady growth in the uncertainty of newly-mapped features existed until an early feature could be reobserved, the loop was closed, and the drift was corrected. A large number of features were seen to swing into better estimated positions simultaneously based on the correlations stored in the covariance matrix. The map of features was then suitable for long-term use, and it would be possible to complete any number of loops without drift in localization accuracy.



**Figure 2.13** (a) Early exploration and first turn (b) Mapping back all and greater uncertainty (c) Just before loop close, maximum uncertainty (d) End of circle with closed loop and drift corrected (Davison et al., 2007).

Yang et al. (2012) presented a monocular vision based SLAM algorithm with a particular focus on navigation of a micro aerial vehicle (MAV) operating in a range of indoor and outdoor environments. The proposed strategy exploited the so-called planar ground assumption, which held for many environments. The proposed methods

included segmentation of the ground plane in an environment, use of epipolar geometry for attitude estimation, and a variation of the FastSLAM algorithm in order to estimate the trajectory of an MAV while building a map by using a single camera and an altitude sensor. Figure 2.14 illustrates how the vision data was processed in order to accomplish all those tasks.



**Figure 2.14** Processing of vision data collected in indoor environments (Yang et al., 2012).

Specifically, three environments were chosen for testing: An indoor corridor environment, an outdoor environment with footpaths, and a river-like environment. Experiments demonstrated that the vision algorithm could effectively map a path in an open environment where there were not many distinguishable objects near the MAV. In such an environment, laser range finder based methods would not be able to map the path, since there is relatively no geometric difference between the path and the grass. For the experiments of a river-like environment, even when the ground around the creek was not strictly planar, the method could produce relatively accurate navigation and mapping results. Also being successful in a corridor environment, algorithm was demonstrated to be robust for the MAV's operation in different environments.

## **CHAPTER 3**

### **DESIGN AND IMPLEMENTATION**

#### **3.1 PROPOSED SYSTEM**

An intelligent agent is the one doing the right thing all the time, and an autonomous agent relies on its own percepts rather than the prior knowledge of its designer to exhibit intelligent behavior. Designing a visionary system which works according to artificial landmarks does not provide a system with autonomy since artificial landmarks themselves are purely source of the prior knowledge. Consequently, this system is designed to work with natural landmarks in an environment.

In order to define the intelligent agent of the designed system, the approach of Russel and Norvig (2003) is employed in the following subsection, which puts an emphasis on the performance measure, environment, actuators, and sensors of the robot.

##### **3.1.1 Agent Description**

The proposed system must cope with difficulties of real world environments as real mobile robots do. First of all, the real world environments are partially observable due to inaccurate or noisy measurements of sensors and missing parts of a state from sensory data. Secondly, there arises the problem of stochasticity from partial observability and unpredictable operation of agent's actuators. Thirdly, the real environments are sequential, which means that the next action of the agent depends on the previous percepts and actions. Fourthly, the agents need to adapt to dynamic environments. Fifthly, uncertain movements and steering angles of the robot in world reference frame brings about continuity problem to the environments, and finally, if the existence of other agents is taken into consideration, which is known as a multi-agent

environment, the problem reaches to an extremely difficult level. However, in this system, the environment is accepted as a semi-dynamic and single-agent environment to simplify the problem. The term semi-dynamic refers to the environments which do not change as the time elapses but in which the performance of the robot changes.

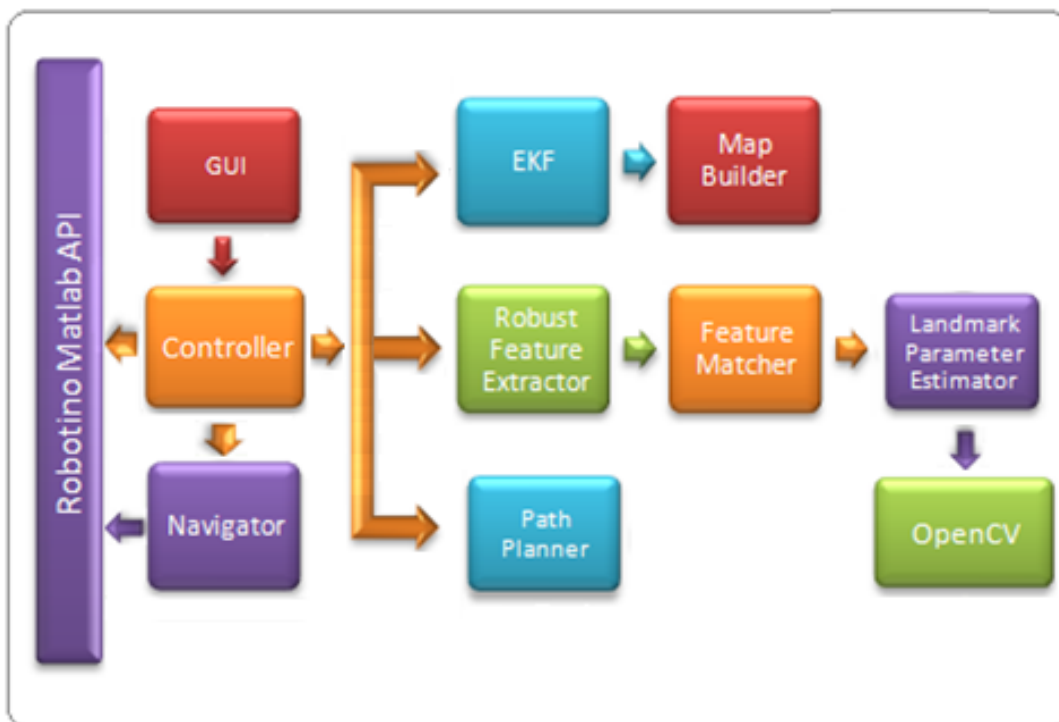
Then, given a target path for the system, the performance measure is how successfully the robot can localize itself and the natural landmarks in the environment, and hence how reliably the map of the environment is built. The primary sensors the mobile robot uses are a single camera to carry out the visionary tasks and an odometer, and the primary actuator is the omni-directional wheels to realize the navigation.

### **3.1.2 System Design**

In order to meet the requirements of the proposed system, a vision-based SLAM algorithm is implemented. The mobile robot is put into an environment full of natural landmarks and is given a goal to follow a path provided beforehand with constant or accelerating velocities. The aim of the algorithm is to localize the robot and to pinpoint the natural landmarks in the environment by tracking some robust features in the camera frames and simultaneously building a representative map of the environment. As the landmarks are re-observed, the localization and the mapping accuracies of the system are expected to increase.

Experimentation of the system is performed on an educational robot called Robotino (see Appendix A). The system is implemented as a MATLAB\C++ application by integrating some existent algorithms, Robotino MATLAB API, and OpenCV library. MATLAB has many built-in functions for convenient visualization and processing of data, and C++ programs enable the application to be faster and integration with OpenCV library.

Figure 3.1 shows the block diagram of system call sequence amongst the processing modules of the system that are briefly to be explained in the subsequent sections. Though, one should be aware that the figure does not indicate the dependence of the modules to each other but implies only which module directly calls another module.



**Figure 3.1** Block diagram of system call sequence.

### **3.1.2.1 Robust Feature Extractor**

This module extracts some features from single camera frames of the robot, which are known to be robust since they are invariant to scale and rotation, and are partially invariant to illumination and camera view point. Hence, these features are highly potential to be found repetitively among frames as the robot moves through the environment. Yet, robust feature extraction is a challenging process to carry out, so some available algorithms such as the SIFT or the SURF are adapted for this module.

### **3.1.2.2 Feature Matcher**

Robust features detected in the frames are matched in this stage. Thus, the recent locations of the features found in the previous frames are determined in the current frame. In order to run the following modules reliably and efficiently, the number of extracted and the matched features are controlled by arranging some related thresholds.

### ***3.1.2.3 Landmark Parameter Estimator***

This module is to estimate the 3D world coordinates of the landmarks in the environment, that are projections of 2D coordinates of the robust features matched among single camera frames. To achieve this, calibration of the camera is done only once to determine the intrinsic and distortion matrices of the camera. Later on, approximate 3D world parameters can be obtained for the landmarks by processing these matrices and the matched robust features through a number of algorithms.

### ***3.1.2.4 Extended Kalman Filter (EKF)***

Estimated robot and landmark positions are filtered in this stage by modeling the robot motion and landmark position uncertainties. Thus, the robot and the landmarks are approximately localized. The predicted robot position is estimated using the odometry of Robotino whereas the landmark positions in real world are measured as explained in the previous module.

### ***3.1.2.5 Map Builder***

Having the results of EKF stage that are the uncertain coordinates of the robot and the landmarks in the environment, this module simply builds a representative map of the environment. Via this map, the performance of the designed SLAM system can be demonstrated.

### ***3.1.2.6 Path Planner***

Unlike the path planning algorithms explained in Section 1.1.3, this unit does not plan a path for the robot to trace but just processes a path provided by the user through an interface. The path is input as a sequence of movements along the x or y axes, or rotation about the z-axis of Robotino. Hence, some rectangular trajectory can be presented to the system through which the robot can repetitively observe the landmarks in the environment.

### **3.1.2.7 Controller**

The controller of the system is a simple executive controller invoking proper modules as necessary to accomplish the localization of the robot and the mapping of the environment in the order of seconds. It can alternate between two sub-controllers one of which executing possibly the rectangular path input through the interface and the other generating the velocities for Robotino to follow a circular path. In order to grab frames and to retrieve the odometry from Robotino, and to transfer velocities designated to Robotino, Controller collaborates with the Robotino MATLAB API.

The Path Planner and the Controller modules are the ones potentially open for research in order for a deliberate control of the system. Then, global solutions to complex tasks can be generated using planning that may be realized in the order of minutes (Russell and Norvig, 2003). However, because a deliberate control depends on the outcomes of the SLAM task, current modules must be implemented reliably enough before its design.

### **3.1.2.8 Navigator**

This module is designed to enable the Robotino to move by prompting its actuators according to the path to be tracked. Though, since MATLAB does not allow threads, it could not be implemented as a stand alone module, and it is embedded into Controller.

### **3.1.2.9 Graphical User Interface (GUI)**

For the user to test the system easily and to observe the results of the working algorithms, this module is devised. Users can specify the path the robot will follow, control the system start up or velocities, observe the tracked landmarks in the environment, and visualize the map built up as the representation of the environment through the interface.

The algorithms, the key concepts, and some implementational details of the modules are analyzed by grouping these modules according to their relevance to each other through subsections 3.2 to 3.5. Then, some experimental results and evaluations



regarding the behavior of the robot and the performances of the modules are presented in Chapter 4.

## 3.2 ROBUST FEATURE EXTRACTION AND MATCHING

Robust Feature Extractor module employs SIFT or SURF algorithm depending on the choice of the user in order to extract features in the camera frames that are invariant to some transformations. Since extracting these features is only a step in this system, rather than trying to cope with all the details of the algorithms, the SIFT application developed by Vedaldi (2006) and the SURF implementation available in OpenCV are adapted. The representational outcomes of the algorithms are changed, and some of the parameters are modified in order to increase their reliability.

However, only the code of the SIFT algorithm is analyzed thoroughly to clarify how a robust feature extraction algorithm can be implemented. The subsections 3.2.1 through 3.2.4 describe the stages pertaining to the SIFT algorithm (Lowe, 2004; Sinha, 2010) and state if some modifications are made to these stages.

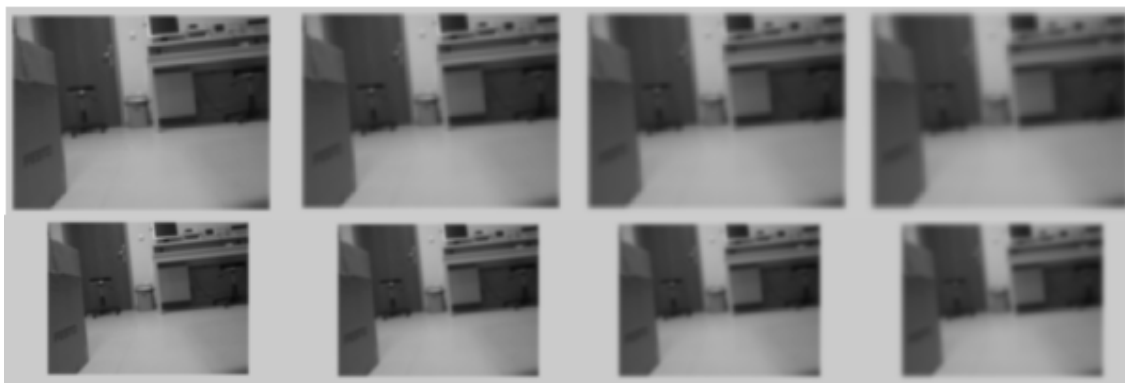
### 3.2.1 Scale Space Extrema Detection

In this initial step, a scale space of the image is generated by progressively blurring out the image according to the Gaussian function in Equation 3.1 in order to ensure scale invariance. Also the original image is resized to half several times and then the same blurring operation is performed on those resized images. Figure 3.2 depicts this procedure. Horizontal images of the same size are samples of an octave whereas each image stands for different scale in the octave. Hence, there are 2 octaves each with 4 scales in Figure 3.2.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (3.1)$$

The number of octaves and scale depends on the size of the original image. However, Lowe (2004) suggests that 4 octaves and 5 blur levels are ideal for the algorithm. If the original image is doubled in size and anti-aliased a bit by blurring it, then the algorithm produces four times more keypoints, which is better. The amount of

blurring in each image, or in other words, computation of the scale of an image is systematic. If the amount of blur in a particular image is denoted as  $\sigma$ , and the number of blurred images in each octave is denoted as  $n$ ; then the amount of blur in the next image becomes  $k*\sigma$  where  $k$  is  $(n-3)$  root of 2.



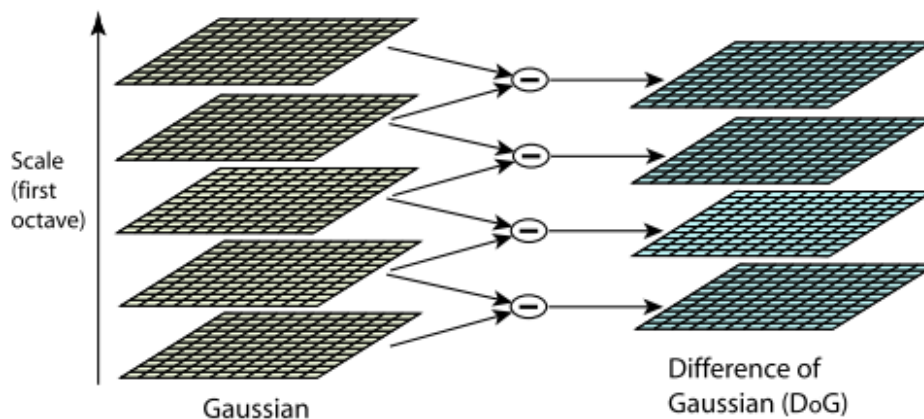
**Figure 3.2** Scale space of an image.

For example, the SIFT algorithm uses 5 octaves and 6 scales in order to compute the scale space of 320x240 sized images of Robotino. Table 3.1 gives the amounts of blur ( $\sigma$ ) used for each octave of the implementation where  $k$  is  $\sqrt[3]{2}$ .

**Table 3.1** Computed scales in an octave.

	SCALE				
OCTAVE	1.2263	1.5450	1.9466	2.4525	3.0900

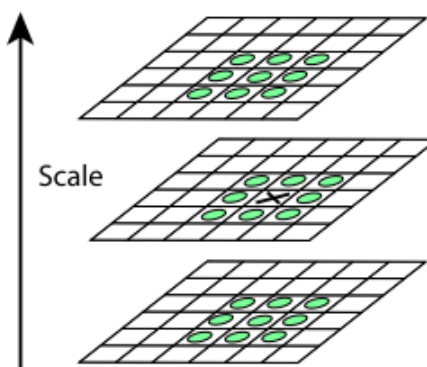
The Laplacian of Gaussian (LoG) is quite useful for finding interesting points, namely the keypoints, in an image. LoG locates the edges and corners in a blurred image by calculating the second order derivatives. Since the second order derivative is extremely sensitive to noise, blurring smoothes the noise and stabilizes the second order derivative. However, LoG is computationally expensive, so the SIFT algorithm replaces LoG computation with the Difference of Gaussians (DoG) that is the difference between two consecutive scales in the scale space representation as shown in Figure 3.3. The DoG images are approximately equivalent to the LoG, so a computationally intensive process is eliminated by the use of a simple subtraction.



**Figure 3.3** Computation of DoG images (Lowe, 2004).

The LoG images are not scale invariant because they depend on the amount of blur applied. If the term  $\sigma^2$  in the denominator of Gaussian expression is removed then true scale independence can be obtained. But the resultant images after the DoG operation are already multiplied by the  $\sigma^2$ . One side effect is that they are also multiplied by another number,  $(k-1)$ . Though, since just the locations of the maximums and minimums in the images are of concern to find keypoints, multiplying the image by some constant does not change the locations of maxima and minima.

As Figure 3.4 shows, a check is performed to locate maxima and minima in DoG images by comparing a pixel marked with X to its 26 neighbors in  $3 \times 3$  regions at the current and adjacent scales that are designated by shaded circles. So the lowermost and topmost scales are not searched to detect keypoints. A few initial checks are usually sufficient to discard a non-maxima or non-minima position.



**Figure 3.4** Maxima and minima of the DoG images (Lowe, 2004).

### 3.2.2 Keypoint Localization

Once pixels containing extreme values are found, they are the approximate maxima and minima. In this stage, the interpolated locations of the maxima or minima are determined to provide an improvement in matching and stability. To achieve that, a 3D quadratic function is fit around each approximate extremum by use of the Taylor series expansion in Equation 3.23., where  $D$  and its derivatives are evaluated at the keypoint and  $\mathbf{x} = (x, y, \sigma)^T$  is the offset from this point. Then, extreme points of this equation can be found by differentiating and equating it to zero. The result is the interpolated, or in other words, the sub-pixel locations of the keypoints.

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x} \quad (3.2)$$

During the process, extremums that do not have enough contrast and that are close to edges are removed in order to choose useful keypoints among all. For the former, if the magnitude of the intensity at the minima or maxima pixel in the DoG image is less than a certain value, then it is rejected. When refining the points, two gradients both perpendicular to each other at the keypoint are computed. The image around the keypoint can be a flat region where both gradients are small, can be an edge where only one gradient is small, or can be a corner where both gradients are large. The Hessian matrix already mentioned in Subsection 2.2 is employed in order to detect corners. The Harris corner detector normally computes two eigenvalues, but the ratio of these two eigenvalues is calculated to increase efficiency in the SIFT algorithm.

### 3.2.3 Orientation Assignment

At this step, the gradient magnitudes and directions around each keypoint are collected. The size of the region for orientation collection around the keypoint is equal to the size of the kernel used for Gaussian blurring whose scale is 1.5 times of the scale of the keypoint. Then, the most prominent one or more orientations in that region are assigned to the keypoint. This provides rotation invariance for the keypoints.

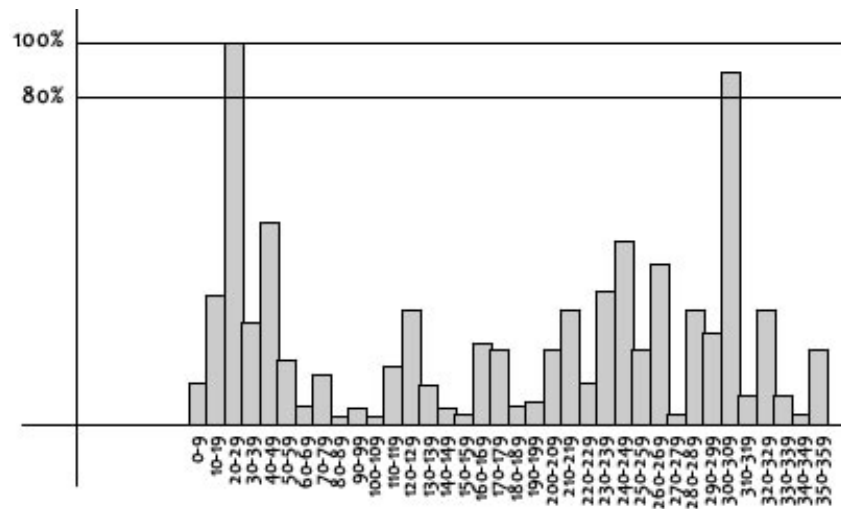
The magnitude and orientation are calculated for all pixels around the keypoint according to the Equations 3.3 and 3.4, respectively. Later, a histogram for which the 360 degrees of orientation is broken into 36 bins each storing orientations at the range

of 10 degrees is generated (see Figure 3.5). If the gradient direction at a certain point in the orientation collection region is 18.759 degrees, it goes into the 10-19 degree bin. The amount that is added to the bin is equal to the magnitude of gradient at that point blurred by a Gaussian-weighted circular window with a blur amount that is 1.5 times that of the scale of the keypoint.

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2} \quad (3.3)$$

$$\tan^{-1} \left( \frac{L(x, y + 1) - L(x, y - 1)}{L(x + 1, y) - L(x - 1, y)} \right) \quad (3.4)$$

When this computation is performed for all pixels around the keypoint, the histogram has a peak at some point. Also, any peak above 80% of the highest peak is converted into a new keypoint which has the same location and scale as the original keypoint, but a different orientation. Then, all peaks are assigned as the orientations to their associated keypoint.



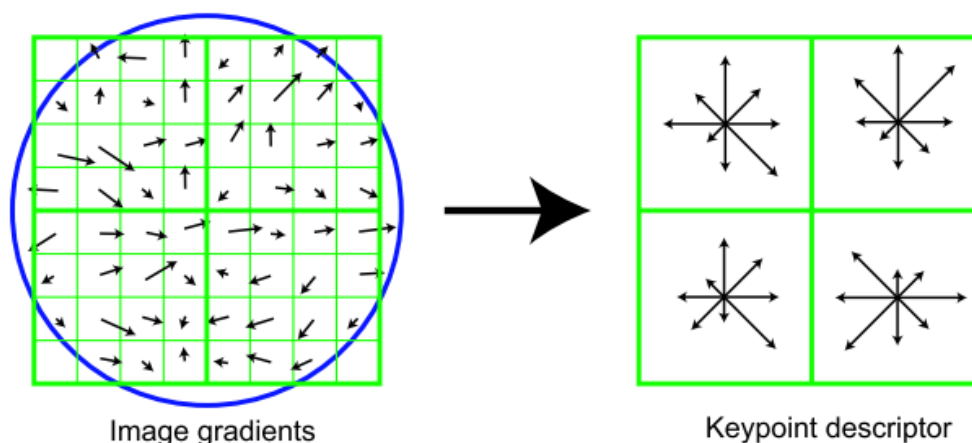
**Figure 3.5** Histogram of orientations to determine peaks (Sinha, 2010).

### 3.2.4 Keypoint Descriptor

In this final stage, local image gradients are measured at the selected scale in the region around each keypoint to distinguish each keypoint from others uniquely. These

are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

To compute the descriptor, first, the image gradient magnitudes and orientations are sampled around the keypoint location using the scale of the keypoint for selecting the level of Gaussian blur of the image. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. Figure 3.6 represents the procedure on an  $8 \times 8$  set of samples used to generate  $2 \times 2$  descriptor array, whereas in real applications  $16 \times 16$  window around each keypoint is split into sixteen  $4 \times 4$  descriptors. Within each  $2 \times 2$  descriptor, gradient magnitudes and orientations, that are shown as the length and direction of each arrow respectively in the right figure, are computed and these orientations are put into an 8 bin histogram. For example, any gradient orientation in the range 0-44 degrees adds to the first bin, 45-89 adds to the next bin and so on. The amount added to the bin depends on the magnitude of the gradient similar to 36 bin histogram computed for orientation designation, but it also depends on the distance from the keypoint. Gradients that are far away from the keypoint add smaller values to the histogram using Gaussian weighting function which is indicated by the circular window in the left figure as these are most affected by mis-registration errors.



**Figure 3.6** A  $2 \times 2$  descriptor array computed from an  $8 \times 8$  set of samples (Lowe, 2004).

It is important to avoid all boundary effects in which the descriptor suddenly changes as a sample shifts smoothly from being within one histogram to another or

from one orientation to another. Therefore, an interpolation is used to distribute the value of each gradient sample into adjacent histogram bins. In other words, each entry into a bin is multiplied by a weight of  $1-d$  for each dimension, where  $d$  is the distance of the sample from the central value of the bin as measured in units of the histogram bin spacing.

For all 16 descriptors,  $4 \times 4 \times 8 = 128$  numbers are obtained by assigning each 16 random pixel orientations into 8 predetermined bins. To remove the effects of illumination change normalizing the 128 numbers by root of sum of squares, a single keypoint can be uniquely identified by this feature vector. Additionally, the influence of large gradient magnitudes is removed by thresholding the values in the unit feature vector to be no larger than 0.2 each, and then by renormalizing the vector to unit length. The descriptors can be of any type, but integer descriptors are usually known to operate faster in matching stage.

### 3.2.5 Matching Keypoints

Keypoints computed by the SIFT or the SURF are matched to each other depending on their descriptors, because those descriptors are known as unique representations of each keypoint. A keypoint is matched to another if and only if their distance to each other is the least among all other comparisons.

The matching stage is the most promising one to improve the performance of the robust feature extraction in camera frames. Based on some confidence bound assumptions, most of the image processing can be avoided as well as alleviating the number of false matches. It means that only the vicinity of a keypoint can be matched to itself. Consequently, the matches found by these algorithms are eliminated in the feature matcher module if their distance to each other is greater than 15.81 pixels approximately. Figure 3.7 (a) and (b) show the SIFT and the SURF features extracted for the same 2 consecutive frames and illustrate each match of the keypoints within these frames with a different color after elimination of farther matches process.



(a)



(b)

**Figure 3.7** The keypoints extracted and matched in 2 consecutive frames with (a) the SIFT and (b) the SURF algorithms.

### 3.3 LANDMARK PARAMETER ESTIMATION

Three-dimensional (3D) world parameter estimation problem of landmarks can be solved via stereo vision using multiple cameras. However, it turns into a challenging problem when just a single camera exists to succeed it. Because, the translation and rotation between each camera pair can be exactly known in multiple camera systems, and the camera frames can be obtained simultaneously. In contrast, when applying stereo vision to a single camera frames, the exact rotation and translation between frames are not known beforehand, and there exists some time drift between the retrieval of frames.

Despite of those challenges, there are some studies in the literature which demonstrated the use of single camera for landmark parameter estimation. Consequently, the parameters or coordinates of the landmarks in the environment, that



are the 3D projections of the 2D locations belonging to the matched robust features in the camera frames, were considered to be estimated by employing the calibration and 3D projection tools of OpenCV library. In order to solve this problem, the steps explained in the following subsections should be executed in sequence.

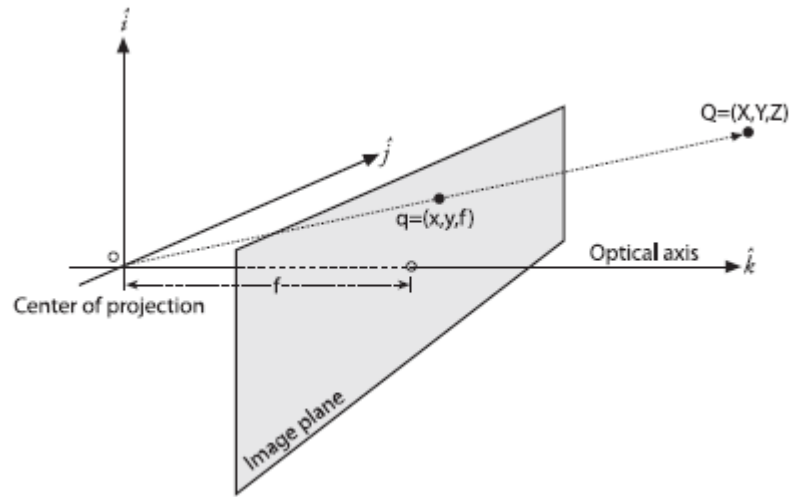
### 3.3.1 Camera Calibration

Camera calibration involves determination of some intrinsic parameters of a camera so that distortions in the camera images can be corrected, or camera measurements in the unit of pixels can be related to the real measurements with physical units in 3D world. The intrinsic parameters are defined by the model of the camera's geometry and the distortion model of the lens (Bradski and Kaehler, 2008a).

Figure 3.8 illustrates projection of points in 3D world onto image plane by modifying the pinhole camera model slightly, which aids the understanding of the model of the camera geometry. In the figure, the focal length,  $f$  is the distance from the center of projection (the pinhole aperture) to the image screen, and the principal point is where the optical axis intersects the image plane. A point  $Q(X, Y, Z)$  in world is projected onto image plane at  $q(x, y, f)$  by the ray passing through the center of projection. Then, using the similar triangles relationship, this projection can be defined by Equation 3.5, where  $c_x$  and  $c_y$  define the possible displacement of the center of the coordinates of the image screen away from the optical axis, and  $f_x$  and  $f_y$  are the product of the physical focal length  $f$  of the lens in mm and the sizes  $s_x$  and  $s_y$  of the imager elements in pixels per mm, respectively. The parameters  $c_x$  and  $c_y$  are resulted from imperfect alignment of the principal point and the center of the imager, and  $f_x$  and  $f_y$  are defined because each individual pixel is a rectangle rather than a square on low-cost imagers. By arranging those parameters into a 3x3 matrix  $M$ , and by determining the image points,  $q$ , as homogeneous coordinates after appending the scaling factor  $w$ , a practical projection of 3D points,  $Q$ , can be obtained as in Equation 3.6.

$$x = f_x \left( \frac{X}{Z} \right) + c_x, \quad y = f_y \left( \frac{Y}{Z} \right) + c_y, \quad (3.5)$$

$$q = MQ, \quad \text{where } q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (3.6)$$



**Figure 3.8** Point projection onto the image plane (Bradski and Kaehler, 2008a).

In addition to these, radial and tangential distortions of camera usually require the determination of 4 distortion parameters. Radial distortions arise as a result of shape of the lens when rays farther from the center of the lens are bent more than those closer. It can generally be characterized by the first two terms of Taylor series expansion around  $r = 0$  that are called  $k_1$  and  $k_2$ . Then the corrected  $x$  and  $y$  positions in the image are given as in Equation 3.7 and Equation 3.8. Tangential distortions arise from the assembly process of the camera as a whole since the lens could not be made exactly parallel to the imaging plane. Two additional parameters  $p_1$  and  $p_2$  can characterize the tangential distortions as indicated in Equation 3.9 and Equation 3.10.

$$x_c = x(1 + k_1 r^2 + k_2 r^4) \quad (3.7)$$

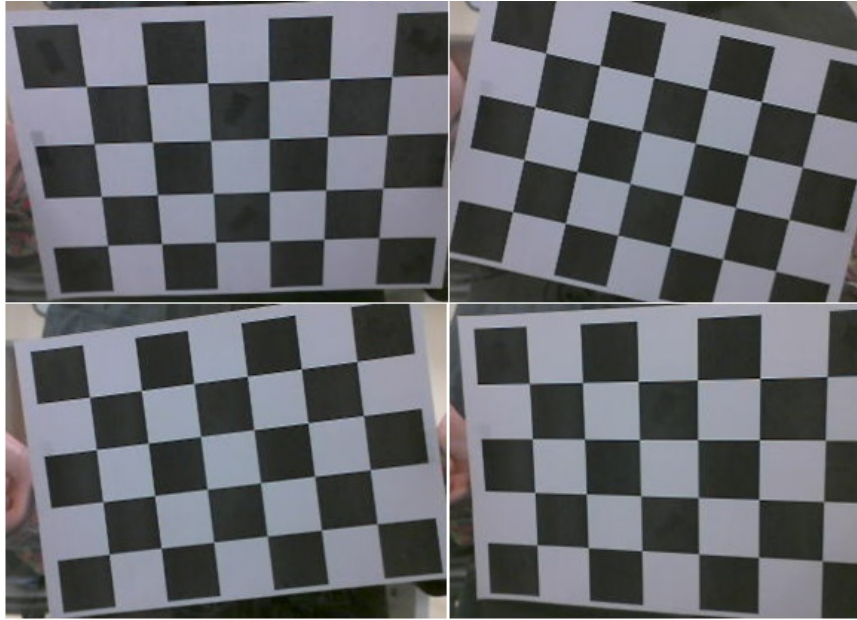
$$y_c = y(1 + k_1 r^2 + k_2 r^4) \quad (3.8)$$

$$x_c = x + [2p_1 y + p_2(r^2 + 2x^2)] \quad (3.9)$$

$$y_c = y + [p_1(r^2 + 2y^2) + 2p_2 x] \quad (3.10)$$

OpenCV calibration routine can be used in order to determine the camera matrix  $M$  and the distortion parameters. To run this routine the camera is targeted on a known structure like a chessboard that has many identifiable points. The chessboard whose relative distance to camera is 35 cm is rotated and translated across the camera without changing the distance as much as possible. Hence, multiple views are provided as

shown in Figure 3.9. These views are later used to compute the relative location and orientation of the camera at the time of each view that are known as extrinsics of the camera as well as to compute the intrinsic parameters of the camera.

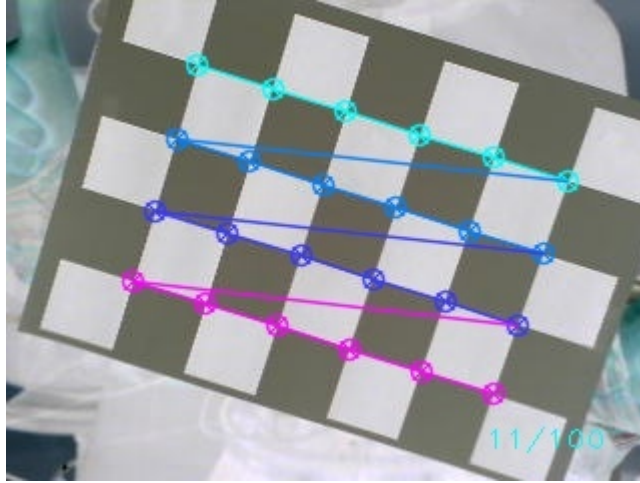


**Figure 3.9** Different views of calibration object.

To determine the coordinates of chessboard corners at imager plane, first of all the chessboard images are filtered using the adaptive threshold that is set on a pixel-by-pixel basis by computing the weighted average of the  $b$ -by- $b$  region around each pixel minus a constant. According to the structure of interior corners on the chessboard, the corner points are determined. To calibrate the camera of Robotino, a chessboard having 6x4 interior corners is used. Then to increase the location accuracy of the corners, their subpixel locations are estimated. Figure 3.10 is one view of calibration object on which the corner locations at subpixel accuracy are drawn. Points in each row are in different color, and they are connected by lines representing the identified corner order.

Later, the 3D physical coordinates which belong to the interior corners of the chessboard are located in the coordinate frame attached to the object. For the sake of simplicity,  $z$  coordinate of those points are given 0 since the calibration object is planar, and their  $x$  and  $y$  coordinates are given as integers starting from (0, 0) up to (0, 5) for the first row, and starting from (3, 0) to (3, 5) for the last row of the chessboard.

However, this means that the camera world, object and camera coordinate units are in mm/39, because the chessboard has squares of 39 mm at each side.



**Figure 3.10** Corner locations on a chessboard view.

Thus, to shift from a coordinate system centered on an object to one centered at camera, the object coordinate system is translated by the physical translation  $T$  and is rotated by the physical rotation  $R$ . For each view of the planar calibration object, a homography matrix  $H$  that can relate points ( $Q$ ) on the source image plane to the points ( $q$ ) on the imager is computed as in Equation 3.11. The matrix  $H$  contains both the transformations  $R$  and  $T$ , and the camera matrix  $M$ . Consequently, multiple homographies from multiple views in which the chessboard movement is distinguishable enough can be used to solve for the camera matrix robustly.

$$q = HQ, \text{ where } H = M[R T] \quad (3.11)$$

OpenCV calibration algorithm solves for 6 extrinsic parameters belonging to chessboard transformations relative to camera for each view together with the intrinsics of the camera. So, if  $N$  is the number of views, in total  $6N+8$  parameters are solved. Since trying to solve for all parameters at once may lead to inaccurate or divergent results, those parameters are solved by holding some of them fixed and solving for the others and later by holding the other parameters fixed and solving for the original a number of times in a sequence. For example, the camera matrix is initially solved by assuming that there is no distortion in the image points obtained. Later, the distortion

parameters are solved by incorporating the distortion equations into process, and the camera matrix and the extrinsics are re-estimated accordingly. For the calibration process, the parameter estimator module does not provide any initial guess on the parameter space, though it is possible to ease this computation.

In order to call the calibration routine of OpenCV from MATLAB for the camera frames of Robotino, first of all the camera frames retrieved from Robotino are stored as .mat file. Secondly, some RGB views of the calibration object that differ from each other distinguishably are gathered in a 4D data structure. Thirdly, a wrapper that can convert 4D image sequence of MATLAB to an array of `IplImage` that is the image storage object of OpenCV is written. Then, the calibration algorithm is modified so that it can handle this new array of images properly. Finally, the mex version of the C++ calibration algorithm is generated by compiling the files to a format callable by MATLAB after MATLAB R2011b is integrated to the compiler of Visual Studio 2010, and proper links are given in `mexopts.bat` file of MATLAB to OpenCV libraries. In addition to the intrinsics and the extrinsics of the camera, the output of the algorithm is how successfully the 3D points of chessboard can be reprojected to image points using the intrinsics and the extrinsics computed.

### **3.3.2 Projection to 3D Coordinates**

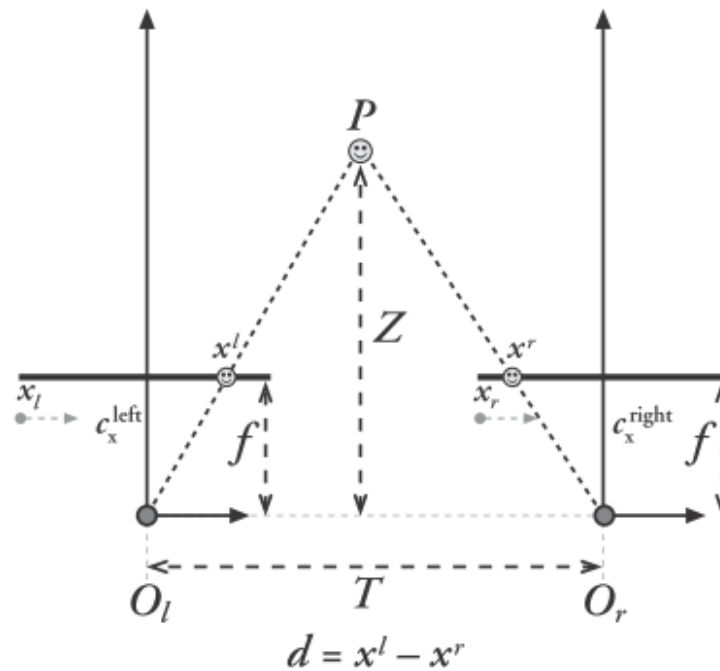
The camera matrix allows us to transform from 3D coordinates to 2D coordinates of the image. However, a line in 3D world corresponds to a point in the image for the reverse operation. So, there is no reliable way to extract 3D information without using multiple images.

One way to extract 3D information is stereo vision in which features in two or more images taken at the same time from separate cameras are matched with the corresponding features in the other images, and the pixel differences of the matched features are analyzed to extract depth information. Another way is SFM in which multiple images are taken at different times and different places by only a single camera, and features in those images are matched to compute a matrix called the fundamental matrix in order to relate two different views together and thus to understand the scene.

If the case of having two cameras that are horizontally mounted is analyzed to have an intuition on the problem, stereo imaging involves four steps: Undistortion, rectification, correspondence, and triangulation. Undistortion is mathematically removing the radial and tangential lens distortions in the images. Rectification means making two images coplanar, in other words row-aligned, according to the rotation and translation between the cameras. Correspondence is finding the same features in the left and the right images so that disparities in the x-coordinates of the features can be computed. Infact this step is already achieved by the SIFT or the SURF algorithms for a subset of pixels. Finally, triangulation enables the extraction of depth to the corresponding features based on the geometric arrangement between the cameras. Figure 3.11 depicts how depth of a point in real world coordinates can be extracted using the similar triangles relationship. In the figure, it is assumed that the left and the right images are row-aligned so that every pixel row of the left camera aligns exacty with the corresponding row in the right camera. If a point P in physical world can be found in the left and the right image planes in coordinates  $x^l$  and  $x^r$ , respectively, the depth of this point Z can be found according to the Equation 3.12. This formula proves that the depth Z is inversely proportional to the disparity d between the coordinates  $x^l$  and  $x^r$ .

$$Z = \frac{fT}{d} \quad (3.12)$$

For the case of having a single camera that is mounted at a mobile robot, in order to satisfy such a configuration using just the frames of the camera, first of all the environment must be static. While the robot moves from one place to another the coordinates of the points in real world should not change. Second, the robot should navigate mostly tangentially rather than perpendicularly to the environment so that some amount of disparity can be measured for the matched points to estimate their depths (Davison et al., 2007). Though, this is not a very natural way to move the robot in the environment. Consequently, this problem is considered as out of scope of this thesis.



**Figure 3.11** Triangulation process for row-aligned stereo images (Bradski and Kaehler, 2008a).

### 3.4 EKF AND MAP BUILDING

When landmark parameter estimation process is accomplished, the SLAM process can be considered as three steps: Updating the current state estimate using the odometry data, updating the estimated state by re-observing landmarks, and adding new landmarks to the current state. Some details on EKF matrices and all of these steps managed by the EKF system are explained according to (Riisgaard and Blas, 2005; Newman, 2003) through the subsections 3.4.1 and 3.4.2.

#### 3.4.1 EKF Matrices

Even though there are different notions for the same matrices of EKF in various papers, the notions that are most commonly used are explained in the following paragraphs.

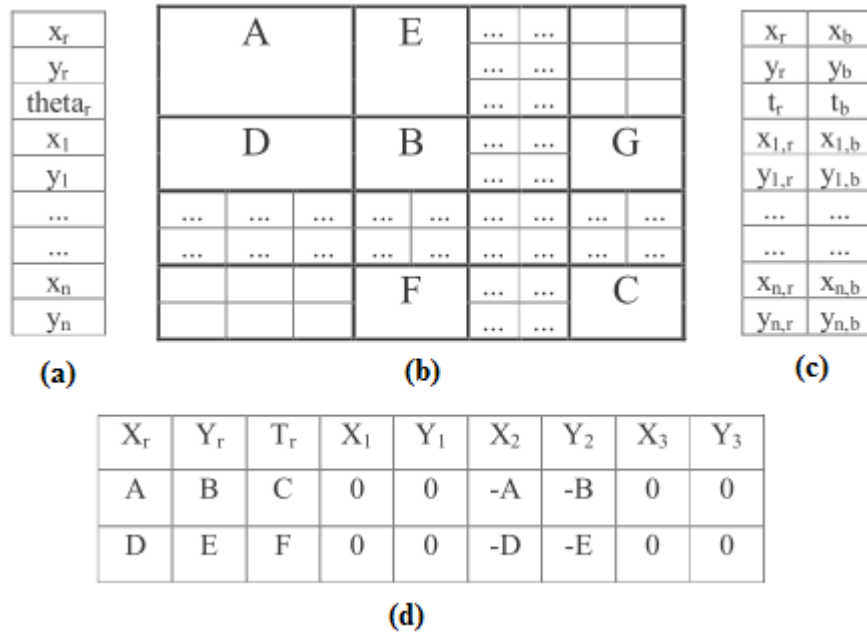
The system state matrix,  $X$ , in Figure 3.12 (a) contains the  $x$  and  $y$  positions, and the rotation  $\theta$  of the robot. It also contains the  $x$  and  $y$  positions of each landmark. In

order for all equations to work properly it has to be a vertical matrix whose size is 1 column wide and  $3+2*n$  rows high, where  $n$  is the number of landmarks.

The covariance matrix,  $P$ , contains the covariance on the robot position, the covariance on each landmark, the covariance between the robot position and the landmarks, and finally, the covariance between the landmarks. The covariance of two variables is a measure of how strongly correlated they are. Figure 3.12 (b) shows the content of the covariance matrix  $P$ . The covariance matrix is built up very systematically. The first cell,  $A$ , contains the covariance on the robot position. It is a  $3 \times 3$  matrix for  $x$ ,  $y$  and  $\theta$  of the robot.  $B$  and  $C$  matrices of  $2 \times 2$  for  $x$  and  $y$  positions of the landmark are the covariance on the first landmark and covariance on the last landmark, respectively. The cell  $D$  contains the covariance between the first landmark and the robot state. On the other hand, the cell  $E$  contains the covariance between the robot state and the first landmark. The matrix  $E$  can be deduced by transposing the sub-matrix  $D$ . Similarly,  $F$  contains the covariance between the last landmark and the first landmark, while  $G$  contains the covariance between the first landmark and the last landmark. Initially, as the robot has not seen any landmarks, the covariance matrix  $P$  only includes the matrix  $A$ . In order to incorporate the uncertainty in the initial position,  $P$  is formed using proper default values for the diagonal, even though the initial robot position is exact. Otherwise, errors may occur in some of the calculations through the process.

The Kalman gain,  $K$ , is computed to determine how reliable the observed landmarks are, and how much of the new knowledge they provide should be reflected onto the process. For example, if the robot should be moved 10 cm to the right according to the landmarks, the resulting movement computed through Kalman gain may only be 5 cm. Because, when the landmark measurement is more unreliable compared to the odometry of the robot, its Kalman gain is given low, and thus a compromise between the odometry and the landmark correction is found. The matrix  $K$  is illustrated in Figure 3.12 (c). The first row shows how much the  $x$  position of the robot should gain from the difference between the expected and the actual observations regarding the landmarks. The remaining rows similarly indicate the gains for the other robot parameters and each landmark positions. The first and second columns describe the gains in terms of the range and the bearing respectively.





**Figure 3.12** Matrices (a)  $X$  (b)  $P$  (c)  $K$ , and (d)  $J_H$  used in EKF (Riisgaard and Blas, 2005).

To compute an expected range and bearing of the measured landmark positions, the measurement model denoted as  $h$  in Equation 3.13 is used, where  $\lambda_x$  and  $\lambda_y$  are the  $x$  and  $y$  positions of the landmark, respectively.

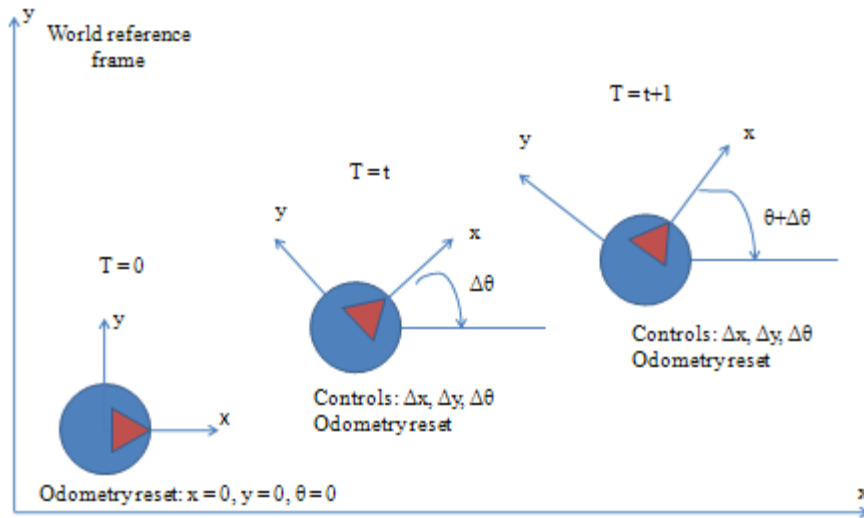
$$h = \begin{bmatrix} \text{range} \\ \text{bearing} \end{bmatrix} = \begin{bmatrix} \sqrt{(\lambda_x - x)^2 + (\lambda_y - y)^2} + v_r \\ \tan^{-1} \left( \frac{\lambda_y - y}{\lambda_x - x} \right) - \theta + v_\theta \end{bmatrix} \quad (3.13)$$

The Jacobian of the measurement model,  $J_H$ , indicates how much the range and bearing to the landmarks change as  $x$ ,  $y$  and  $\theta$  of robot position changes. The contents of the usual  $J_H$  for regular EKF state estimation is given by Equation 3.14. The first row is the change in range, whereas the second row is the change in bearing depending on robot parameters  $x$ ,  $y$  and  $\theta$ . For instance the third column in the first row is zero since the range does not change as the robot rotates. During SLAM, some additional values for the landmarks are needed in this matrix. Figure 3.12 (d) shows the matrix  $J_H$  for the second landmark whose first 3 columns are the regular  $J_H$  and each added two columns belong to the landmarks. The  $X_2$  column of the second landmark is set to  $-A$  and  $-D$ , the  $Y_2$  column is set to  $-B$  and  $-E$  and the columns for the rest of the landmarks are set to 0.

$$J_H = \begin{bmatrix} \frac{x-\lambda_x}{r} & \frac{y-\lambda_y}{r} & 0 \\ \frac{\lambda_y-y}{r^2} & \frac{x-\lambda_x}{r^2} & -1 \end{bmatrix} \quad (3.14)$$

The prediction model defines how to compute an expected position of the robot based on the old position and the control input as in Equation 3.15, where  $\Delta x$ ,  $\Delta y$ , and  $\Delta\theta$  are the controls obtained from the drivers of Robotino via `Odometry_construct` in MATLAB API and  $q$  is the error term. Figure 3.13 illustrates how these controls are generated in the reference frame of the robot. Then the Jacobian of the prediction model,  $J_F$ , used for robot position prediction can be defined as in Equation 3.16.

$$F = \begin{bmatrix} x + \Delta x \cos \theta - \Delta y \sin \theta + q \\ y + \Delta x \sin \theta + \Delta y \cos \theta + q \\ \theta + \Delta \theta + q \end{bmatrix} \quad (3.15)$$



**Figure 3.13** Odometry controls generated in the reference frame of robot.

$$J_F = \begin{bmatrix} 1 & 0 & -\Delta x \sin \theta - \Delta y \cos \theta \\ 0 & 1 & \Delta x \cos \theta - \Delta y \sin \theta \\ 0 & 0 & 1 \end{bmatrix} \quad (3.16)$$

The prediction of model of landmark positions when a new feature is being added to the state matrix  $X$  is given by the Equation 3.17 where  $r$  and  $b$  are the measured range and bearing of the landmark positions. The SLAM specific Jacobians,  $J_{x_r}$  and  $J_z$ , are also for integration of new features and computed based on the Equation 3.17. The Jacobian of Equation 3.17 with respect to the robot position  $(x, y, \theta)$ ,  $J_{x_r}$ , is given in

Equation 3.18. The Jacobian of the Equation 3.17 with respect to the range and the bearing,  $J_z$ , is as in Equation 3.19.

$$\begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} = \begin{bmatrix} x + r \cos(\theta + b) \\ y + r \sin(\theta + b) \end{bmatrix} \quad (3.17)$$

$$J_{xr} = \begin{bmatrix} 1 & 0 & -r \sin(\theta + b) \\ 0 & 1 & r \cos(\theta + b) \end{bmatrix} \quad (3.18)$$

$$J_z = \begin{bmatrix} \cos(\theta + b) & -r \sin(\theta + b) \\ \sin(\theta + b) & r \cos(\theta + b) \end{bmatrix} \quad (3.19)$$

The process noise,  $Q$ , is a Gaussian noise proportional to the controls,  $\Delta x$ ,  $\Delta y$ , and  $\Delta \theta$ . If  $W$  is a vector storing the controls, then  $Q$  matrix of  $3 \times 3$  can be computed as in Equation 3.20 by multiplying some Gaussian sample  $C$  with  $W$  and  $W^T$ .  $C$  is a representation of how exact the robot odometry is, which can be set according to the results of experiments on the robot odometry performance. However, the matrix of  $Q$  is not defined as being proportional to the controls in EKF module, since no direct relationship is observed between the controls and the odometry uncertainties (see Subsection 3.4.2.1). The matrix  $Q$  adopted in the EKF process of the system is defined as in Equation 3.21.

$$Q = W \cdot C \cdot W^T \quad (3.20)$$

$$Q = \begin{bmatrix} 2(0.01)^2 & 0 & 0 \\ 0 & 2(0.01)^2 & 0 \\ 0 & 0 & 2\left(\frac{1.5\pi}{180}\right)^2 \end{bmatrix} \quad (3.21)$$

The measurement noise matrix,  $R$ , is a Gaussian noise proportional to the range and the bearing for the range measurement device. The exact measurement noise is calculated by  $VRV^T$ , where  $V$  is a 2 by 2 identity matrix and  $R$  is a 2 by 2 matrix with the range,  $r$ , and the bearing,  $b$ , multiplied by some constants  $c$  and  $d$ , respectively, in the diagonal. The constants represent the accuracy of the measurement device. For example, if the range error has 1 cm variance,  $c$  should be a Gaussian with variance 0.01, or if the bearing error is always 1 degree,  $bd$  should be replaced with 1. The bearing error should not be proportional to the size of the angle since this does not make sense.

### 3.4.2 EKF Steps

The first step in EKF is the addition of the robot controls to the old state estimate. For example, if the robot is at point  $(x, y)$  with rotation  $\theta$ , and the translational and rotational controls are  $(dx, dy, d\theta)$ , then the result of the first step is the new state of the robot  $(x+dx, y+dy, \theta+d\theta)$ . In the second step, using the estimate of the current robot position, it is possible to predict where the observed landmarks should be. Based on what the robot measures, the difference between the estimated robot position and the actual robot position is called the innovation. The uncertainty of each observed landmark is updated according to the innovation. At this point, if the uncertainty of the current landmark position is very low, re-observing a landmark from current position with low uncertainty increases the landmark certainty. In the third step, new landmarks are added to the state, in other words to the robot map of the world. This is achieved by incorporating the information on the current position of robot and the correlations of new and old landmarks. Thus, the SLAM process is realized by repeating these three steps consecutively.

#### 3.4.2.1 State Update of Odometry

In this prediction step, the current state is updated using the odometry data as in Equation 3.15. Thus, the first three spaces of the matrices  $X$  and the Jacobian of prediction model  $J_F$  are updated accordingly. Finally, the covariance for robot position reserved at the upper left  $3 \times 3$  segment of the covariance matrix  $P$  ( $P^{rr}$ ) and the robot to feature cross correlations ( $P^{ri}$ ) are refreshed according to the Equation 3.22 and Equation 3.23.

$$P^{rr} = J_F \cdot P^{rr} \cdot J_F^T + Q \quad (3.22)$$

$$P^{ri} = J_F \cdot P^{ri} \quad (3.23)$$

#### 3.4.2.2 State Update of Re-observed Landmarks

The estimate for the robot position is not completely exact due to the odometry errors of the robot. To compensate for these errors, the displacement of the robot can be calculated using the associated landmarks compared to what the predicted robot position is. Later, the robot position is updated accordingly.

This step is run only for each re-observed landmark but not for the new landmarks. Hence, the computation cost is reduced. Landmark range and bearing denoted as  $h$ , and the Jacobian  $J_H$  for the current landmark are computed according to the Equation 3.13 and Equation 3.14 respectively, as mentioned above. Then, the computed range and bearing can be compared to the range and bearing for the landmark obtained from the data association, which is denoted as  $z$ . Additionally, the measurement noise matrix  $R$  is updated to reflect the range and bearing in the current measurements and the Kalman gain is computed using the following formula in Equation 3.24.

$$K = P \cdot J_H^T (J_H \cdot P \cdot J_H^T + V \cdot R \cdot V^T)^{-1} \quad (3.24)$$

Then, the Kalman indicates how much each of the landmark positions and the robot position should be updated according to the re-observed landmark. The term  $(HPH^T + VRV^T)$  is called the innovation covariance,  $S$ , that can be also used in the data association step when deciding the validity of the landmarks. Finally, the new state vector  $X$  and the covariance matrix  $P$  are computed using the Kalman gain as in Equation 3.25 and Equation 3.26. In consequence of repetition of this process for each matched landmark, the robot position along with all the landmark positions is updated.

$$X = X + K (z - h) \quad (3.25)$$

$$P = P - KSK^T \quad (3.26)$$

### 3.4.2.3 Adding New Landmarks

This final step updates the state vector  $X$  and the covariance matrix  $P$  with new landmarks that can be matched through the process. A new landmark is added to the state vector  $X$  as in Equation 3.27.

$$X = [X \ x_N \ y_N]^T \quad (3.27)$$

In addition to this, two new rows and two new columns to the covariance matrix  $P$  are added for the new landmark which is represented with the shaded area in Figure 3.14. Firstly, the covariance for the new landmark is added in the cell C according to Equation 3.28, where  $P^{N+1N+1}$  stands for the covariance of the  $N+1^{\text{th}}$  landmark.

Secondly, the robot-landmark covariance for the new landmark is added according to the Equation 3.29, which corresponds to the upper right corner of the covariance matrix. The landmark-robot covariance is the transposed value of the robot-landmark covariance corresponding to the lower left corner of P. Finally, the landmark-landmark covariance is added to the remaining section of the lowest row depending on Equation 3.30, whose transpose is similarly registered to the remaining of the rightmost column.

$$P^{N+1 \times N+1} = J_{xr} \cdot P \cdot J_{xr}^T + J_z \cdot R \cdot J_z^T \quad (3.28)$$

$$P^{r \times N+1} = P^{rr} \cdot J_{xr}^T \quad (3.29)$$

$$P^{N+1 \times i} = J_{xr}(P^{ri})^T \quad (3.30)$$

A			E		...	...		
					...	...		
					...	...		
D			B		...	...	G	
					...	...		
...	...	...	...	...	...	...	...	
...	...	...	...	...	...	...	...	
			F		...	...	C	
					...	...		

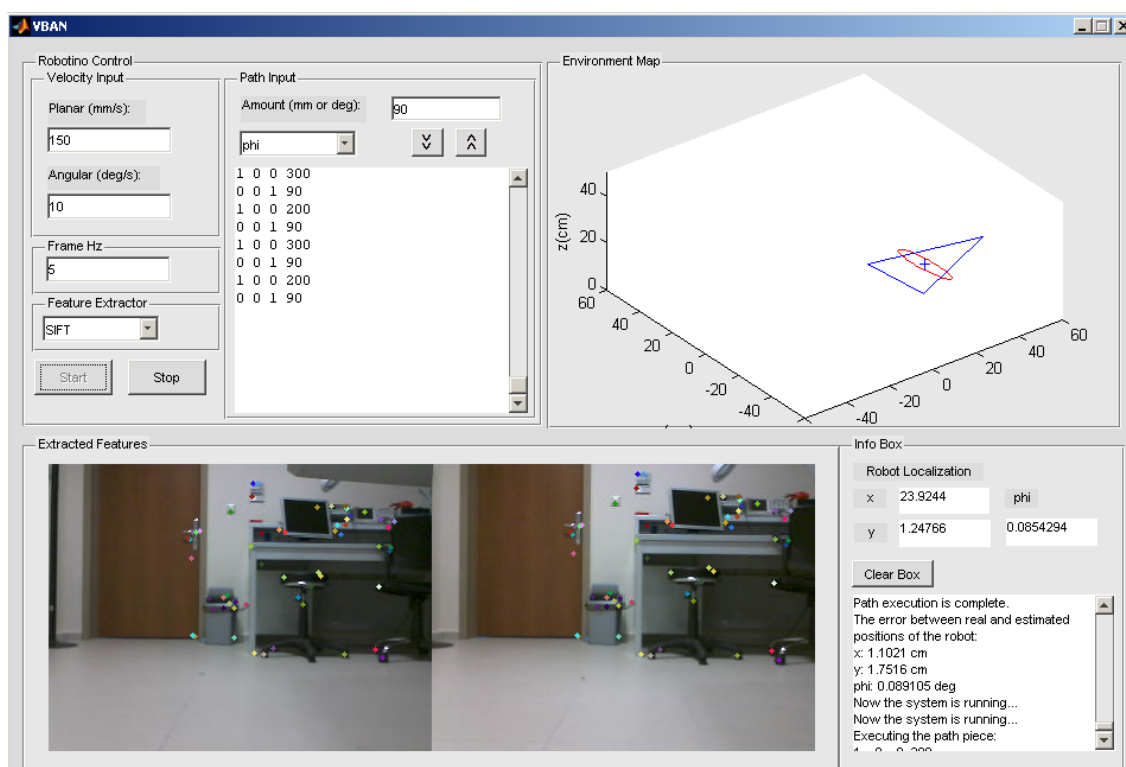
**Figure 3.14** Rows and columns appended to matrix P for a new landmark (Riisgaard and Blas, 2005).

### 3.5 MATLAB INTERFACE

Demonstration of the system is achieved through an interface designed in MATLAB. A screenshot of the interface so-called VBAN is shown in Figure 3.15. VBAN is mainly composed of four sections: Robotino Control, Environment Map, Extracted Features, and Info Box.

The section of Robotino Control is to arrange the velocities and the path to be used during navigation. Also, at what frequency frames are to be grabbed and which type of algorithm is to be applied on these frames can be determined through this section. Two buttons, Start and Stop, are to initiate and to terminate the navigation of the robot, respectively. The Extracted Features section displays the robust features

detected and matched in the two recent camera frames of the Robotino. Environment Map represents the estimated robot location in the environment and the uncertainty in localization of the robot during navigation. Whenever the navigation terminates, the real and the estimated trajectories of the robot are drawn in this map. Finally, the section of Info Box informs the user about the localization of the robot, the system functioning, the path execution of the robot, or the error between the real and the estimated positions of the robot at the final localization.



**Figure 3.15** Matlab interface of the system so-called VBAN.

### 3.6 SYSTEM CONTROL AND NAVIGATION

The Controller module of the system is initiated immediately after the user presses the Start button of the interface. If a circular path is chosen for Robotino to follow, then controller enabling circular movement of the robot is called. Otherwise, controller executing the path fragments provided through the interface is employed.

During navigation of the robot, planar and angular velocities are kept constant for simplicity when path fragments input are being executed. Yet, varying velocities are set

along the x and y axes of the robot when a circular path is being tracked. If the Robotino is to be navigated by some amount along an axis, the odometry of the robot is used to check whether the robot has moved by the desired amount and to terminate the movement.

As the proper velocities for related path are generated, Controller module tries to grab frames at the frame frequency specified by the user. At the same time the frames were grabbed, the odometry of the Robotino is tried to be sampled. Because, the measurements obtained from the camera and the odometry data should be synchronized as much as possible in order to be related in the EKF system. Then, again according to the choice of the user the SIFT or the SURF features are extracted from the camera frames, and they are related to the previous features extracted. Subsequently, the EKF system is run depending on the odometry and the landmark parameter measurements that can be estimated based on the explanations in Subsection 3.3.2. During the EKF routine, the localization results of the robot are displayed on the interface.

The computer code of the Controller module managing the other modules through function calls in order both to run the Robotino and to realize the system is given in Appendix C. For circular paths, the `controlCircular` function, and for user specified paths, `controlRectangular` function is called, respectively.



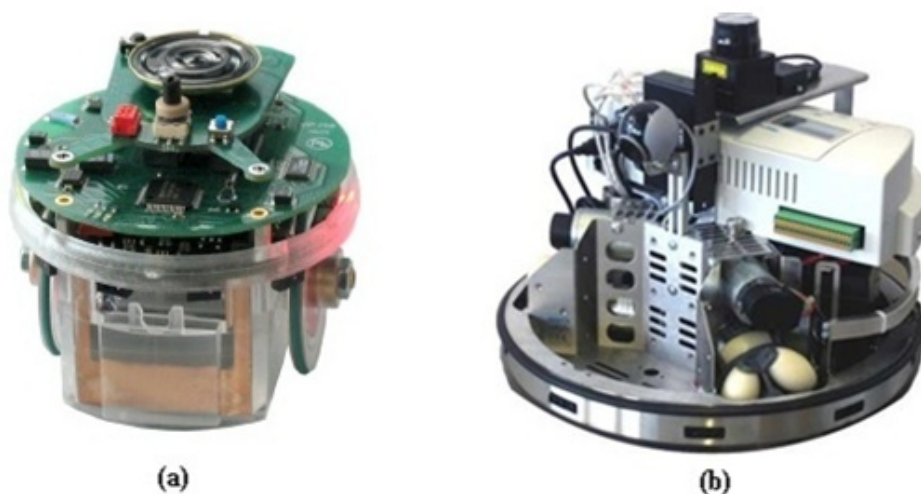
## CHAPTER 4

### EXPERIMENTS AND RESULTS

#### 4.1 EXPERIMENTAL PLATFORMS

The designed system is considered to be run on a robotic platform to measure its performance in a real world environment. Two educational robots available to implement the system were E-puck by GCTronics and Robotino by Festo which are shown in Figure 1.1 (a) and (b). Appendix B and Appendix A give some information on the hardware specification of E-puck and Robotino, respectively.

Subsection 4.1.1 explains why E-puck could not be employed for this system along with emphasizing some constraining details of its camera module, and Subsection 4.1.2 clarifies how the system is adapted to Robotino and presents the results of various experiments performed on Robotino's actuators and sensors.



**Figure 4.1** Educational robots (a) E-puck (b) Robotino.

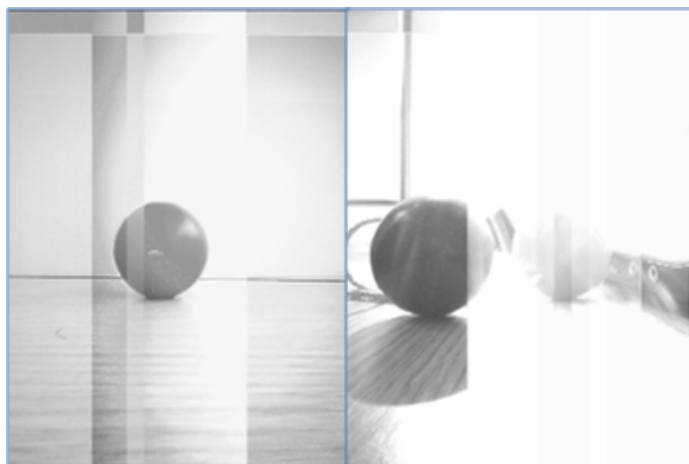
### 4.1.1 E-Puck

The version of E-puck available had a 16-bit digital signal controller (DSC) running at 30 Hz, 8kB of RAM, and 144 kB flash in order to store programs. Downloading the E-puck libraries of bluetooth, ADC, camera, codec, motor\_led, UART, FFT, I2C from its svn and integrating them to the project properly, E-puck could be programmed in C language using MPLAB integrated development environment. MPLAB compiled those C programs and generated files with the extension .hex which could directly be transferred to E-puck by a robot simulator program Webot via bluetooth.

Evidently, the capacity of E-puck was insufficient to realize the designed system. Due to similar problems, transfer of images received from its camera module to a computer was also too slow to implement the main system on the computer.

#### 4.1.1.1 Camera Module

Having a 640x480 resolution for height and width of the images and 2 bytes of storage for each pixel in RGB mode, E-puck even can not store one single image of its camera. If the only task the robot performs is to grab camera frames, for instance, in 40x40 sub-sampled RGB and gray scale modes, it can receive 4 fps and 8 fps, respectively. Figure 4.2 demonstrates the quality of the images that could be retrieved from the camera of E-puck. It is clear that the images are distorted too much to be reliably used in a vision based application.



**Figure 4.2** 320x240 images retrieved from E-puck camera module.

### 4.1.2 Robotino

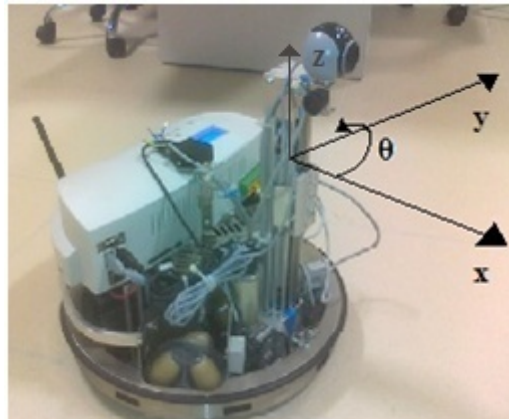
As a more enhanced educational platform, Robotino enabled realization of the system designed. The Robotino has an embedded PC working at 500 MHz. The PC runs Linux operating system, and it works in coordination with a microcontroller that is responsible of sensor readings and actuator operations.

Though, again the Robotino itself is not sufficient to develop high-level applications other than the ones requiring basic control of the robot. However, via W-LAN provided by the Robotino and through the services it provides, it is possible to transfer images or sensor values of the Robotino to an external computer, and to control the robot from this computer in an advanced manner.

The system ran on a 3 GHz desktop computer and communicated with the Robotino depending on Robotino MATLAB API via a 300 Mbps W-LAN adapter. The experimental setup mainly involved the navigation of the Robotino along a looping path which may be determined according to the choice of the users to enable the robot to return to its initial position approximately and to re-observe the landmarks in the environment. The degree of freedom (DoF) available to the Robotino via its omnidirectional wheels is shown in Figure 4.3. For a sample rectangular path the Robotino may follow, the order of controls with respect to the axes defined is given in Table 4.1.

**Table 4.1** Controls generating a sample rectangular path for Robotino.

<b>Movement</b>	<b>Amount</b>
Along x	2010 mm
About z	90°
Along x	1210 mm
About z	90°
Along x	2010 mm
About z	90°
Along x	1210 mm
About z	90°



**Figure 4.3** Degree of freedom of Robotino.

#### **4.1.2.1 Data Acquisition**

The camera frames and the odometry values are the most frequently acquired data from the Robotino for the system. The frames grabbed are transferred as 320x240x3 pixels in RGB mode and are in good quality to be processed reliably. When the robot does not have any other task to do, Table 4.2 gives the results of a number of experiments on how many frames can be grabbed per second (ps) from robot to the desktop computer, and Table 4.3 similarly indicates at what frequency the odometry information of robot can be queried.

**Table 4.2** Frequencies of frame retrieval.

<b>Experiment</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>Frame ps</b>	29	27	26	29	29	28	30	30	28	29

**Table 4.3** Frequencies of odometry retrieval.

<b>Experiment</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
<b>Odometry ps</b>	3483	3482	3239	3274	3318	3303	3481	3479	3277	3460

The results of experiments indicate that there are no consistent frequencies of frame and odometry retrieval. This is mostly because whenever the data request occurs, some underlying protocols are executed to transfer data via W-LAN from robot to PC,

and some conditions are needed to be satisfied depending on the requested data type to make the data ready for transfer. Due to unpredictable operations of the protocols and data preparation phase, the same type of data retrievals may last in varying durations. However, the results of the experiments prove that in a second 5 frames can be grabbed, and the odometry of robot can be queried 3000 times safely under normal conditions.

#### ***4.1.2.2 Odometry Accuracy***

The odometry performance measure is how well the robot can estimate its own position from the rotation of the wheels. Determination of the odometer accuracy is important because proper noise values should be integrated with the matrices for the computation of uncertain robot positions in EKF system. To measure the odometry accuracy of the Robotino, a number of experiments were conducted in which robot was moved at constant planar or angular velocities along various distances or angles, and how much the odometry differed from the real pose of the robot was estimated by considering the initial position of the robot as ( $x=0$ ,  $y=0$ ,  $\theta=0$ ). Now that the default planar and angular velocities for the system are given as 15 cm/s and 10 deg/s respectively, those velocities were used during the experiments.

The measurements obtained from the odometer of the robot are as in Table 4.4. They indicate that even though the robot is commanded to move along or rotate about an axis by some amount, it may end up with some different pose other than what is estimated. This behavior mostly stems from configuration of the wheels of the Robotino as well as some random factors that may affect the navigation process. Three wheels of the Robotino are placed on its steel frame by having  $120^\circ$  between each. During the experiments, this configuration led to a slight slippage on other axes when the robot was rotating, yet a great change occurred in  $x$  coordinate or  $\theta$  when the robot was moving along  $y$  axis.

However, those slippages were also recorded by the odometry of the robot that was accurate to the extent computed in Table 4.5. Differences between the real pose and the odometry presented no distinguishable positive correlation with respect to the distance covered by the robot. Consequently, two similar approaches were taken into consideration to find an approximate uncertainty for each axis of the robot. The first

approach only considered the errors in the direction of motion, and the second approach averaged the errors in all measurements.

**Table 4.4** Odometry accuracy experiments.

<b>Experiment</b>	<b>Odometer x (mm)</b>	<b>Odometer y (mm)</b>	<b>Odometer <math>\theta</math> (deg)</b>	<b>Real x (mm)</b>	<b>Real y (mm)</b>	<b>Real <math>\theta</math> (deg)</b>
1000 mm on x	1000.3	-22.13	-2.93	1010	21	-2
-1000 mm on x	-1004.5	5.17	-1.13	-1033	-2	1.5
2000 mm on x	2004.3	101.59	5.23	1988	127	7
-2000 mm on x	-2004.5	-47.88	2.19	-2024	-103	0.9
3000 mm on x	3003.6	74.31	3.01	3006	146	6
-3000 mm on x	-3001.7	338.48	-13.5	-2959	388.5	-12
1000 mm on y	142.54	1004.9	-15.82	140	1010	-19.5
-1000 mm on y	118.19	-1003	13.92	149.5	-1026	18.5
2000 mm on y	588.59	2000.3	-32.67	573	2027.5	-31
-2000 mm on y	493.32	-2003.1	28.61	543	-2018	33
3000 mm on y	1556.2	3001.3	-54.45	1529	3045	-56
-3000 mm on y	1321.3	-3000.3	47.80	1365	-3019	50
45° turn	2.84	-0.31	45.21	0	0	45
-45° turn	-3.12	0.57	-45.18	0	0	-45
90° turn	3.56	-0.89	90.08	0	0	92
-90° turn	-4.43	0.30	-90.13	0	-0.6	-93
180° turn	1.21	4.55	180.08	1	5	184
-180° turn	-6.07	2.20	-180.10	-2	2	-182,5

By convention, a robot should not have an error of more than 2 cm per meter of movement and 2° per 45° of turn (Riisgaard and Blas, 2005). Even though, the findings on odometry uncertainties of the Robotino were not computed according to the specific cases of per meter of movement and 45° of turn, the computed uncertainties were close to the indicated uncertainties in either approach of uncertainty estimation.

**Table 4.5** Odometry uncertainties.

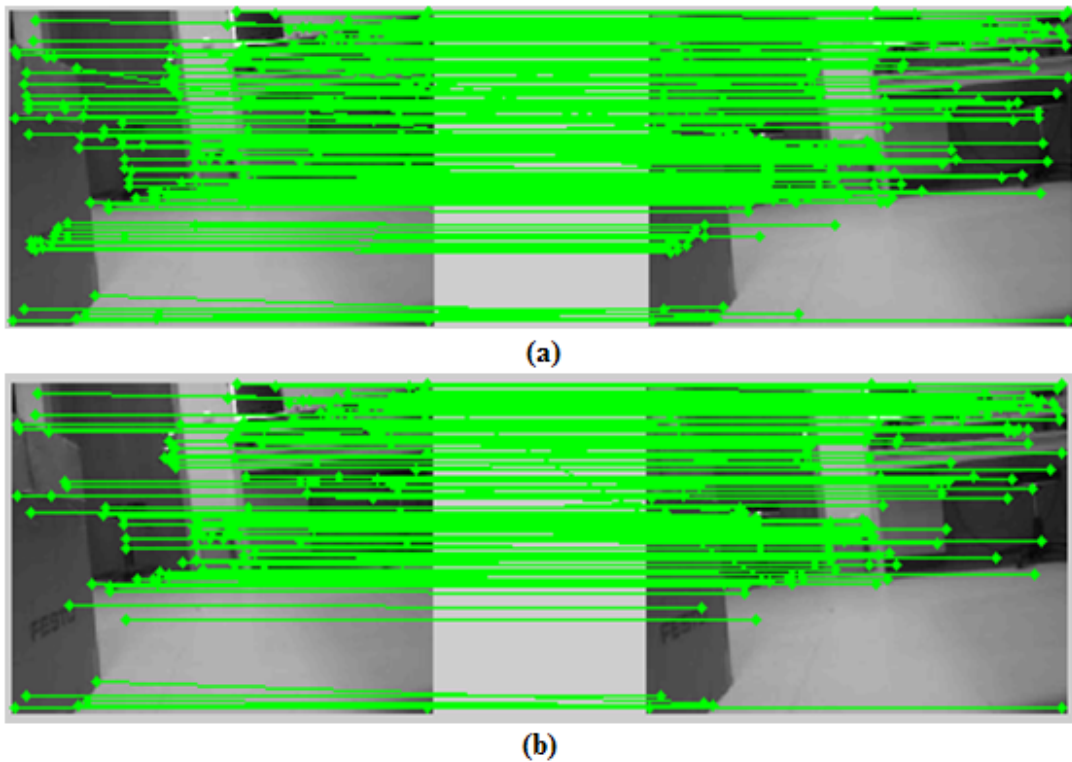
<b>Axis</b>	<b>Uncertainty (Direction of motion)</b>	<b>Uncertainty (Batch)</b>
x	1.97 cm	2.62 cm
y	2.21 cm	2.04 cm
$\theta$	1.92°	2.25°

## 4.2 FEATURE EXTRACTION AND MATCHING

### 4.2.1 Evaluation of the SIFT Algorithm

The number of SIFT features extracted can be kept under control and enhanced by arranging some thresholds related to extreme points detection and keypoint localization steps. Infact, both of the thresholds are to guarantee that the value of an extreme point in DoG images is greater than a specified parameter so that more contrasting extreme points can be selected. Figure 4.4 (a) and (b) exhibit the extracted and the matched SIFT features before and after the threshold update, respectively. As the values of the thresholds increased more contrasted points were obtained, but the number of points extracted were decreased.

The default values for those thresholds were 0.0053 and 0.0067 for Vedaldi's SIFT. Both of the thresholds were changed to 0.01 for this implementation. Table 4.6 through Table 4.9 indicate the performances of the SIFT algorithm and keypoint descriptor matching stages with respect to the number of points extracted after the threshold update. The tests were conducted on 6 consequent frames retrieved at about 5 Hz and 0.5 Hz, when the robot was moving on the x axis with the default velocity of 150 mm/s.



**Figure 4.4** Extracted and matched SIFT features (a) before and (b) after threshold update.

True positive (TP) and false positive (FP) matches were also computed for all matches in each consequent frame pairs. However, the experiments measured the matching accuracy of points in independent frame pairs for simplicity, rather than along a frame sequence which is more important to us to prove repetitiveness of the keypoints. Again, each matching result in the tables may give an intuition on how many of the keypoints detected in the current frame could be found in the next frames.

**Table 4.6** Performance of the SIFT algorithm on frames at 5 Hz.

Frame #	Point #	SIFT Computation Time
1	112	1.73032
2	117	1.9116
3	127	1.9021
4	110	1.8755
5	109	1.8938
6	109	1.9009



**Table 4.7** Performance of the SIFT descriptor matching stage on frames at 5 Hz.

<b>Frames</b>	<b>Point #</b>	<b>Matching Time</b>	<b>TP</b>	<b>FP</b>
<b>1-2</b>	85	0.033	84	1
<b>2-3</b>	100	0.011	95	5
<b>3-4</b>	91	0.008	87	4
<b>4-5</b>	80	0.007	78	2
<b>5-6</b>	86	0.007	85	1

**Table 4.8** Performance of the SIFT algorithm on frames at 0.5 Hz.

<b>Frame #</b>	<b>Point #</b>	<b>SIFT Computation Time</b>
<b>1</b>	112	1.8375
<b>11</b>	92	1.8030
<b>21</b>	81	1.7532
<b>31</b>	75	1.7394
<b>41</b>	68	1.7351
<b>51</b>	85	1.7671

**Table 4.9** Performance of the SIFT descriptor matching stage on frames at 0.5 Hz.

<b>Frames</b>	<b>Point #</b>	<b>Matching Time</b>	<b>TP</b>	<b>FP</b>
<b>1-11</b>	58	0.006	57	1
<b>11-21</b>	41	0.004	39	2
<b>21-31</b>	55	0.003	53	2
<b>31-41</b>	42	0.003	41	1
<b>41-51</b>	34	0.003	32	2

The results demonstrate that even when the frames were grabbed at 0.5 Hz, which means that just one frame is retrieved in 2 seconds; matches of keypoints were reliable enough. Consequently, since the robot was moving perpendicular to the environment during the experiments, the scale invariance of the algorithm was tested and guaranteed.

### 4.2.2 Evaluation of the SURF Algorithm

Similarly, another experiment in which the frames were grabbed at 0.5 Hz was performed with the SURF algorithm to enable a comparison with the SIFT. Table 4.10 shows the computation time of the algorithm for the given number of points and Table 4.11 indicates the reliability and efficiency of the matching stage.

As the values in the tables point out, the matching accuracy of the SIFT and the SURF algorithms was nearly the same, when the scale invariance of the algorithms mattered most. On the other hand, the SURF was about 40 times faster than the SIFT in computing the same number of keypoints. So, the SURF algorithm only enables the realization of the system in real-time.

**Table 4.10** Performance of the SURF algorithm on frames at 0.5 Hz.

Frame #	Point #	SURF Computation Time
1	120	0.0517
11	103	0.0378
21	91	0.0394
31	83	0.0375
41	88	0.0396
51	79	0.0386

**Table 4.11** Performance of the SURF descriptor matching stage on frames at 0.5 Hz.

Frames	Point #	Matching Time	TP	FP
1-11	52	0.004	50	2
11-21	50	0.001	48	2
21-31	55	0.001	51	4
31-41	49	0.001	46	3
41-51	45	0.001	43	2

In addition to these, the number of TP or FP matches given in the tables are purely the matching results of the SIFT or the SURF algorithms. Though, when the matches in which the points are farther than about 15.8 pixels to each other were eliminated, the accuracy of the algorithms was improved further so that nearly no FP matches existed.

### 4.3 CALIBRATION AND PROJECTION TO 3D

For camera calibration, six experiments were conducted on four different frame sequences that had been stored as .mat file in MATLAB to measure the effects of some factors on the computed camera parameters. Frames were chosen from these sequences depending on the total number of images used for calibration by skipping as much frames as possible between two consecutive images. Thus, view point variation was maximized among the images. The factors whose effects were investigated involved the image count, the size of the squares on calibration object, or whether the ratio of  $f_x/f_y$  was fixed or not.

Table 4.12 gives the average reprojection errors computed after the 3D corner points of the calibration object in the world were reprojected to their image coordinates by using the estimated camera matrix and distortion parameters in Table 4.13 and Table 4.14, respectively. Thus, Table 4.12 indicates how successful each calibration experiment was.

**Table 4.12** Average reprojection errors of the calibration routines.

<b>Experiment Number</b>	<b>Square Size</b>	<b>Flags</b>	<b>Image Count</b>	<b>Average Reprojection Error</b>
<b>1</b>	1	none	20	0.1603
<b>2</b>	1	none	60	0.1647
<b>3</b>	1	none	80	0.1667
<b>4</b>	39 mm	none	100	0.1651
<b>5</b>	1	none	100	0.1651
<b>6</b>	1	Fix ratio of $f_x/f_y$	100	0.3161

**Table 4.13** Camera matrix parameters computed.

Experiment Number	$f_x$	$f_y$	$c_x$	$c_y$
1	338.19	332.96	165.28	112.03
2	339.36	334.26	165.23	111.63
3	344.07	338.77	165.27	113.21
4	343.93	338.61	165.89	113.09
5	343.93	338.61	165.89	113.09
6	312.04	312.04	164.44	82.90

**Table 4.14** Distortion parameters computed.

Experiment Number	$k_1$	$k_2$	$p_1$	$p_2$
1	0.0664	-0.1001	-0.0045	-0.0018
2	0.0622	-0.0671	-0.0045	-0.0017
3	0.0611	-0.05845	-0.0043	-0.0015
4	0.0621	-0.0651	-0.0042	-0.0013
5	0.0621	-0.0651	-0.0042	-0.0013
6	0.0882	-0.0777	-0.0200	-0.0017

The first four experiments were to comprehend if increasing the number of images enhances the calibration process. However, since the same frame sequences were used to choose the images, the more the number of images used to calibrate the camera was, the more corner point locations with similar characteristics were obtained. Hence, in this case, increasing the image count may result in overfitting the camera parameters to the data. The fourth and the fifth experiments were to prove that changing the way points are represented in the object coordinate does not affect the parameters estimated. Because, just the unit of the points was changed, and the units canceled each other while computing the parameters. Finally, the last experiment was run based on the assumption that the ratio of the parameters  $f_x$  and  $f_y$  is 1, which means that the camera has square pixels. But, the average reprojection error given in Table 4.12 for the sixth experiment refuted the assumption.

All in all, the first experiment run with image count of 20 can be regarded as successful enough. Consequently, the camera matrix and the distortion parameters computed through this experiment can be used during the projection stage of the 2D features extracted by the SIFT or the SURF algorithms to 3D world coordinates.

#### 4.4 LOCALIZATION OF THE ROBOT AND THE LANDMARKS

The EKF module was run only used on the odometry of the robot since the landmark positions in the environment could not be estimated. Consequently, increasing uncertainty in the robot position is expected for such localization. The two experiments in which the Robotino navigated along a rectangular path and a circular path showed parallel results to this expectation.

Table 4.15 and Table 4.16 contain the diagonal uncertainties of covariance matrix  $P$  belonging to the positions  $x$  and  $y$ , and the rotation  $\theta$  of the robot for the first 10 iterations of the EKF called by the rectangular and the circular path controllers, respectively. The tables also indicate the controls  $\Delta x$ ,  $\Delta y$  and  $\Delta\theta$  computed based on the odometry when the paths were being executed by the robot. The controls  $\Delta x$  and  $\Delta y$  were added to the world coordinates of the robot after they were transformed according to the current pose  $\theta$  of the robot to find the recent coordinates. The recent pose of the robot was then simply  $\theta + \Delta\theta$ . The units of the numerical values in the tables are centimeter or degree for positions, controls, and the errors; and  $\text{cm}^2$  or  $\text{degree}^2$  for the covariances.

The rectangular path involved the motion commands: 300 mm on  $x$ ,  $90^\circ$  turn, 200 mm on  $x$ ,  $90^\circ$  turn, 300 mm on  $x$ ,  $90^\circ$  turn, 200 mm on  $x$ , and  $90^\circ$  turn in order. The circular path was generated by setting sinusoidal velocities ranging from 20 cm/s to -20 cm/s in  $x$  or  $y$  directions. The true and the estimated positions of the robot again during the first 10 iterations of EKF are in Table 4.17 and Table 4.18 for the rectangular and the circular path executions, respectively.

**Table 4.15** Diagonal values of covariance matrix P and controls computed based on the odometry for the rectangular path.

#	P(1,1)	P(2,2)	P(3,3)	Control $\Delta x$	Control $\Delta y$	Control $\Delta \theta$
1	1.0002	1.3185	0.0114	5.6580	0.0197	-0.0049
2	1.0054	7.7086	0.0127	19.2751	-0.1065	-0.0073
3	1.0722	24.1152	0.0141	20.3401	-0.3168	-0.0122
4	1.0760	25.8409	0.0155	1.5874	0.0057	-0.0002
5	1.0695	25.7704	0.0169	-0.0717	0.0959	0.2344
6	1.0689	25.8754	0.0182	0.0974	-0.0170	0.1948
7	1.0610	26.0687	0.0196	0.2228	0.0794	0.1963
8	1.0632	25.9715	0.0210	-0.0857	0.0203	0.2040
9	1.0610	26.0832	0.0223	0.1023	-0.0284	0.1796
10	1.0573	26.1292	0.0237	0.0780	0.0204	0.1916

**Table 4.16** Diagonal values of covariance matrix P and controls computed based on the odometry for the circular path.

#	P(1,1)	P(2,2)	P(3,3)	Control x	Control y	Control $\theta$
1	1.0002	1.3448	0.0114	5.8724	-0.0083	-0.0044
2	1.0004	1.6803	0.0127	2.3350	0.0195	-0.0002
3	1.0010	2.6759	0.0141	4.3905	0.0302	0.0010
4	1.0010	4.4077	0.0155	4.9214	0.0002	0.0030
5	1.0245	6.6687	0.0169	4.5128	1.5570	0.0026
6	1.2214	8.9898	0.0182	3.7081	2.4450	-0.0012
7	1.6978	10.9325	0.0196	2.7778	2.6499	0.0024
8	2.9168	12.8457	0.0210	2.6083	3.8606	0.0019
9	4.9505	14.3800	0.0223	1.9463	4.1977	-0.0014
10	7.9300	14.9443	0.0237	0.9392	4.4484	0.0012

**Table 4.17** The true and the estimated positions of the robot for the rectangular path.

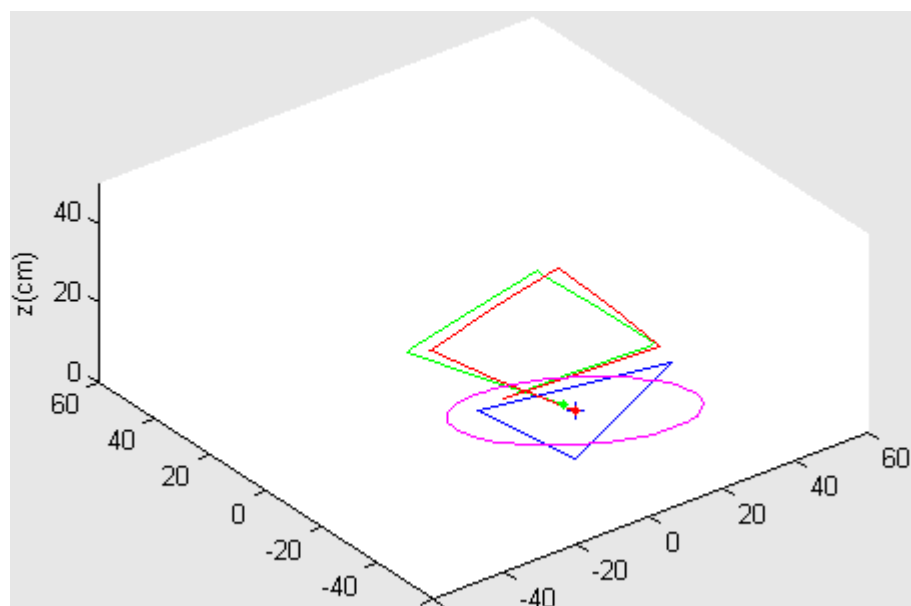
#	$x_{true}$	$y_{true}$	$\theta_{true}$	$x_{Est}$	$y_{Est}$	$\theta_{Est}$
1	5.6580	0.0197	-0.0049	5.6422	0.0229	-0.0303
2	24.9324	-0.1810	-0.0122	24.8995	-0.6635	-0.0716
3	45.2671	-0.7463	-0.0244	45.1759	-2.4310	-0.0466
4	46.8542	-0.7794	-0.0246	46.7592	-2.4899	-0.0760
5	46.7849	-0.6818	0.2098	46.6956	-2.3774	0.1414
6	46.8837	-0.6781	0.4046	46.7898	-2.3648	0.3156
7	47.0573	-0.5175	0.6009	46.9627	-2.2175	0.5114
8	46.9751	-0.5492	0.8049	46.8756	-2.2557	0.7245
9	47.0665	-0.4952	0.9845	46.9753	-2.2094	0.9101
10	47.0927	-0.4189	1.1762	47.0162	-2.1336	1.0933

**Table 4.18** The true and the estimated positions of the robot for the circular path.

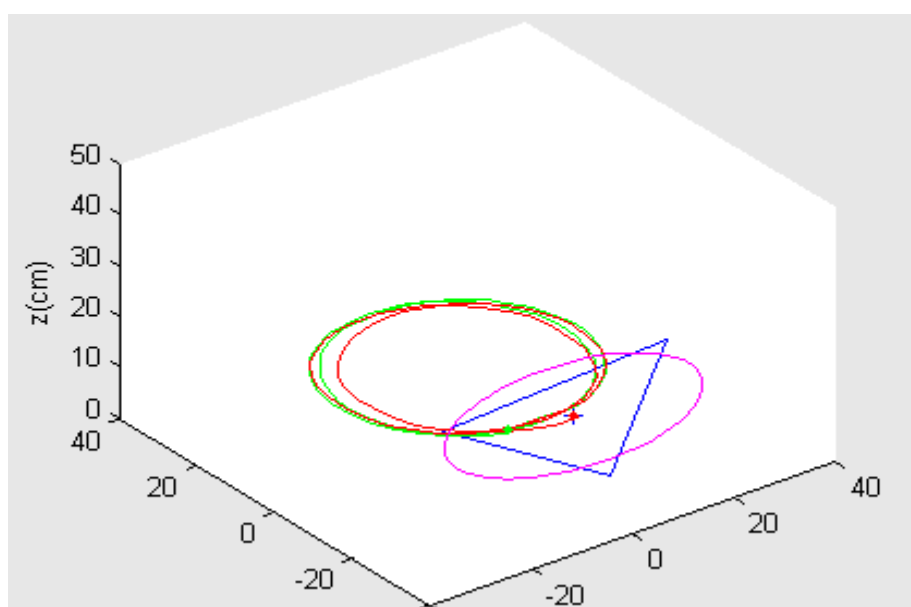
#	$x_{true}$	$y_{true}$	$\theta_{true}$	$x_{Est}$	$y_{Est}$	$\theta_{Est}$
1	5.8724	-0.0083	-0.0044	5.8700	0.0068	-0.0233
2	8.2075	0.0010	-0.0045	8.2001	-0.0319	-0.0408
3	12.5981	0.0113	-0.0035	12.5968	-0.1833	0.0087
4	17.5195	-0.0057	-0.0005	17.5178	-0.1217	-0.0520
5	22.0331	1.5489	0.0021	22.1091	1.2014	0.0013
6	25.7361	4.0016	0.0009	25.8166	3.6690	0.0496
7	28.5115	6.6539	0.0033	28.4539	6.4412	0.0745
8	31.1070	10.5232	0.0052	30.7595	10.4831	0.0615
9	33.0313	14.7310	0.0038	32.4447	14.7985	0.0788
10	33.9534	19.1830	0.0051	33.0330	19.2977	0.0578

The overall trajectories the Robotino followed were drawn in a coordinate system defined in centimeters as Figure 4.5 and Figure 4.6 illustrates. The robot is represented as a triangle in color of blue whose initial position and heading are defined as ( $x=0$ ,

$y=0$ ,  $\theta=0$ ) and towards the  $x$ -axis, respectively. The green lines are the trajectories composed of the true positions of the robot without noise, and the red lines are the estimated positions in EKF into which the uncertainties estimated for the odometry in Subsection 4.1.2.2 were incorporated. The magenta ellipses around the triangle robot are the uncertainties of the final robot locations.



**Figure 4.5** Trajectories drawn for the robot navigation along a rectangular path.



**Figure 4.6** Trajectories drawn for the robot navigation along a circular path.



After the navigations along the paths terminated, Table 4.19 shows the final localization results, uncertainties in the localizations, and errors between the true and the estimated positions of the robot for both the rectangular and the circular paths.

**Table 4.19** Evaluations on final robot localization for rectangular and circular paths.

	<b>Rectangular Path</b>	<b>Circular Path</b>
$x_{Est}$	14.9480	17.0772
$y_{Est}$	-13.4143	-2.5110
$\theta_{Est}$	0.1425	0.4036
<b>P(1,1)</b>	72.4307	71.0824
<b>P(2,2)</b>	40.2523	25.8292
<b>P(3,3)</b>	0.0731	0.1032
$x_{Err}$	0.26414	11.3068
$y_{Err}$	3.9231	2.4169
$\theta_{Err}$	0.1902	0.39223

## CHAPTER 5

### CONCLUSIONS AND FUTURE WORK

#### 5.1 CONCLUSIONS

According to the current knowledge in the literature, design of a purely vision-based autonomous system that can work in any environment is still a challenging research problem. The studies mostly fuse information obtained through other sensors with visionary information to increase the accuracy of the system and are limited to mapping the environment sparsely whereas localizing the robot rather than having designated goals. Similarly, the system presented in this thesis is a derivation of visionary SLAM applications in the literature, which fuses the information of the odometer and the single camera of a mobile robot navigating on a planar ground.

A typical visionary SLAM system involves many difficulties. Because, the stage of frame processing takes too much time due to very high input data rate of camera and complexity of the robust feature extraction algorithms, tracking the features among frames persistently constitutes a problem, the direct depth measurement to environment landmarks is something hard to achieve accurately using just a single camera, and running the EKF system depending on sometimes unreliable outputs of the previous stages with increasing number of landmarks detected can not always yield reasonable localization results and puts an extra burden on the system performance.

However, it could not be possible to experience all of these difficulties since the system designed could be realized partially. The reason was that 3D environment coordinates of the 2D features in the frames are not estimated, and hence the EKF could not be run with the landmark coordinates as required. On the other hand, the problem of reliable feature matching, the effect of complex feature extraction task on the running

system, and the increase in the uncertainty of the robot localization without the landmarks detected in the environment could be observed as indicated in the studies referenced.

Ultimately, conducting this research is important in order to comprehend the requirements and difficulties in implementation of a visionary SLAM algorithm which is considered to be going to have an impact in some application areas including low-cost and advanced robotics, wearable computing, augmented reality for industry and entertainment, or medical imaging (Davison et al., 2007). Consequently, this study is like an initial step to learn the environment through exploratory motions towards a more advanced and goal-driven vision-based systems such as a humanoid capable of cleaning a home or a mobile phone application aiding blind people in real-time for structure estimation of the environment.

## **5.2 FUTURE WORK**

In the scope of the implemented system, there are a number of enhancements that can be done to improve its operation and performance in the future. Firstly, the Landmark Parameter Estimator module should be implemented properly. Also, some alternative algorithms may be proposed for the module to estimate the 3D coordinates of the landmarks in the environment more precisely. Secondly, the SIFT algorithm may be completely replaced by the SURF algorithm which is known to be faster than and nearly as accurate as SIFT. Thirdly, most of the image processing task may be transferred to a GPU which reduces the CPU work and can boost the system speed up to approximately 12 times paving way for a successful real-time operation. This can be achieved by integrating CUDA libraries to the application, which enables C++ applications to utilize GPU as an auxiliary processor. Besides, some pertinent algorithms can be employed in the Path Planner and the Controller modules for the system to carry out some goals.

From a wider perspective, recovering detailed 3D surface maps in real-time rather than sets of sparse landmarks is the future goal of visionary SLAM applications. And designing genuinely practical systems requires coping with both indoor and outdoor environments, more dynamic motions, more complicated scenes with significant occlusions and changing lighting conditions (Davison et al., 2007).

## REFERENCES

- Argall, B.D. and Chernova, S., Veloso, M. and Browning, B., 2009, "A Survey of Robot Learning From Demonstration", *Robotics and Autonomous Systems* 57, 469-483, 2009.
- Atiya, S. and Hager, G.D., "Real-Time Vision-Based Robot Localization", *IEEE Transactions on Robotics and Automation*, December 1993.
- Bay, H., Tuytelaars, T., and Gool, L.V., "SURF: Speeded Up Robust Features", *European Conference on Computer Vision*, 2006.
- Bouget, J.-Y. and Perona, P., "Visual Navigation Using a Single Camera", *ICCV Proceedings*, 1995.
- Bradski, G. and Kaehler, A., *Learning OpenCV Computer Vision with the OpenCV Library*, O'Reilly, USA, 2008.
- Bradski, G. and Kaehler, A., "Robot-Vision Signal Processing Primitives", *IEEE Signal Processing Magazine*, January 2008.
- Davison, A. J. and Murray, D. W., "Simultaneous Localization and Map-Building Using Active Vision", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, July 2002.
- Davison, A. J., "Real-Time Simultaneous Localization and Mapping with a Single Camera", *Proceedings of the Ninth IEEE International Conference on Computer Vision*, 2003.
- Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O., "MonoSLAM: Real-Time Single Camera SLAM", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, June 2007.
- DeSouza, G.N. and Kak, A.C., "Vision for Mobile Robot Navigation: A Survey", *IEEE Transactions On Pattern Analysis And Machine Intelligence*, February 2002.
- Dev, A., Kröse, B., and Groen, F., "Navigation of a Mobile Robot on the Temporal Development of the Optic Flow", *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, September 1997.
- Hager, G., *Introduction to Robotics: Lecture 9*, 2008,  
<http://see.stanford.edu/see/lecturelist.aspx?coll=86cc8662-f6e4-43c3-a1be-b30d1d179743>

- Harris, C. and Stephens, M., “A Combined Corner and Edge Detector”, Proceedings of the 4th Alvey Vision Conference (pp. 147–151), 1988.
- Jia, Z., Balasuriya, A., and Challa, S., “Recent Developments in Vision Based Target Tracking for Autonomous Vehicles Navigation”, in Proceedings of the IEEE Intelligent Transportation Systems Conference, Toronto, Canada, September 2006.
- Kosaka, A. and Kak, A.C., “Fast Vision-Guided Mobile Robot Navigation Using Model-Based Reasoning and Prediction of Uncertainties”, Computer Vision, Graphics, and Image Processing Image Understanding, 1992.
- Kracht, S. and Nielsen, C., Robots in Everyday Human Environments On Platform Development and Behaviour Dependent Control, M.S. Thesis, Aalborg University, 2007.
- Lowe, D.G., “Distinctive Image Features from Scale-Invariant Keypoints”, Vancouver, B.C., Canada, January 2004.
- Lucas, B. D. and Kanade, T., “An Iterative Image Registration Technique with an Application to Stereo Vision”, Proceedings of the 1981 DARPA Imaging Understanding Workshop (pp. 121–130), 1981.
- Matsumoto, Y., Inaba, M., and Inoue, H., “Visual Navigation Using View-Sequenced Route Representation”, Proceedings of the IEEE International Conference on Robotics and Automation, April 1996.
- McAndrew, A., An Introduction to Digital Image Processing with MATLAB, Australia, 2004.
- Meng, M. and Kak, A.C., “NEURO-NAV: A Neural Network Based Architecture for Vision-Guided Mobile Robot Navigation Using Non-Metrical Models of the Environment”, Proceedings of the IEEE Int’l Conference on Robotics and Automation, 1993.
- Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptoch, A., Magnenat, S., Zufferey, J.C., Floreano, D., and Martinoli, A., “The E-puck, a Robot Designed for Education in Engineering”, Proceedings. of the 9th Conference on Autonomous Robot Systems and Competitions, 2009.
- Moravec, H.P., “The Stanford Cart and the CMU Rover”, Proceedings of the IEEE, July 1983.
- Newman, P., C4B Mobile Robots Example MATLAB Code, 2003, <http://www.robots.ox.ac.uk/~pnewman/Teaching/C4CourseResources/Matlab.html>
- Pan, J., Pack, D.J., Kosaka, A., and Kak, A.C., “FUZZY-NAV: A Vision-Based Robot Navigation Architecture Using Fuzzy Inference for Uncertainty-Reasoning”, Proceedings of the IEEE World Congress Neural Networks, July 1995.
- Pomerleau, D.A., ALVINN: An Autonomous Land Vehicle in a Neural Network, Technical Report CMU-CS-89-107, Carnegie Mellon University, 1989.

- Riisgaard, S. and Blas, M.R., SLAM for Dummies: A Tutorial Approach, MIT, 2005.
- Russell, S. and Norvig, P., Artificial Intelligence: A Modern Approach, 2nd ed., chapters 2, 24-25, Prentice Hall, 2003.
- Santos-Victor, J., Sandini, G., Curotto, F., and Garibaldi, S., "Divergent Stereo for Robot Navigation: Learning from Bees", Proceedings of the IEEE CS Conference on Computer Vision and Pattern Recognition, 1993.
- Saxena, A., Wong, L., Quigley, M., and Ng, A.Y., "A Vision-based System for Grasping Novel Objects in Cluttered Environments", International Symposium of Robotics Research (ISRR), 2007.
- Schepelmann, A., Snow, H.H., Hughes, B.E., Merat, F.L., and Quinn, R.D., "Vision-based Obstacle Detection and Avoidance for the CWRU Cutter Autonomous Lawnmower", IEEE, 2009.
- Se, S., Lowe, D., and Little, J., "Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks", The International Journal of Robotics Research, Vol. 21, No. 8, pp. 735-758, August 2002.
- Shi, J. and Tomasi, C., "Good Features to Track", 9th IEEE Conference on Computer Vision and Pattern Recognition, June 1994.
- Sinha, U., SIFT: Scale Invariant Feature Transform, 2010,  
<http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/>
- Spong, M.W., Hutchinson, S., and Vidyasagar, M., Robot Modeling and Control, John Wiley, 2006.
- Thorpe, C., Kanade, T., and Shafer, S.A., "Vision and Navigation for the Carnegie-Mellon Navlab", Proceedings on Image Understand Workshop, 1987.
- Thrun, S., "Learning Metric-Topological Maps for Indoor Mobile Robot Navigation", Artificial Intelligence, February 1998.
- Vedaldi, A., Code - SIFT for MATLAB, 2006,  
<http://www.vlfeat.org/~vedaldi/code/sift.html>
- Velat, S.J., Lee, J., Johnson, N., and Crane, C.D., "Vision Based Vehicle Localization for Autonomous Navigation", Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, Jacksonville, FL, USA, June 2007.
- Yang, J., Rao, D., Chung, S.-J., and Hutchinson, S., "Monocular Vision based Navigation in GPS-Denied Riverine Environments", American Institute of Aeronautics and Astronautics, 2010.
- Yang, J., Rao, D., Chung, S.-J., and Hutchinson, S., "A Monocular Vision Based Navigation Algorithm for MAVs in Multiple GPS-Denied Environments", 2012.

## **APPENDIX A**

### **ROBOTINO**

The three Robotino drive modules are integrated into a stable, laser-welded stainless steel frame. The frame is protected against collisions by a rubber protection strip with integrated switching sensor. Numerous additional components, such as sensors or handling units, can be mounted on a platform with prepared threaded holes. With its omnidirectional drive, Robotino moves quick as a flash forwards, backwards, and sideways, and also turns on the spot. Three sturdy DC industrial motors with optical shaft encoders having a resolution of 2048 increments per revolution and gear units with a reduction ratio of 1:16 allow speeds of up to 10 km/h.

The frame contains nine infrared distance sensors. An analogue inductive sensor and two optical sensors available enable the Robotino to recognize and follow predefined paths that are marked in colour or with an aluminium strip. The Robotino is supplied with a colour web camera with jpeg compression. The compressed web camera image can be transmitted to an external PC via the WLAN for image evaluation by external PC or used as a live camera image. Power is supplied via two 12 V lead-gel rechargeable non-spillable electric storage batteries, permitting a running time of up to two hours.

At the heart of the PC 104 controller is the real-time Linux operating system, provided on a 1 GB CF card. This communicates with the new EA09 control board via a serial interface, to evaluate the sensor data and control the Robotino's drive units. It can also communicate directly with a Linux program in the PC 104 or with another external PC application via W-LAN. Robotino APIs are available in order to program the robot using the languages such as C++, .Net, JAVA, or MATLAB. The EA09 control board is fitted with an interface card, which provides four Ethernet interfaces, one of which has a direct external link. At the heart of EA09 control

board is an LPC2377 32-bit microcontroller, which directly generates the PWM signals for controlling four electric DC motors. Xilinx Spartan3 FPGA to read the encoder values for the motors. This enables the odometer data and any additional sensor-specific correction data to be calculated directly in the microcontroller. This results in a considerable improvement in accuracy. The microcontroller is externally accessible and can be used directly for programming custom applications. The microcontroller firmware can be updated via operating system of the Robotino.

**Table A.1** Robotino hardware specification.

<b>Robot</b>
Diameter of 370 mm
Height including housing without web cam of 210 mm
Total weight of about 11 kg
Three omnidirectional drive units each featuring a 3600 rpm Dunker motor
Maximal payload of about 5 kg
<b>Sensors</b>
Rubber protection strip with built-in collision-protection sensor
9 analogue infrared distance sensors
Analogue inductive sensor
2 digital optical sensors
Colour web camera with USB interface and jpeg compression
Three optical shaft encoders
<b>Embedded Controller</b>
Embedded PC 104 with AMD LX800 processor (500 MHz)
SDRAM 64 MB
Compact Flash card 1GB
WLAN access point with antenna to 802.11g and 802.11b, client mode and optional WPA2 encoding
Interfaces: Ethernet, 2 x USB, 2x RS232 and VGA connection
<b>I/O Interface Card</b>
Outputs for controlling the three omnidirectional drive units
8 analogue inputs (0 – 10 V, 50 Hz), 2 analogue outputs
8 digital inputs, 8 digital outputs (24 V, short circuit proof and overload proof)
2 relays for additional actuators



## APPENDIX B

### E-PUCK

E-puck is the miniaturization of a complex robotic system that is sold at a low price. The sensors, actuators, and interfaces of the E-puck are representatives of a wide range of devices that can be found in several engineering subdomains. It has two types of processors: general purpose and DSP. Sensors are in different modalities like audio, visual, distances to objects, and gravity. Input devices are with different bandwidths from 10 Hz to 10 MHz. Actuators are with different actions on the environment such as displacement, audio, or display. It enables wired and wireless communication with other devices. Figure B.1 illustrates the electronic structure whereas Table B.1 presents the hardware specification of E-puck (Mondada et al., 2009).

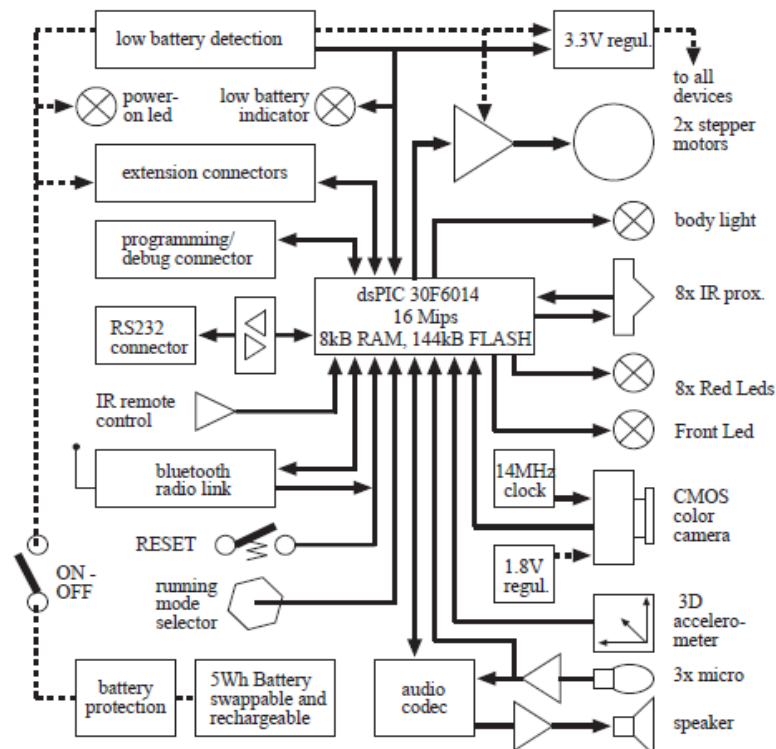


Figure B.1 Outline of electronic circuitry of the E-puck.

**Table B.1** E-puck hardware specification.

<b>Robot</b>
Diameter of 75 mm
Height depends on the connected extensions
Four injected plastic parts: the main body, the light ring, and the two wheels
Main PCB containing most of the electronics is on top of the main body
Main body encloses battery
<b>Sensors</b>
Eight infrared (IR) proximity sensors (10-100 Hz)
A 3D accelerometer to measure the inclination and the acceleration (0.1-1 kHz)
Three microphones ( 10-40 kHz )
A color CMOS camera with a resolution of 640_480 pixels (10k-10MHz)
<b>Actuators</b>
Two stepper motors (1000 steps per wheel revolution)
A speaker connected to an audio codec
Eight red light emitting diodes (LED)
A set of green LEDs
A red front LED placed beside the camera
<b>Microcontroller</b>
dsPIC30F6014 by Microchip
A 16 bit processor running at 30 MHz
A 16 entry register file
A digital signal processor (DSP) unit
RAM 8 kB
Flash memory 144 kB
<b>Interfaces</b>
Two LEDs for powering status of robot and the status of the battery
A connector to interface to an in-circuit debugger
An infrared remote control receiver
A classic RS232 serial interface.
A Bluetooth radio link
A reset button.
A 16 positions rotary switch to specify a 4 bit number

## APPENDIX C

### MATLAB CODE OF CONTROLLER

```
function
controller(robotCntrl,type,algorithm,pathText,infobox,odomBox)
    global robotino xErr;
    %ROBOTINO CONNECTION ENABLE
    robotino=getRobotino();
    robotino.planar = robotCntrl.planar;
    robotino.angular = robotCntrl.angular;
    robotino.frameHz = robotCntrl.frameHz;
    Odometry_set( robotino.OdometryId, 0, 0, 0 );
    setInfoBox(infobox,'Now the system is running...');
    if strcmp(type,'Circular')
        controlCircular(algorithm, infobox, odomBox);
    else
        controlRectangular(pathText, algorithm, infobox, odomBox);
    end;
    plotTrajectories();
    setInfoBox(infobox,'Path execution is complete. ');
    setInfoBox(infobox,['The error between real and estimated
        positions of the robot:' char(10) 'x: ' num2str(xErr(1)) ' cm'
        char(10) 'y: ' num2str(xErr(2)) ' cm' char(10) 'phi: '
        num2str(xErr(3)) ' deg' ]);
    releaseRobotino(robotino);
end;
function controlCircular(algorithm, infobox, odomBox)
    global robotino enable1 systemStop;
    currentFrame=[]; prevFrame=[];
    f1=[];d1=[];s1=[];
    iteration=0;
    EKFInit([1 0 0 200]);
    setOdomBoxes(odomBox);
    startVector= [200.0; 0.0];
    a = 0.0;
    tmElapsed = 0;
    setInfoBox(infobox,'The robot is executing a circular path. ');
    tmStart = tic;
    prevOdom=[0 0 0]';
    x=0; y=0; phi=0; i=0;
    enable1 = 1;
    while(~systemStop && tmElapsed <= 20)
        dir=J2([0;0;deg2rad(a)],[])*[startVector;0];
        dir=sum(dir,2);
        %rotate 360 degrees in 10s
        a = 360.0 * tmElapsed / 10;
        OmniDrive_setVelocity(robotino.OmniDriveId,dir(1),dir(2),0);
        tmElapsed= toc(tmStart);
        %Capture frames with the input Hz
```

```

if(enable1)
    sleep1(robotino.frameHz);
    enable1=0;
    currentFrame=getRobotinoFrame(robotino);
    %Sample odometry nearly at the same time:important
    [ x, y, phi ] = Odometry_get( robotino.OdometryId );
    Odometry_set( robotino.OdometryId, 0, 0, 0 );
    prevOdom=[0 0 0]';
    i=i+1;
    %Apply the selected algorithm to camera frames
    [prevFrame,f1,d1,s1] = callFeatureExtractor(algorithm,
                                                currentFrame, prevFrame,f1,d1,s1);

    %compute control
    [odom,prevOdom,iteration] =
        callEKF(odomBox,[x;y;phi],prevOdom,iteration);
end
wait( 0.05 );
end;
OmniDrive_setVelocity(robotino.OmniDriveId, 0, 0 ,0);
wait(0.1);
[ x, y, phi ] = Odometry_get( robotino.OdometryId );
Odometry_set( robotino.OdometryId, 0, 0, 0 );
[odom,prevOdom,iteration] =
    callEKF(odomBox,[x;y;phi],prevOdom,iteration);
end;
function controlRectangular(pathText, algorithm, infobox, odomBox)
    global robotino enable1 systemStop;
    global path;
    currentFrame=[]; prevFrame=[];
    f1=[];d1=[];s1=[];
    path = processPath(str2path(pathText));
    nPieces = size(path,1);
    iteration=0;
    EKFINit(path);
    setInfoBox(infobox,'Now the system is running...');
    setOdomBoxes(odomBox);
    enable1 = 1;
    for loop=1:nPieces
        if(systemStop)
            break;
        end;
        setInfoBox(infobox,['Executing the path piece: ' char(10)
                            num2str(path(loop,:)) ]]);

        piece = path(loop,:);
        x=0; y=0; phi=0; i=0;
        prevOdom=[0 0 0]';
        distance = piece(4);
        if(distance > 0)
            velocity = robotino.planar;
            angVelocity = robotino.angular;
        else
            velocity = -robotino.planar;
            angVelocity = -robotino.angular;
        end
        tmStart = tic;
        if piece(1) %movement on x axis
            OmniDrive_setVelocity(robotino.OmniDriveId,velocity,0,0);
            wait(0.1);
            while (~systemStop && abs(x) < abs(distance))
                %Capture frames with the input Hz per second
                if(enable1)

```

```

        sleep1(robotino.frameHz);
        enable1=0;
        currentFrame=getRobotinoFrame(robotino);
        %Sample odometry nearly at the same time:important
        [ x, y, phi ]=Odometry_get(robotino.OdometryId );
        i=i+1;
        %Apply the selected algorithm to camera frames
        [prevFrame,f1,d1,s1] =
callFeatureExtractor(algorithm,currentFrame,prevFrame,f1,d1,s1);
        %compute control
        [odom,prevOdom,iteration] =
            callEKF(odomBox,[x;y;phi],prevOdom,iteration);
    end
    if(toc(tmStart)>=(distance/velocity))
        [ x, y, phi ] = Odometry_get( robotino.OdometryId );
    end
end
elseif piece(2) %movement on y axis
    OmniDrive_setVelocity(robotino.OmniDriveId,0,velocity,0);
    wait(0.1);
    while (~systemStop && abs(y) < abs(distance))
        %Capture frames with the input Hz per second
        if(enable1)
            sleep1(robotino.frameHz);
            enable1=0;
            currentFrame = getRobotinoFrame(robotino);
            %Sample odometry nearly at the same time:important
            [ x, y, phi ] = Odometry_get(robotino.OdometryId);
            i=i+1;
            %Apply the selected algorithm to camera frames
            [prevFrame,f1,d1,s1] =
callFeatureExtractor(algorithm,currentFrame,prevFrame,f1,d1,s1);
            %compute control
            [odom,prevOdom,iteration] =
                callEKF(odomBox,[x;y;phi],prevOdom,iteration);
        end
        if(toc(tmStart)>=(distance/velocity))
            [ x, y, phi ] = Odometry_get( robotino.OdometryId );
        end
    end
elseif piece(3) %rotation about z
    OmniDrive_setVelocity(robotino.OmniDriveId,0,0,angVelocity);
    wait(0.1);
    direction = (distance > 0);
    distance = degreeWrap(distance);
    phiRead=0;
    while (~systemStop)
        if abs(phiRead)>3 && ((direction && phi >= distance)||
            (~direction && phi <= distance))
            break;
        end
        %Capture frames with the input Hz per second
        if(enable1)
            sleep1(robotino.frameHz);
            enable1=0;
            currentFrame = getRobotinoFrame(robotino);
            %Sample odometry nearly at the same time:important
            [x,y,phiRead] = Odometry_get(robotino.OdometryId);
            phi=degreeWrap(phiRead);
            i=i+1;

```

```

        %Apply the selected algorithm to camera frames
        [prevFrame, f1, d1, s1] =
callFeatureExtractor(algorithm, currentFrame, prevFrame, f1, d1, s1);
        %compute control
        [odom, prevOdom, iteration] =
            callEKF(odomBox, [x;y;phi], prevOdom, iteration);
    end
    if(toc(tmStart) >= (distance/angVelocity))
        [x, y, phiRead] = Odometry_get(robotino.OdometryId);
        phi=degreeWrap(phiRead);
    end
end
end
end
OmniDrive_setVelocity(robotino.OmniDriveId, 0, 0 ,0);
wait(0.1);
[ x, y, phi ] = Odometry_get( robotino.OdometryId );
Odometry_set( robotino.OdometryId, 0, 0, 0 );
[odom, prevOdom, iteration] =
    callEKF(odomBox, [x;y;phi], prevOdom, iteration);
tmElapsed = toc(tmStart);
setInfoBox(infoBox, ['Path piece is executed in:'
                    num2str(tmElapsed) ' seconds']);
end;
end;

```