Ben-Gurion university of the Negev
Faculty of Engineering Sciences
Department of Industrial Engineering and Management

Evaluating Grid-Map Based Sensor Fusion mapping algorithms for Autonomous Mobile Robots

Thesis submitted in partial fulfillment of the requirements
for the M.Sc degree

Keren Kapach

August 2007

Evaluating Grid-Map Based Sensor Fusion algorithms for Autonomous Mobile Robots

Thesis submitted in partial fulfillment of the requirements
for the M.Sc degree

Keren Kapach

Supervised by: Prof. Yael Edan

Author:______________________________ **Date:**____________

Supervisor:___________________________ **Date:**____________

Chairman of Graduate Studies Committee:________________ **Date:**_____________

2007 תשס"ז

BEER – SHEVA

This work was carried out under the supervision of

Prof. Yael Edan

In the Department of Industrial Engineering and Management

Faculty of Engineering Sciences

For my beloved husband and best friend, Ido.

"If I have seen farther than other men, it is
because I have stood on the shoulders of giants"
Sir Isaac Newton

Acknowledgments

To my advisor, **Prof. Yael Edan**, thanks for the dedicated guidance throughout the past two years, even on tight schedules and crazy deadlines that we knew well during this work. Thanks for the professional assistance and for making all the efforts for finding financial and technical support whenever I needed it. Thanks for inspiring me to pursue academic and personal excellence; I hope our paths will cross again soon.

Thanks for **Dr. Ofir Cohen**, for his smart and helpful comments; thanks for helping me achieving the tremendous goal of getting into your huge shoes.

I would like to thank the labs team **Yossi Zahavi**, **Nisim Abuhazira**, **Rubi Gartner** and **Paul Erez** for their support and assistance inside and outside the labs.

Special thanks for the talented programmer, **Oren Braitstein**, for helping me so much with the code which gave me a great starting point.

To **Juan Wax**, **Uri Cartoon** and **Shahar Laykin** – thanks for the professional help and for contributing me from your knowledge and wide experience, you've all been a great help.

To all the Automation course TA friends, **Ziv Har Zahav**[bm], **Yael Salzer**, **Amit Gil** and **Yuval Oren**, thanks for the understanding and for covering up for me, which allowed me to complete this thesis.

To my family, **Mom, Dad** and my sisters – **Gali** and **Hadar**, thanks a lot for all the mental support and patience you gave me during the long hours I spent in the labs and in the university.

Finally, I would like to thank my husband, **Ido**, for his love, understanding, tenderness and encouragement during these tough and challenging times; I could have never completed this thesis without you.

Keren Kapach
Ben-Gurion University of the Negev
Beer Sheva, 2007

Table of Contents

List of Figures

List of Tables

Abstract

This work focuses on sensor fusion algorithms for mapping a mobile robot's environment. To function in unknown and unstructured environments a mobile robot must be equipped with several types of sensors, in order to better understand its surroundings and to overcome inaccurate or wrong information when sensors malfunction or fail. Sensor fusion deals with the synergetic combination of information produced by various sensors and is important in obtaining a more complete and accurate image on the phenomena being studied.

In this research, a previously developed sensor fusion framework was extended and revised. An additional physical sensor was added to the system, and the system was expanded to fuse data from this sensor as well. A new sensor fusion algorithm was developed.

Mapping the environment is important for several robotic tasks including exploration tasks and path planning. The binary grid map model is a common mapping technique and was employed in the previously developed sensor fusion framework. In this thesis, a non-binary grid map was used to indicate each cell’s certainty.

Down through the years, many sensor fusion algorithms for mapping the environment for mobile robots have been developed and implemented. Most of them require *a-priori* information about the sensor's performances or the surroundings conditions, which is hard and sometimes impossible to find in unstructured environments. In this research, a new adaptive sensor fusion algorithm was developed and implemented. The algorithm uses non-binary grid maps and on-line measures of each sensor’s performances. The measures developed give more weight in the fusion process to the better performing sensor. The algorithm also uses a new enhancement procedure that aims to improve the maps created by the different sensors. The enhancement procedure checks each cell’s neighbors and determines which cell indeed contains an obstacle and which cell should be treated as noise.

The algorithm was evaluated using a previously developed statistical evaluation method for evaluating the different fusion algorithms and choosing the best one. The method defines the experimental setup and procedure for testing the algorithms in various environmental conditions. Two evaluations were made. The first one aimed to test the extended sensor fusion system, while the second aimed to test the performance of the new sensor fusion algorithm in comparison to previously developed fusion algorithms.

Results from the first evaluation indicate that the best performing algorithm in the previous framework, an adaptive fuzzy logic algorithm, is the best performing algorithm in the extended framework as well. Results from the second evaluation indicate that the enhancement procedure did not affect the results at all and that in comparison to previously developed fusion algorithms, the new developed adaptive weighted algorithm is superior.

Keywords: Sensor fusion, mobile robots, mapping algorithms, grid map, adaptive algorithms, performance measure, evaluation.

This thesis is in part based on the following publications:

1. Kapach K. and Edan Y. 2007. Adaptive weighted average sensor fusion algorithms for mobile robots, IADIS International Conference Intelligent Systems and Agents: 43-50.
2. Kapach K. and Edan Y. 2007. Evaluation of grid-map sensor fusion mapping algorithms, IEEE International Conference on Systems, Man and Cybernetics.

1. Introduction

1.1 Problem description

To perform in unknown environments a mobile robot must build an accurate map that describes the robot's surroundings.
The first step in building a map is to choose the appropriate representation model [Cohen, 2005]. There are several different techniques for representing the environment of a mobile robot, including configuration space [Lozano-Perez, 1981]; generalized cones [Brooks, 1982]; spherical octree [Chen, 1987]; and occupancy grid maps [Moravec and Elfes, 1985; Stepan, 2005]. In this research the grid map paradigm was used since it is a simple and fast technique. In the grid map model the environment is divided into discrete cells, each containing a value that indicates whether the area represented by the cell is *Occupy* or *Empty*. There are several ways to fill the cells within occupancy grid map. One method is the binary grid map, where '0' represents *Empty* cell and '1' represents *Occupy* cell [Cohen, 2005]. Another method is a probabilistic grid map, where each cell contains the probability of being occupied or empty [Moravec and Elves, 1985; Stepan, 2005]. Another common approach for grid maps is the Certainty Values method (CV), where is cell is assigned a value that indicates the measure of confidence that an obstacle exists within that the cell area [Ribo and Pinz, 2001; Hong *et al.*, 2002]. Probabilities and CV's are usually derived from distribution functions based on the sensor's model.
An autonomous mobile robot must be equipped with several sensors in order to robustly sense its surroundings due to the different sensory characteristic and their inherent uncertainty in sensory information. To enhance the accuracy of the maps the environmental information received from multiple sensors must be merged. Sensor fusion deals with synergetic merging of information from several different physical sensors [Adibi and Gonzales, 1992].
Multisensor integration is the synergetic use of the information provided by multiple sensory devices to assist in the accomplishment of a task by the system. Multisensory fusion refers to any stage in the integration process where there is an actual combination of different sources of sensory information into one representational format. The distinction is made between multisensory integration and a more restricted notion of multi sensor fusion to separate the more general issues involved in the integration of multiple sensory devices at the system architecture and control level, from the more specific issues – possibly mathematical or statistical – involved in the actual combination (or fusion) of multisensory information [Luo and Kay, 1989].
To complete the mapping mission, it is necessary to choose a method for handling the multitude of sensors. In multi-sensor systems, the logical sensor paradigm is commonly used [Henderson and Shilcrat, 1984]. A *logical sensor* is an abstract definition of a sensor that can be used to provide a uniform framework for multisensory integration [Henderson and Shilcrat, 1984]. This approach enables to add sensors to the system without changing its whole concept. In this work several logical sensors were implemented.
The next step towards an accurate environment mapping is to choose the desired sensor fusion algorithm. Over the years, several sensor fusion methods and algorithms have been developed for mapping the environment of a mobile robot, e.g., weighted average [Belknap *et. al.*, 1986]; probabilistic [Harmon, 1986]; certainty factors [Belknap, 1986; Hanson *et al.,* 1988; Kamat, 1985]; and fuzzy logic [Huntsberger and Jayaramamurthy, 1987].

These algorithms and other different fusion methods are employed when assuming a-priori characteristics of the sensors (e.g., probabilities and standard deviation) including Bayesian [Sukumar *et al.*, 2007]; Neyman-Pearson [Thomopoulos *et. al.*, 1989]; Kalman filter [Zhu *et. al.*, 2006] and extended Kalman filter [Mirzaei *et al*., 2007].
In unconstructed and dynamic environment, which is a common mobile robot's environment, it is very hard and sometimes impossible to predict *a-priori* the sensors characteristics. Therefore, there is a need to estimate online the desirable characteristics while the system is in motion. Furthermore, a mobile robot operating in a dynamic system must respond online to environmental changes. This requires an algorithm that is able to adapt to the changing environment and sensory performances. Cohen [Cohen, 2005] developed an adaptive sensor fusion framework that fuses data from different physical sensors using different fusion algorithms, including an adaptive fuzzy logic algorithm.

This research is based on Cohen's work and intends to extend it.

1.2 Objectives

This research deals with grid-map based sensor fusion for mapping the environment of a mobile robot. The objectives of this research were to:

- Extend Cohen's work and evaluate its sensor fusion framework with a system that contains three physical sensors.
- Develop a new adaptive sensor fusion algorithm based on the extended fusion system.

1.3 Research significance and innovations

Cohen's work has been extended to fuse data from three physical sensors, and its performances were evaluated through the statistical evaluation procedure.
Cohen's binary grid map paradigm was extended to include non-binary grid maps. The maps was changed so occupied cells contains an integer value that indicates the number of time the sensor declares this cell as occupied, while in Cohen's work the binary grid map gave information that this cell is *Occupy* (by assigning the binary value '1'), regardless of the number of times this cells was declared as *Occupy*. This concept gives a lot of new information about the environment that was loss in the binary grid map concept. Using the non-binary grid map allows to give more weight to cells with higher values than the others, because most chances are that this cells indeed contains an obstacle instead of being marked as occupied due to noises of sensor's deviations. A new map enhancement procedure was developed and implemented. The enhancement procedure aims to improve maps accuracy and filter noises. The procedure checks each occupied cell's neighbors within the non-binary grid map and decided whether this cell indeed contains an obstacle or it should be treated as noise.
In addition, new performance measures were developed in order to evaluate online the sensors performances. These performance measures are used in the new adaptive sensor fusion algorithm. The new algorithm includes the use of non-binary grid map and the new type of performance measures, and allows the system online adaptation to changing environmental conditions. The new algorithm was proven to be superior to previous developed algorithms.

2. Scientific background

Chapter overview

This chapter reviews the literature of the relevant research topics including: sensor fusion applications, sensor fusion for autonomous mobile robots; different mapping algorithms for ultrasonic, camera and laser sensors; and sensor fusion evaluation methods.

2.1 Sensor fusion

Sensors are devices that collect data about the world around them. Sensors range from inexpensive cameras to earth observation satellites costing millions of dollars. In spite of this variety, all sensors have a few things in common. Every sensor device has a limited accuracy, and is a subject to the effect of some type of "noise", and will under some conditions function incorrectly [Brooks and Iyengar, 1998].

To overcome these drawbacks most applications employ multiple sensors. This requires the multitude of sensory data from multiple sensors to provide more reliable and accurate information [Luo *et al.*, 2002]. When done properly, sensor fusion combines input from many independent sources of limited accuracy and reliability to give information of know accuracy and proven reliability [Brooks and Iyengar, 1998]. The potential advantages in integrating and/or fusing information from multiple sensors are that information can be obtained more accurately, concerning features that are impossible to perceive with individual sensors, as well as in less time, and at a lesser cost [Adibi and Gonzales, 1992].

Sensor fusion is a rapidly evolving research area and requires interdisciplinary knowledge in control theory, signal processing, artificial intelligence, probability, statistics, *etc.* [Luo *et al.*, 2002]. In recent years, benefits of multisensor fusion have motivated research in a variety of application areas such as military applications, remote sensing, biomedical applications and robotics applications.

Military applications include the area of intelligence analysis, situation assessment, force command and control, avionics and electronic warfare [Luo *et al.*, 2002]. Filippidis and Martin [Filippidis *et al.*, 2000] presented a sensor fusion system that the detects surface land mines given multiple registered images of the mined area, obtained from a suite of visible to IR wavelength sensors. Carson [Carson *et al.*, 1996] fused data from radar and a set of sensors named identification friends-or-foe (IFF) sensors to improve capabilities of tracking and target identification system using two algorithms. The IFF sensor provides target height which is used to improve accuracy of 2D radar. The radar provides consistent and accurate bearing and range measurements which are not always available from the IFF sensor (*i.e.*, from hostile targets). The fusion of data obtained from these sensors provides data not obtainable by either sensor alone [Carson *et al.*, 1996]. Remote sensing applications include monitoring climate, environment, water sources, soil and agriculture as well as discovering natural sources and fighting the import of illegal drugs [Bell, 1995]. Solaiman [Solaiman *et al.*, 1999] applied fuzzy based multisensor fusion to land cover classification using ERS-1/JERS-1 SAR composites. Several sensor fusion applications were implemented in biomedical systems. Hernandez [Hernandez *et al.*, 1999] presents a multisensor multisource data fusion scheme to improve atrial (AA) and ventricular activity (VA) detection in critical care environments. The approach seeks to integrate, with the usual electrocardiogram (ECG) signals, complementary data from hemodynamic processes or from the esophageal ECG (EECG). Solaiman [Solaiman

et al., 1998] used fuzzy logic based fusion methods for feature extraction from ultrasound medical images, and results showed good quality detection.
Since robots are usually equipped with different sensors, multisensor integration and fusion techniques are suitable for areas of industrial robots such as motion planning, material handling, part fabrication, inspection and assembly [Luo and Kay, 1992]. Thomas [Thomas *et al.*, 2007] implemented a particle filter using sensor fusion for different assembly tasks. A sensor fusion system that fuses data from force and acceleration sensors to improve environmental force estimator in industrial robot was presented by [Garcia *et al.*, 2004]. Luo and Lin [Luo and Lin, 1996] have applied multisensor fusion techniques via an artificial neural network to fuse measurement data from force sensors, acoustic emission, accelerometer data and power signal to predict tool wear [Luo and Lin, 1996].
Groen [Groen *et al.*, 1986] describe a multisensor robotic assembly station equipped with vision, ultrasonic, tactile and force sensors. In operation, vision sensors are used to recognize different parts of the assemblies as they arrive in varying order and at undefined positions. Feedback information from the force sensors and the passive compliance of the robot's gripper are used for bolt insertion operations and to transport and place assembly housing on work spots. Final inspection is performed with vision sensors [Groen *et al.*, 1986].

The interaction between the sensors can be in three major ways: complementary, competitive or cooperative [Durrant-Whyte, 1998]. *Complementary* sensors do not depend on each other directly but can be merged to form a more complete picture of the environment. For example, a set of radar stations covering non-overlapping geographical regions. In this case, fusion implementation is easy since no conflicting information is presented. *Competitive* sensors provide equivalent information about the environment. For example, three identical radar stations covering overlapping geographical regions. In this case, a failure of one or two units can be tolerated. In this case, fusion must handle the case of conflicting reading. *Cooperative* sensors work together to derive information that neither sensor alone can provide. For example, two video cameras in stereo for three-dimensional vision. Fusion in this case cannot be approached as a general problem because it depends on details of the physical devices involved [Brooks and Iyengar, 1998].
In this research, the sensors operate complementary. Fusion in this case offers several advantages. First, fusion of redundant information can reduce overall uncertainty and thus increase the accuracy with which the features are perceived by the systems. Second, multiple sensors providing redundant information can also serve to increase reliability in case of sensor error or failure. In addition, complementary information from multiple sensors allows features in the environment to be perceived that would otherwise be impossible to acquire if we only used the information supplied from each individual sensor operating separately [Durrant-Whyte, 1988b; Luo *et al.*, 2002].

There are different levels of representation where fusion from multiple sensors can take place [Castellanos *et al.*, 2001; Cohen, 2005]:

- *Signal level* fusion refers to the combination of signals from a group of sensors to provide a signal that is usually of the same form as the original but with higher quality [Adibi and Gonzales, 1992].
- *Pixel level* fusion can be used to increase the information content associated with each pixel in an image formed through a combination of multiple images, *e.g.*, the fusion of a range image with a two-dimensional intensity image adds depth information to each pixel in the intensity image. This can be useful in the subsequent processing of the image [Adibi and Gonzales, 1992].

- *Feature level* fusion can be used both to increase the likelihood that a feature extracted from the information provided by a sensor actually corresponds to an important aspect of the environment and as means of creating additional composite features for the system to use [Adibi and Gonzales, 1992].
- *Symbol level* fusion allows the information from multiple sensors to be effectively used together at the highest level of abstraction. Symbol level fusion may be the only means by which sensory information can be fused if the sensors are very dissimilar or refer to different regions of the environment [Adibi and Gonzales, 1992].

Most of the sensors typically used in practice provide data that can be fused at one or more of these levels. In this research, we used pixel level fusion.

2.2 Sensor fusion for autonomous mobile robots

Mobile robots often operate in an unstructured and dynamic environments and are equipped with different types of sensors (*e.g.,* vision, laser and ultrasonic) to perform a wide variety of tasks (such as dead-reckoning or mapping). As a result from this diversity, sensor fusion methods are needed to translate the different sensory inputs into reliable information that is needed to complete tasks such as self-location, mapmaking, path computing, motion planning and execution. Hence, it becomes necessary to consider integrating or fusing data from a variety of different sensors so that an adequate amount of information from the environment can be quickly perceived [Adibi and Gonzales, 1992].

In implementing these tasks, different approaches have evolved for accumulating geometrical representations of the unknown environment for mobile robots [Cohen, 2005]. The representations used for robots operating in unknown or unstructured environments allow their world models to be dynamically modified and updated with uncertain sensor information [Abidi and Gonzales, 1992]. Among these representations there are spherical octrees [Chen, 1998]; configuration space [Loranzo-Perez, 1981]; generalized cones [Brooks, 1982]; Voroni diagrams [Miller, 1985] and polygon region model [Miller, 1985].

Multisensory information can be represented in a multi-dimensional grid of cells. Each cell in the grid corresponds to a region of space from which the sensor information is assumed to have originated [Adibi and Gonzales, 1992]. Discrete or continuous values can be used to map free and occupied areas within the environment. When continuous values are employed, the values represent the certainty of an obstacle being in the cell, with '0' and '1' values respectively implying an empty or an occupied cell [Moravec and Elfes, 1985]. Discrete values can be either binary [Cohen, 2005] where ‘0’ and ‘1’ represents *Empty* or *Occupy* cells, respectively. Another approach for discrete grip maps that was implemented in this research is filling the map with integer values that indicates the number of times the sensors declared each cell as occupied. The latter gives a lot more information regarding the cell’s state, and allows giving more weight to cells with higher values. The grid map model is attractive as a means of representing multisensory information because the data from each sensor are automatically in spatial correspondence, as long as each sensor can correctly map its data to the grid. It is also possible for fusion and other processing to take place within each cell before any further high level processing is required, a feature that is important in many real-time applications. A possible disadvantage of a grid representation that it usually requires a large amount of memory to store the grid [Adibi and Gonzales, 1992].

Many algorithms and methods have been commonly used in sensor fusion when mapping the environment for mobile robots (Table 1).

Table 1 Selected examples of multisensor mapping mobile robots

Mobile robot	**Sensors**	**Operating environment**	**World model representation**	**Fusion method**	**Reference**
Pioneer 2-AT	ultrasonic camera	Indoor	Grid map	adaptive fuzzy logic	[Cohen, 2005]
Magellen pro robot	Ultrasonic Laser	Manmade	Grid map	Rule based	[Lai, 2005]
Mobile platform	Sonar Infrared	Indoor corridor	Feature based map	Extended Kalman filter	[Vazquez, 2005]
Pioneer 2DX	Odometer Sonar	Indoor corridor	Grid map	Extended Kalman filter	[Ivanjko, 2005]
Nomad 200	Laser rangefinder	Unknown	Feature based map	Extended Kalman filter	[Costa *et al.*, 2006]
Jinny	Laser GPS	Unknown	Grid map	Rule based	[Chang *et al.*, 2006]
Kim's mobile robot	Sonar Infrared Camera	Indoor corridor	Grid map	hierarchical fusion: geometric, rule based and Bayesian	[Kim, 2006]
Pioneer II virtual mobile robot	A set of ultrasonic sensors	Small static virtual	3-D grid map	Rule based	[Li *et al.*, 2006]
Liu's mobile robot	Ultrasonic	Laboratory	Grid map	Consensus theory	[Liu *et al.*, 2006]
Nomad XR4000	Ultrasonic Laser	Laboratory	X-Y graph	Tangential regression	[Bank, 2007]
Wang's mobile robot	Ultrasonic	Laboratory	Feature based map	Rule based	[Wang *et al.*, 2007]

2.3 Sensor mapping algorithms

2.3.1 Ultrasonic sensors

Many mobile robots, including the one described in this research, use Polaroid ultrasonic sensors for environmental representation [Moravec and Elfes, 1984; Oriolo *et al.*, 1997; Toledo *et al.*, 2000; Karaman and Temeltas, 2004; Bank and Kampke, 2007]. One key characteristic of ultrasonic sensors is the propagation pattern, as shown in Figure 1. A "lobe" is defined as the angular range between the normal direction, *i.e.*, 0 and a zero of the first derivative of the plot, as can be seen in Figure 1. The primary lobe is about 15° wide. The secondary lobe is about 30° and the tertiary is about 45° wide [Cao and Borenstein, 2002].

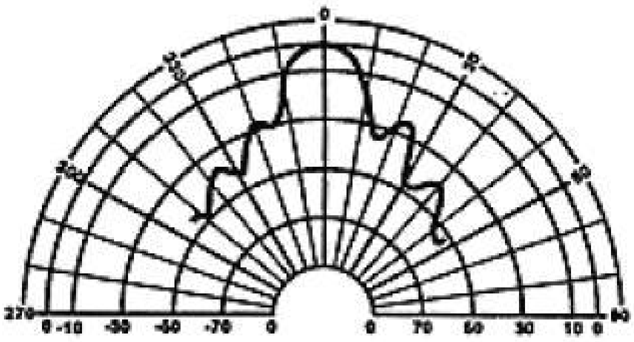


Figure 1 Propagation pattern for Polaroid ultrasonic sensors

(Adapted from ultrasonic sensor's data sheet)

For map-building task a large beam width is undesirable since it increases the uncertainty about the actual location of an obstacle. The result of this uncertainty is that obstacles tend to be represented in the map as larger than they really are [Cao and Borenstein, 2002]. Since the beam width of the ultrasonic sensor is limited, environmental mapping using ultrasonic sensors is commonly done by a set or a ring of sensors mounted on the robot in different angles. The multiple sensors also help locate the sensor in the horizontal direction.

The common occupancy grid map paradigm was first introduced by [Moravec and Elfes, 1984]. In this model, the sonar maps are two dimensional arrays of cells corresponding to a horizontal grid imposed on the area to be mapped. The grid has MXN cells, each of size deltaXdelta. The sonar reading in the final map has cell values in the range [-1, 1], where values less than 0 represent probably empty regions, exactly zero represents unknown occupancy, and greater than zero implies a probably occupied space. After preprocessing the incoming readings from the sonar to remove chronic errors by thresholding, averaging and clustering, the reading is projected into the correct cell using density functions according to the uncertainty regions in the sonar reading [Moravec and Elfes, 1984]. These probabilistic sensor-level sonar maps serve as the basis for a multilevel description of the robot's operating environment. These multiple descriptions are developed for various kinds of problem solving activities. Several dimensions of representation are defined: the *abstraction* axis, the *geographical* axis and the *resolution* axis [Elfes, 1987]. In the certainty grid model [Moravec, 1988]; the robot's work area is represented by a grid map. Each cell within the grid contains a certainty value (CV) that indicates the measure of confidence that an obstacle exists within the cell area. CV's are updated by a heuristic probability function that takes into account the conical field of view of the sonar [Elfes, 1987].

Histogram in-motion mapping (HIMM) is presented in [Borenstein and Koren, 1991]. The HIMM model uses a two dimensional Cartesian histogram grid for obstacle representation. Like the certainty grid concept, each cell in the histogram grid holds a certainty value that represents the confidence in the existence of an obstacle at that location. In this model, only the cell that corresponds to the measures distance and lies on the acoustic axis of the sensor is incremented. A probability distribution is obtained by continuously and rapidly sampling each sensor while the vehicle is moving. Thus, the same cell and its neighboring cells are repeatedly incremented. This results in a histogramic probability distribution, in which high certainty values are obtained in cells close to the actual location of the obstacle [Borenstein and Koren, 1991].

Fuzzy logic concepts are also used for robot perception as well as planning collision-free motions [Oriolo *et al.*, 1997; Karaman and Temeltas, 2004]. A map of the environment is defined as the fuzzy set if unsafe points, whose membership function quantifies the possibility for each point to belong to an obstacle. Each point in the map has two fuzzy values (*Occupy* and *Empty*) which are not complementary. The membership function derives from the ultrasonic sensor model and describes how the degree of certainty of the assertions '*Empty*' and '*Occupy*' varies in the map for a given range reading.

An occupancy grid map can also be built using artificial neural networks [Toledo *et al.*, 2000]. The neural network output supplies the probability of occupancy of the points considered as input. Thus, the occupancy estimated value is realized bearing in mind the different ultrasonic sensors readings at the same time. The neural network training is supervised and is carried out for a series of representative contours located into the sensors zone. Each contour is composed of a set of points chosen as function of the different response of the ultrasonic sensors in real environment. During training the output target for the network is 1 if the considered point is occupied and 0 if is not. Once trained, the neural network output generates a value between 0 and 1 representing the probability of occupancy.

Another common environment mapping is the segment-based map [Perez-Lorenzo et al, 2004]. The environment representation is modeled with features detected by sonar sensors. First, a local metric occupancy grid is built. Each cell in the map yields the occupancy probability of the corresponding region of the environment. The local grid acquired at instant t is combined with the metric map acquired at *t-1*. Second, each occupancy value in the local grid map is thresholded. Cells whose occupancy value is above a certain value are considered occupied cells, and all other cells are considered empty. Next, a Hough transform generation finds lines in the local grid map. These lines are extracted, and are used to find the main segments in the local metric map.

A probability based solution for map building is presented in [Kodagoda *et al.*, 2006]. The map is built using a single ultrasonic sensor, in contrast to most of the available map building systems. The sensor is mounted on a rotating shaft and discrete sonar observations taken at regular time intervals, by rotating the shaft in small step angles are incrementally merged into partial planes to produce a realistic representation of environment that is amenable to sonar localization. The probability model that has been implemented is based on the sonar sensor model in an effort to allocate probabilities based on a sonar reading. In this model the sensors' range area is divided into three regions based on the delay (*i.e.*, the alleged distance of the obstacle). The regions are: unknown, probably empty and probably occupied. The three regions have different equations based on which the probability of occupation is calculated [Kodagoda *et al.*, 2006].

High resolution ultrasonic imaging [Bank and Kampke, 2007] can be built using straight line representation. Sensor data is interpreted using tangential regression that considers sensor properties as well as physical reflection properties of ultrasound. This allows reliable detection and localization of straight lime segments which describe the boundary of geometric objects.

2.3.2 Machine vision

Visual information is the most powerful signal source of sensory information available to a system [Luo and Kay, 1989]. Many different types of non visual sensors are used in combination with vision sensors to compensate for some of the difficulties encountered in the machine processing of visual information [Luo and Kay, 1989; Miura *et al.*, 2002; Lu *et al.*, 2005; Tomono, 2005]. Tasks such as object recognition, feature extraction and SLAM can sometimes require the aid of additional types of sensors to approach the capabilities of a human using just visual information [Lua and Kay, 1989; Arras *et al.*, 2001; Davison and Murray, 2002; Kluge, 2003]. As small, cheap cameras have grown increasingly common, the software/hardware interfaces needed to grab camera frames has become easier to find and use [Wooden, 2006].

A lot of research has been done in the area practical implementation of visual-based sensing for performing several tasks in robotics system. Several are mentioned in this section.

An application for robot's localization using geometric features (vertical and horizontal lines) from a 360° laser rangefinder and a monocular vision system is presented in [Arras *et al.*, 2001]. Vertical lines are extracted from images of an embarked CCD camera. First, a specialized Sobel filter approximates the image gradient in the horizontal direction and the most relevant edge pixels are extracted and thinned by using a standard method. Next, the horizontal position of each edge pixel is corrected yielding a new position using a dedicated formula resulting from the camera model. Finally, columns with a predefined number of edge pixels are labeled as vertical lines.

Active visual sensing has been used for the exploration of sparse landmark information required in robot map-building [Davison and Murray, 2002]. The visual landmarks in use are features which are easily distinguishable from their surroundings, robustly associated with the scene geometry, viewpoint invariant, and seldom occluded. The robot points its two cameras in rather arbitrary directions and acquires features if regions of image interest are found. Features are detected using Harris corner detector [Harris and Stephens, 1988]. This rather rough collection of features is then refined naturally through the map maintenance steps into a landmark set which spans the robot's area of operation.

Detecting free regions in the robot's surrounding using stereo vision and visual tracking of persons is presented in [Tanaka *et al.*, 2003]. First, landmarks are found by using a correlation based stereo method proposed by [Faugeras *et al.*, 1993]. In this method, a dense depth map from a pair of images taken at different viewpoints is calculated through determining each pixel's correspondence to a landmark by correlation, and the depth is calculated based on the focal length of the cameras, baseline and the disparity. Points in the depth map that belong to the ceiling or floor are recognized as landmarks using their height, and are not regarded as obstacles. The remaining points are called landmark points L. Uncertainties in the correlation is analyzed according to the camera's model and some unreliable objects can be completely eliminated. Next, pixels that may correspond to a person are identified by subtracting the background image from the current frame, and each x-column is scanned in up direction until a pixel belonging to a person is found. Then, the pixel is regarded as a feature point and a navigation map based on the feature pointes is built [Tanaka *et al.*, 2003].

Locating and tracking a human in the vicinity of a robot using several sensors and a vision system is described in [Lu *et al.*, 2005]. The robot's surrounding is described using an occupancy grid, and the sensing objective is to determine the cell occupied by a human. The cell's size is sufficient for robot collision avoidance and preserving the human's safety. Two analog color cameras equipped with wide angle lenses and a frame grabber were installed on the ceiling, facing towards the center of the occupancy grid. The cameras are used individually so a single occlusion does not cause the vision system to fail. The image processing algorithm first captures a color image and converts it from RGM to HSI color space. Then, it finds the blob corresponding to the human's hardhat by thresholding and size filtering. The centroid of the blob is computed and based on the camera calibration is converted from the 2-D image coordinates to a line of sight in 3-D world coordinates. Then, the line is truncated into a line segment using the typical range of human height and is projected onto the occupancy grid indicating the cells potentially occupied by a human [Lu *et al.*, 2005].

Another vision based sensing for outdoor real-time robotics platform is described in [Wooden, 2006]. The robot is equipped with two pairs of color cameras mounted on its head and the plan is to plan a path to a known goal point using the grid based map building process. The map building process starts as a frame grabber captures a pair of images from two calibrated cameras. The images pass through a stereo library, which has knowledge of the relative physical geometry of the cameras as well as their intrinsic properties. The output is a depth map, *i.e.*, three dimensional terrain, in a local coordinate frame. Then, a first coordinate transformation step corrects the depth map for the pitch and roll of the robot, based on the inertial navigation unit, and a second coordinate transformation converts the depth map from the local frame to global, using the robot's yaw and current global position. Next, the terrain is described using a derivative operation on the small terrain map and the new information is incorporated into the robot's global map [Wooden, 2006].

2.3.3 Laser

A radial laser scanner is a device that measures distances to the objects in the environment intercepted by the laser beam [Reina and Gonzales, 1997]. Laser range scanners have become the sensor of choice because of their accuracy and wide availability [Amigoni *et al.*, 2006]. Three basic technologies are used in active laser ranging. Amplitude Modulation Continuous Wave (AM-CW) lasers use the difference of phase between emitted and received mean; Time-of-flight (TOF) lasers measure the travel time of a pulse; and Frequency Modulation Continuous Wave (FM-CW) use the frequency shift of a frequency modulated laser for measuring range [Hebert, 2000]. Among the available 2-D laser scanners, SICK LMS-200 has been broadly used, including in this research. This is a TOF laser sensor; a pulsed infrared laser beam is emitted and reflected from the object surface. The time between the transmission and the reception of the laser beam is used to measure the distance between the scanner and the object. The laser beam is reflected by a rotating mirror turning at 4500 rpm, which results in a fan-shaped scan pattern [Ye and Borenstein, 2002]. The third type of laser is the AM-CW lasers. These lasers are faster and perform best at close to medium range (*e.g.*, 50m range). They are typically more sensitive to ambient light, and therefore more suitable for indoor use. TOF scanners can perform at long range and are best suited for mobile robot application in outdoor settings; FM-CW sensors can be considerably more accurate, but at a cost of a more complex design and more brittle packaging. Most robotic applications use AM-CW or TOF sensors [Hebert, 2000].

Over the years, a lot of algorithms for mapping mobile robot's environments using laser scans were developed. Some of the methods are described in the following section.
Map building for a mobile robot equipped with a 2-D laser range finder is described in [Gonzales *et al.*, 1994]. The map consists of a set of short segments approximating the shape of the environment, and the update process involves a correspondence problem between segments from the current global map and segment from the local map obtained in each position. The advantage of using geometric descriptions over the more common grid-based representations is that line segments can be represented with few numbers and produce maps that are easier to use [Amigoni *et al.*, 2006]. The local map building is accomplished in four different steps: filtering scanned points that do not exhibit a local alignment within a tolerance, clustering the scan at points where the distance between successive points exceeds a threshold, clusters segmentation into pieces of scan suitable for a good linear fitting and line segment fitting where line segments are selected through best fitting all points within the above segmented groups [Amigoni *et al.*, 2006]. The final result of this process is a local map that composed of a set of line segments that approximate the contour of the surrounding obstacles [Gonzales *et al.*, 1994].

Geometrical primitives maps produced by a 2-D laser range finder is described in [Vandrope *et al.*, 1996]. Their map is composed of two different geometrical primitives. The first primitive is the line segment which is used to model all objects with a width exceeding 30 cm. The second primitive is a cluster which is used to measure points which lie close to each other

but do not pass the criteria for line extraction. The parameters on the geometrical primitives are provided with uncertainties depending on the uncertainty of the robot position estimate and the uncertainties of all measurements leading to this primitive. The map is dynamic, so objects which have removed from the real world are removed from the map as well.
Another geometric features laser map is presented in [Castellanos *et al.*, 2001]. First, the laser data is processed by a segmentation algorithm [Castellanos and Tardos, 1999]. Next, three types of features are extracted: segments, which are considered low-level features; corners and semi planes, which semantically upgrade the representation of the environment. Corners are found in the intersection of two consecutive segments whilst semi planes are found at the free endpoints of segments. Finally, a landmark is formed by each set of consecutive segments and their derived corners and semi planes.
In general, the environment around the mobile robot could be quite complex, and it may be composed of many obstacles such as chairs, boxes, the legs on the tables and so on. Thus, it is not practical to represent all those obstacles as either lines or clusters [Kwon and Lee, 1997]. To remedy this problem, another map model is suggested that represents the entire environment by a series of stochastic obstacle regions, with their own stochastic variables [Kwon and Lee, 1997]. Each stochastic region is represented by the mean, variance, covariance and by the number of scanning data used to determine the stochastic variable. Laser scans are mapped into a number of cluster regions. If the distance between two successive data points is smaller than 20cm, the points are denoted to be in the same cluster. New scans are matched and updated using rule-based algorithm according to the stochastic variables of each cluster [Kwon and Lee, 1997].
Grid map building using laser scans is described in [Patel *et al.*, 2005]. The purpose is to identify key unknown regions in the trajectory of the mobile robot and navigating the robot through it by using active laser sensing. The algorithm performs a look ahead search which picks the optimal direction to pan the laser according to a utility function. The vehicle builds a map as it senses its environment in the following way: Each laser scan is converted into a set of points in the global coordinate frame. The points in the scan are compared to their neighbors and labeled. Points with sudden steep changes in z values are labeled as obstacles, otherwise they are labeled as free. The points are placed into their corresponding grid cells and the status of the cell is updated [Patel *et al.*, 2005].
Most map building methods employed by mobile robots are based on the assumption that an estimate of robot poses can be obtained from odometry readings or from observing landmarks or other robots [Amigoni, 2006]. However, odometry data is often unreliable or does not exist for miniature robots. In addition, it is not possible to interrupt the mapping process and resume it at a later time without having to reset the initial poses of the robots [Amigoni, 2006]. Hence, a method for building segment-based maps without pose information was developed and detailed in [Amigoni, 2006]. Points returned from a 2-D laser scanner are approximated by line-segments denoted as a partial map. A line segment is represented by its two end points in the reference frame of the map. Range data can be collected by single or different robots. Two partial maps are integrated and a set of partial maps is merged in order to build a global map using a matching process. In this method, it is indifferent if the scans are collected during a single session or multiple sessions, by multiple robots or a single robot. Robots can be added or removed at any time, and they do not need to know their own position [Amigoni, 2006].

2.4 Sensor fusion evaluation

Along the years, many sensor fusion algorithms have been developed and implemented [Luo and Kay, 1989]. Each algorithm has its advantages and disadvantages and therefore a method for comparing performances is needed. Performance evaluation of sensor fusion in most cases involves real environment experiments, which is problematic in dynamic and unstructured environment, since it is impossible to repeat experiments under identical conditions. A second approach for performance evaluation uses theoretical analysis, but it is also hard to implement since it is usually difficult to characterize sensory performances in unstructured environments [Cohen, 2005].

As a result of the difficulties, there is a need to find a quantitative comparison of algorithms to identify the most effective fusion technique. As the method of the evaluation can have a significant effect on the validity of the evaluation, the evaluation approach should be taken with care. Among the characteristics that we would expect such a method to provide are: the evaluation should not be biased in favor of specific systems and should ideally be independent of the data used. In addition, the evaluation should be objective but in broad agreement with a subjective assessment. The evaluation should give an overall indication of system performances and should not be significantly affected by exceptional results [Schwering *et al.*, 2002].

An example of performance measures for fusion algorithms is available for landmine detection problems. The probability of detection is plotted against the probability of false alarm for an adjustable threshold and creates the ROC (receiver operator characteristics curve). Based on this ROC curve, the minimal risk can be calculated for specific cost functions [Cramer *et al.*, 2001; Schwering *et al.*, 2002]. ROC curves are also used to diagnose performances in radiologic imaging using statistical methods [Metz, 1986].

Several performance measures for comparing and quantitative evaluations of sensor fusion mapping algorithms were developed. A *fitness factor* for comparing fused grid map generated from three fusion algorithms is presented in [HoseinNezhad, 2002]. The factor is calculated for each map, and represents the similarity of the fused map to the corresponding true map of the simulated environment by calculating the difference in the occupancy probability between two corresponding cells in the obstacle's perimeters [HoseinNezhad, 2002].
Another performance measure is the *Score* measure [Martin and Moravec, 1996] which is defined as the *match* of a map to an a-priori ideal map. The *match* between two maps (for a given relative displacement) is the log of the probability that the maps represent the same world. However, this method does not describe the quality of the grid with respect to using the grid for planning, and for that reason a *safety measure* was introduced [Stepan *et al.*, 2005]. The safety measure can be computed from the planned path in the fused map and the pattern grid and allows selecting the best fusion method for a specific environment.

Statistical measures usually require a-priori assumptions such as on the data distribution [Faceli *et al.*, 2004]. Such a-priori assumptions often lack validation in real world situation and therefore are not accurate enough. A statistical evaluation method for comparing sensor fusion mapping algorithm that does not require *a-priori* information about data distribution or sensors performances was developed in [Cohen, 2005]. This method defines the experimental design and statistical analysis procedure and was implemented in this research (chapter 7).

2.5 Cohen's Work [Cohen, 2005]

2.5.1 General

This research is based on Cohen's PhD thesis and aimed to extend the previous analysis.
Cohen's system was developed based on three basic concepts: *logical sensors, grid map* paradigm and *performance measures*.

The *logical sensor* paradigm used to provide a uniform framework for multisensory integration [Henderon and Shilcrat, 1984]. This approach enables to add sensors to the system without changing its whole concept.

The *grid map* paradigm was chosen to present the environment perception due to its simple implementation and use [Moravec and Elfes, 1985]. Using the grid map representation, the environment is divided into a fixed size discrete grid. Each grid cell is assigned a value that indicates if that location is occupied by an obstacle or not. Cohen in his work used binary grid maps where each cell is assigned either '1' to represent an *'Occupy'* cell or '0' to represent an *'Empty'* Cell. The *performance measures* quantify the difference between two grid maps and uses the difference between binary decisions about the cell's condition in the grid maps. Cohen defined two types of performance measures, to quantify the logical sensors and the sensor fusion algoriths performances.

Sensor fusion algorithms are used to merge or combine the logical sensor's grid maps into one grid map, using different algorithms. Cohen used two types of fusion algorithms: logical and adaptive,which are elaborated below.

In order to evaluate the fusion algorithms performances, Cohen used mobile robot experiments and a statistical evaluation method. The latter aimed to choose the best performing algorithm. The statistical evaluation method defines the experimental setup and makes sure that the results are not specific for one certain data set.

This section describes Cohen's major development basics that were the foundations for the development in this work.

2.5.2 Information flow

The system includes *N* logical sensors representing *k* physical sensors. The logical sensors work asynchronously. The schematic description of the information flow is presented in Figure 2. At each time step *t*, the i^{th} logical sensor maps the environment using the physical sensor readings and creates a local observation grid map (LOGM), denoted by y_i^t $i = 1, 2, \cdots, N$. Let c_i and d_i be the local observation grid map dimensions. $y_i^t \in \square^{c_i \times d_i}$ with values from the range $\{0, 1, \ldots\ldots\ldots, r_i\}$ for each cell of that map. The values indicate the number of times each cell was sampled by the physical sensor.

The system transfers each sensor's LOGM into a local grid map (LGM), denoted by $u_i^t, i = 1, 2, \cdots, N$. Let *c* and *d* be the local grid map dimension. $u_i^t \in \mathrm{N}^{cxd}$ with values from the range $\{0, 1, \ldots\ldots\ldots, r_i\}$ for each cell of that map. The LGM dimensions are identical for all logical sensors.

There are two types of fusion algorithms: **logical** and **adaptive**. The algorithms differ in the memory and feedback properties; the adaptive algorithms use performance measures (explained below) in the fusion process while the logical algorithms do not.

For **logical** algorithms (Figure 2):

The LGM reaches the fusion center, where it yields the fused grid map (FGM) $u_0^t \in \mathrm{N}^{cxd}$, based on all the LGM $u^t, u^t \in (u_1^t, u_2^t, \cdots, u_N^t)$, using the fusion rule $f(\cdot)$ as follows:

$$u_0^t = f(\boldsymbol{u}^t) \qquad [1]$$

For **Adaptive** algorithms (Figure 2):

The performance measures of the logical sensors are calculated based on the previous local grid maps $\boldsymbol{u^{t\text{-}1}}$, $\boldsymbol{u^{t\text{-}1}} = (u_1^{t-1}, u_2^{t-1}, ..., u_N^{t-1})$ of the logical sensors and the previous fused grid map defined as u_0^{t-1}. The performance measures are denoted as p_i^{t-1}, where $i = 1, 2, \cdots, N$. The calculation of the performance measure depends on the fusion rule in the fusion center. A detailed description on the performance measures that were used in each adaptive algorithm can be found in sections 2.5.4.2 and 5.2. An average value of the logical sensor performance measures $\boldsymbol{p^{t\text{-}1,t\text{-}2}}$, $\boldsymbol{p^{t\text{-}1,t\text{-}2}} = (p_1^{t-1,t-2}, p_2^{t-1,t-2}, ..., p_N^{t-1,t-2})$ is calculated, based on $\boldsymbol{p^{t\text{-}1}}$ and $\boldsymbol{p^{t\text{-}2}}$, where $p_i^{t-1,t-2} = \dfrac{p_i^{t-1} + p_i^{t-2}}{2}$, $i = 1, 2, ..., N$.

Both the local grid maps $\boldsymbol{u^t}$ and an average value of the logical sensor performance measures $\boldsymbol{p^{t\text{-}1,t\text{-}2}}$ are transmitted to the fusion center. At the fusion center, based on all local grid maps $\boldsymbol{u^t}$ and the average value of the logical sensor performance measures $\boldsymbol{p^{t\text{-}1,t\text{-}2}}$ the sensor fusion algorithm yields the fused grid map u_0^t at time step *t*, using the decision rule $f(.)$ as follows:

$$u_0^t = f\left(\boldsymbol{u^t}, \boldsymbol{p^{t\text{-}1,t\text{-}2}}\right) \qquad [2]$$

The fused grid map, u_0^t is fed back to all logical sensors ($\boldsymbol{u^t}$) to calculate the new performance measures ($\boldsymbol{p^t}$).

Two types of adaptive algorithms were employed. The first type is the adaptive fuzzy logic (denoted as AFL, explained below) developed, which uses the logical sensor's performance measures as fuzzy variables with three fuzzy sets (low, average and high) using trapezoid membership function and If-Then rules to decide about the cell's condition (*'Occupy'* or *'Empty'*).

The following information flow is identical to both logical and adaptive algorithms.
At each time step *t*, a virtual global grid map (VGGM), denoted by $Z_0^t, Z_0^t \in \mathbb{R}^{a \times b}$, expands the size of the fused grid map u_0^t from $c \times d$ to $a \times b$, which is the full size. This is done by assigning zero values to all cells of the virtual global grid map Z_0^t, except those which appear in u_0^t (their values are as in the u_0^t map).

All the VGGM's are placed in a new map, the global grid map (GGM), denoted by $Z, Z \in \mathbb{R}^{a \times b}$. The VGGM's are places in the GGM according to the robot's new position along the path. The GGM is an $a \times b$ matrix and is the output of the entire mapping process, that represents the whole environment mapping along the robot's path.

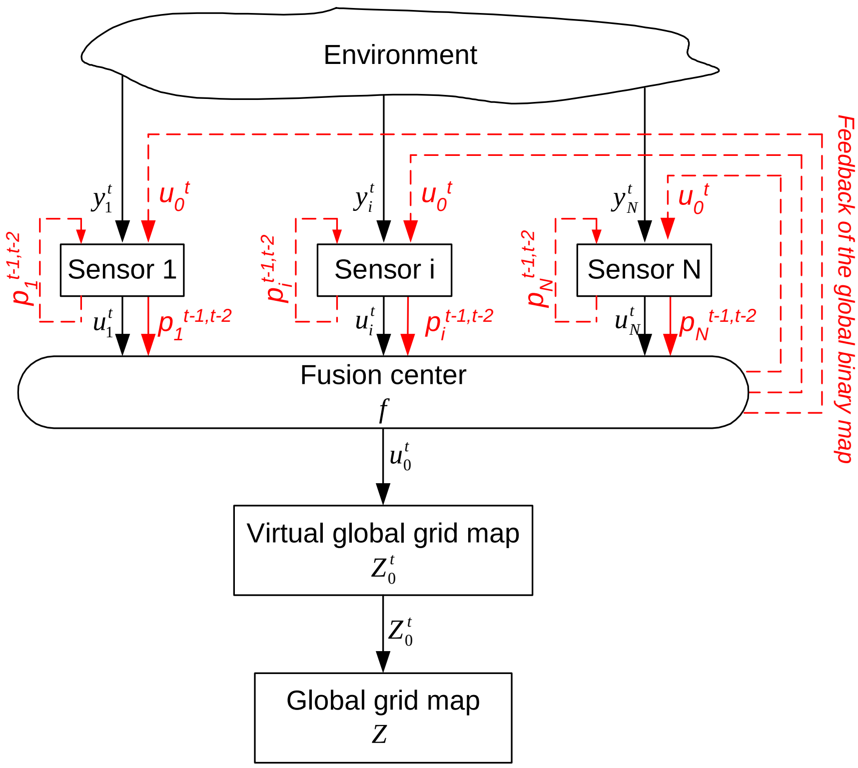


Figure 2 Information flow at time t – logical and adaptive algorithms

Data in black represents the logical algorithms information flow
Data in red with the data in black represent the adaptive algorithms information flow.

2.5.3 Performance measures

Cohen's performance measures use the binary decisions about the cell's condition in the grid maps. Since the cell's condition is a binary value (a positive value indicates *'Occupy'* and '0' indicates *'Empty'*), there are four logical conditions for the difference between the two maps. the performance measures are defined as the summation over all cells of the four logical conditions: *Occupy – Occupy*, *Empty – Empty*, *Occupy – Empty* and *Empty – Occupy*.

Cohen's performance measures are used in two cases. In each case the calculation process is slightly different. In the first case, they are used in the adaptive fusion algorithms, to quantify the difference between the logical sensor's maps and the fused map received as an output from the fusion algorithm. In the second case, they are used in the sensor fusion evaluation process, to quantify the difference between the sensor fusion's map and the original truth map.
The former measures are denoted in Cohen's work as 'logical sensor performance measures' and the latter are denoted as 'sensor fusion algorithm performance measures'.

2.5.3.1 Logical sensor performance measures

The logical sensor performance measures are measured by comparing each cell for each logical sensor's local grid map (LGM, u_i^t) with the corresponding cell in the adaptive fuzzy logic fused grid map (FGM, u_0^t) for time sample t. each logical sensor has four performance measures ($p_i^t \in \mathbb{R}^4, \forall i = 1, 2, \cdots, N$ where N is the total number of logical sensors). Since the cell's condition is a binary value (a positive value indicates *'Occupy'* and '0' indicates *'Empty'*), there are four logical conditions for the difference between the two maps.

Let:

$$LGM_{jk}=\begin{cases}0 & Empty\ jk\ cell\ in\ LGM\\ >0 & Occupied\ jk\ cell\ in\ LGM\end{cases}$$
$$FGM_{jk}=\begin{cases}0 & Empty\ jk\ cell\ in\ FGM\\ >0 & Occupied\ jk\ cell\ in\ FGM\end{cases} \qquad [3]$$

Where:

$LGM^t(i)_{jk}$ are cells in the sensor's local grid map (u_i^t) and

FGM^t_{jk} are cells in the fused grid map (u_0^t) corresponding to the j row and k column

Then:

$$OO^t(i)=\begin{cases}\dfrac{\sum_j\sum_k\left(LGM^t(i)_{jk}\cdot FGM^t_{jk}\right)}{\sum_j\sum_k FGM^t_{jk}} & \sum_j\sum_k FGM^t_{jk}>0\\ E_{LGM}E_{FGM}{}^t(i) & else\end{cases} \qquad [4]$$

$$EE^t(i)=\begin{cases}\dfrac{\sum_j\sum_k\left(\left(1-LGM^t(i)_{jk}\right)\cdot\left(1-FGM^t_{jk}\right)\right)}{\sum_j\sum_k\left(1-FGM^t_{jk}\right)} & \sum_j\sum_k\left(1-FGM^t_{jk}\right)>0\\ O_{LGM}O_{FGM}{}^t(i) & else\end{cases} \qquad [5]$$

$$OE^t(i)=\begin{cases}\dfrac{\sum_j\sum_k\left(LGM^t(i)_{jk}\cdot\left(1-FGM^t_{jk}\right)\right)}{\sum_j\sum_k\left(1-FGM^t_{jk}\right)} & \sum_j\sum_k\left(1-FGM^t_{jk}\right)>0\\ 1-O_{LGM}O_{FGM}{}^t(i) & else\end{cases} \qquad [6]$$

$$EO^t(i)=\begin{cases}\dfrac{\sum_j\sum_k\left(\left(1-LGM^t(i)_{jk}\right)\cdot FGM^t_{jk}\right)}{\sum_j\sum_k FGM^t_{jk}} & \sum_j\sum_k FGM^t_{jk}>0\\ 1-E_{LGM}E_{FGM}{}^t(i) & else\end{cases} \qquad [7]$$

Note: for the calculation process, the maps (i.e., FGM^t and LGM^t) were transformed into binary maps, which means that positive values were changed to '1'.

The vector $p_i^t=\left(OO^t(i),EE^t(i),OE^t(i),EO^t(i)\right)$ is calculated using [4]-[7] to each of the logical sensors separately in every time step t, where i is the i^{th} logical sensors, $i=1,2,\cdots,N$ and N is the total number of logical sensor.

In order to combine the four measures, an additional united measure ($UM^t(i)$) was defined by Cohen:

$$UM^t(i)=OO^t(i)-OE^t(i)=EE^t(i)-EO^t(i) \qquad [8]$$

2.5.3.2 Sensor fusion performance measures

The sensor fusion performance measures are calculated by comparing each cell of the original truth map (ORG, $ORG \in (0,1)^{a \times b}$), with the corresponding cell on the global grid map (GGM) which is defined as Z. The performance measures use the binary decisions about the cell's condition in the grid maps. the values of the sensor performance measures OO, EE, OE and EO were calculated by multiplying the relevant variables by the coefficients defined.

Table 2 Coefficients for calculating the sensor fusion performance measures

Occupy$_{\text{Coefficient}}$
$$\begin{aligned} &If\ (Occupy_{ORG} = Occupy_{GGM} = 0)\ Then \\ &\quad Occupy_{Coefficient} = Empty_{Coefficient} \\ &elseif \quad 0 \le \frac{Occupy_{GGM}}{Occupy_{ORG}} \le 1\ Then \\ &\quad Occupy_{Coefficient} = \frac{Occupy_{GGM}}{Occupy_{ORG}} \\ &else \\ &\quad Occupy_{Coefficient} = \frac{Occupy_{ORG}}{Occupy_{GGM}} \end{aligned}$$
Empty$_{\text{Coefficient}}$
$$\begin{aligned} &If\ (Occupy_{ORG} = Occupy_{GGM} = a \cdot b)\ Then \\ &\quad Empty_{Coefficient} = Occupy_{Coefficient} \\ &elseif \quad 0 \le \frac{a \cdot b - Occupy_{GGM}}{a \cdot b - Occupy_{ORG}} \le 1\ Then \\ &\quad Empty_{Coefficient} = \frac{a \cdot b - Occupy_{GGM}}{a \cdot b - Occupy_{ORG}} \\ &else \\ &\quad Empty_{Coefficient} = \frac{a \cdot b - Occupy_{ORG}}{a \cdot b - Occupy_{GGM}} \end{aligned}$$
Where a and b are defined as the global grid map's dimensions.

<u>Let:</u>

$$GGM_{jk} = \begin{cases} 0 & Empty\ jk\ cell\ in\ GGM \\ > 0 & Occupied\ jk\ cell\ in\ GGM \end{cases}$$
$$ORG_{jk} = \begin{cases} 0 & Empty\ jk\ cell\ in\ ORG \\ > 0 & Occupied\ jk\ cell\ in\ ORG \end{cases} \qquad [9]$$

Where:
GGM_{jk} are cells in the global grid map (Z) and
ORG_{jk} are cells in the original map (ORG) corresponding to the j row and k column

Then:

$$O_{GGM}O_{ORG} = \begin{cases} \dfrac{\sum_j \sum_k \left(GGM_{jk} \cdot ORG_{jk}\right)}{\sum_j \sum_k ORG_{jk}} & \sum_j \sum_k ORG_{jk} > 0 \\ E_{GGM}E_{ORG} & else \end{cases} \quad [10]$$

$$E_{GGM}E_{ORG} = \begin{cases} \dfrac{\sum_j \sum_k \left(\left(1-GGM_{jk}\right)\cdot\left(1-ORG_{jk}\right)\right)}{\sum_j \sum_k \left(1-ORG_{jk}\right)} & \sum_j \sum_k \left(1-ORG_{jk}\right) > 0 \\ O_{GGM}O_{ORG} & else \end{cases} \quad [11]$$

$$O_{GGM}E_{ORG} = \begin{cases} \dfrac{\sum_j \sum_k \left(GGM_{jk} \cdot \left(1-ORG_{jk}\right)\right)}{\sum_j \sum_k \left(1-ORG_{jk}\right)} & \sum_j \sum_k \left(1-ORG_{jk}\right) > 0 \\ 1-O_{GGM}O_{ORG} & else \end{cases}$$

$$E_{GGM}O_{ORG} = \begin{cases} \dfrac{\sum_j \sum_k \left(\left(1-GGM_{jk}\right)\cdot ORG_{jk}\right)}{\sum_j \sum_k ORG_{jk}} & \sum_j \sum_k ORG_{jk} > 0 \\ 1-E_{GGM}E_{ORG} & else \end{cases} \quad [12]$$

Note: for the calculation process, the maps (i.e., FGM^t and LGM^t) were transformed into binary maps, which means that positive values were changed to '1'.

And

$$OO = Occupy_{Coefficient} \cdot \left[O_{GGM}O_{ORG}\right] \quad [13]$$

$$EE = Empty_{Coefficient} \cdot \left[E_{GGM}E_{ORG}\right] \quad [14]$$

$$OE = (1 - Empty_{Coefficient}) \cdot \left[O_{GGM}E_{ORG}\right] \quad [15]$$

$$EO = (1 - Occupy_{Coefficient}) \cdot \left[E_{GGM}O_{ORG}\right] \quad [16]$$

2.5.4 Sensor fusion algorithms

In his work, Cohen defined two types of sensor fusion algorithms. The first type consists of *logical* algorithms in which the logical sensor distinguishes between two basic states, *Occupy* and *Empty*. The second type use the performance of the logical sensors in the fusion, and are denoted as *adaptive* algorithms. These algorithms are considered as algorithms that have feedback and memory. The adaptive algorithm uses the fuzzy logic theorem, as detailed below.

2.5.4.1 Logical algorithms

Three logical sensor fusion algorithms were evaluated. These algorithms present different versions of *Identify the obstacle by at least n logical sensors*: Logical OR (n=1), MOST (n>N/2) and logical AND (n=N), where *N* is the total number of logical sensors in the system [Cohen, 2005; Blum *et al.*, 1997; Klein, 1993].

The inputs are the local grid map (*i.e.,* u_i^t) and their output is the fused grid map (*i.e.,* u_0^t).

2.5.4.2 Adaptive fuzzy logic algorithm

The algorithm's inputs are all the logical sensor's local grid maps (*i.e.*, u^t) and the average value of the logical sensor performance measures (*i.e.*, $p^{t-1,t-2}$). The output is a fused grid map (*i.e.*, u_0^t) of the fused information.

The Adaptive fuzzy logic (denoted as AFL) algorithm that was evaluated was the algorithm that achieved best performances according to Cohen's evaluation (denoted as 1010 in [Cohen, 2005]).

The adaptive fuzzy logic algorithm uses Cohen's logical sensors performance measures as fuzzy variables with three fuzzy sets: High, Average and Low. Each fuzzy set member is associated with a trapezoid membership function. The membership function evaluates the degree of membership of each variable value of the respective fuzzy set member. Fuzzy sets values and membership function of the fuzzy variables are presented in Table 3.

Table 3 Fuzzy set values of the fuzzy variables

Fuzzy variable	**Fuzzy sets**		
	Low	**Avg.**	**High**
$O_{LGM}O_{FGM}{}^{t-1,t-2}(i)$	0,0,0.3,0.45	0.4,0.45,0.55,0.6	0.55,0.7,1,1
$E_{LGM}E_{FGM}{}^{t-1,t-2}(i)$	0,0,0.3,0.45	0.4,0.45,0.55,0.6	0.55,0.7,1,1
$O_{LGM}E_{FGM}{}^{t-1,t-2}(i)$	0,0,0.3,0.45	0.4,0.45,0.55,0.6	0.55,0.7,1,1
$E_{LGM}O_{FGM}{}^{t-1,t-2}(i)$	0,0,0.3,0.45	0.4,0.45,0.55,0.6	0.55,0.7,1,1

For each logical sensor at every time stamp *t*, two fuzzy output variables are calculated: $Occupy_i^t$ and $Empty_i^t$, where $i = 1, 2, \cdots, N$ and *N* is the total number of the logical sensors. These output fuzzy variables also have three fuzzy sets: High, Average and Low. Each fuzzy set member is associated with a trapezoid membership function. The fuzzy sets values of the output fuzzy variables are presented in Table 4.

Table 4 Fuzzy sets values of the fuzzy output variables

Fuzzy variable	**Fuzzy sets**		
	Low	**Avg.**	**High**
Occupy	0,0,0.3,0.45	0.4,0.45,0.55,0.6	0.55,0.7,1,1
Empty	0,0,0.3,0.45	0.4,0.45,0.55,0.6	0.55,0.7,1,1

The fuzzy output variables are calculated using twelve **If-Then** rules presented in Table 5.

Table 5 If-Then rules

	Fuzzy variables input				**Fuzzy variables output**	
Rule	$O_{LGM}O_{FGM}{}^{t-1,t-2}(i)$	$E_{LGM}E_{FGM}{}^{t-1,t-2}(i)$	$O_{LGM}E_{FGM}{}^{t-1,t-2}(i)$	$E_{LGM}O_{FGM}{}^{t-1,t-2}(i)$	Occupy	Empty
1	High				High	
2	Avg.				Avg.	
3	Low				Low	
4			High		Low	
5			Avg.		Avg.	
6			Low		High	
7		High				High
8		Avg.				Avg.
9		Low				Low
10				High		Low
11				Avg.		Avg.

12				Low		High

The rules are defuzzified using the Mamdani inference with centroid method [Mamdani and Assilian, 1975; Kosko, 1992; Zadeh, 1978] and are evaluated to determine the final value of the $Occupy_i^t$ and $Empty_i^t$ final value.

The fused map cells are binary, where '1' indicates that the cell is 'Occupy' and '0' indicated that the cell is 'Empty'. The decision rule for the fused map cell's is based on the summation of the logical sensor's $Occupy_i^t$ and $Empty_i^t$ final values. For all the corresponding cells in the logical sensor's map that are '0', their $Empty_i^t$ values are summed. For all the corresponding cells in the logical sensor's map that are '0', their $Occupy_i^t$ values are summed. If the *Occupy* sum is greater than the *Empty* sum, the cell in the fused map is set to '1', otherwise – '0'. The pseudo-code for fused map decision rule is presented in Figure 3:

Adaptive fuzzy logic decision rule
$for\ x = 1 : MapSizeX$ $for\ y = 1 : MapSizeY$ $for\ i = 1 : N$ $if\ u_i^t(x, y) = 0$ $Empty^t = Empty^t + Empty_i^t$ $else$ $Occupy^t = Occupy^t + Occupy_i^t\}$ $if\ Occupy^t > Empty^t$ $u_0^t(x, y) = 1$ $else$ $u_0^t(x, y) = 0$

Figure 3 Pseudo code for fused map decision rule

2.5.5 Mobile robot experiment

Cohen's experiment consisted of a mobile robot (Pioneer 2-AT). The robot was equipped with an array of 16 ultrasonic sensors on the robot's front and back panel (eight sensors on each panel), and a PTZ SONY CCD camera. All sensors were used to scan the area in front of the robot; therefore only six ultrasonic sensors from the front of the robot were used. The robot's specifications and parameters are detailed in Appendix I.

During the experiment, the robot moved forward at a constant velocity (10 cm/sec) along a 574 cm X 240 cm path in a controlled laboratory environment. As it moved, it mapped the area in front of it. This area consisted of a black path with five obstacles in a fixed location. To increase disagreement between logical sensors, two types of decoy obstacles were set along the robot's path. These decoys were made of light brown rug. The first type of decoy was less than 6 cm. in width and length; the size of the second type was around 25 cm. Decoys locations were randomly changed between repetitions. The obstacles and decoys were not always noticeable to all logical sensors because of differences in the algorithms as in the color, size and structure of the decoys themselves. These differences caused the logical sensors to disagree.

2.5.6 <u>Statistical evaluation method</u>

2.5.6.1 General

To evaluate the algorithm's performances, several different experiments were performed. The experiments differ by changes in the input and in the sensory conditions. Malfunctions were created artificially by setting logical sensors to empty, full and shifting positions by a constant value. Each experiment is performed R times (called repetitions), under the same environmental and sensory conditions. Of course, there are some deviations from one repetition to another, due to changes in the lighting conditions (day/night), temperature, shadows, etc. The algorithms performances are quantified using Cohen's sensor fusion algorithm performance measures, as detailed in section 2.5.3.2.

The preliminary step in the evaluation process is to ensure that the experiments are different enough and that there are enough repetitions. The number of repetitions is defined by the statistical parameters (mean and standard deviation) of the performance measures as defined in [Cohen, 2005].

A statistical analysis determines the best performing algorithm. The procedure evaluates the performance measures that were calculated for each algorithm's map in each experiment and repetition. Non parametrical statistics were used since the data was very scattered. The statistical analysis includes three stages. The first one is the *Friedman's test*, which tests that the algorithms performances are different. The test is followed by a *multiple comparisons procedure*, which divides the algorithms into homogenous subgroups. The third and last step is the *sign test* that picks the best performing algorithm.

This section details the evaluation method procedures. After confirming that the experiments are different from each other and repetitions are similar enough by calculating the volume of overlap region [Tin and Mitra, 2002], the number of repetitions required is calculated using the statistical parameters (*i.e.*, mean and standard deviation) of the performance measures. This step is followed by a statistical analysis that includes three non-parametric tests. The evaluation process was programmed in MATLAB, the code is detailed in
Appendix VII.

2.5.7 <u>Different experiments</u>

For each experiment between every two repetitions, each logical sensor's map from one repetition is subtracted from all other logical sensor maps from the other repetitions and saved as an absolute value. For example, LS1 map from experiment 1, repetition 1 is compared to LS1 map from experiment 2 and all its repetitions, experiment 3 and all its repetitions and so on. The number of cells different than '0' (signed cells) is saved for each comparison. The total number of subtracted maps (*NExp.*) is presented in equation [17]. For each comparison, the worst difference of all logical sensors is saved.

$$N_{Exp.} = (NumOfLS) \cdot (NumOfRep)^2 \cdot \binom{NumOfExp}{2} \qquad [17]$$

2.5.8 <u>Similar repetitions</u>

For each experiment, each logical sensor's map is subtracted from all its repetitions in pairs and saved as an absolute value. For example, LS1 map from experiment 1 is subtracted from LS1 maps from all other repetitions. This is conducted for all LS, and the number of cells different than '0' (signed cells) is saved for each comparison. The total number of comparisons (*NRep.*) is presented in equation [18]. For each comparison, the worst difference is saved, *e.g.*, the maximum number of signed cells.

$$N_{Rep.} = (NumOfLS) \cdot (NumOfExp) \cdot \binom{NumOfRep}{2} \qquad [18]$$

2.5.9 Volume of overlap region

This measure is an indicator that the experiments and repetitions are indeed different. This measure evaluates the overlap of two populations (*e.g.*, experiments and repetitions) and should be as negative as possible [Tin and Mitra, 2002]. If the volume is not negative, the experiments are not different enough and more experiments need to be performed. The volume is calculated using the minimum and maximum number of signed cells from all the comparisons between the experiments and repetitions, as shown in equation [19].

$$VOLR = \frac{MIN(\max(Exp.), \max(Rep.)) - MAX(\min(Exp.), \min(Rep.))}{MAX(\max(Exp.), \max(Rep.)) - MIN(\min(Exp.), \min(Rep.))} \qquad [19]$$

2.5.10 Number of repetitions

The number of repetitions is based on a t-test detailed in [Cohen, 2005] and is calculated for chosen values of α and β. For each performance measure, the number of repetitions was calculated, and the final number was taken as the maximum number from all the performance measures. The standard deviation (S) for each performance measure was taken as the upper bound of the standard deviation for this performance measure from all the experiments. Δ is the minimum difference to be detected and is taken also for each performance measure separately. In case the number of repetitions that were conducted is not sufficient, more repetitions are required.

2.5.11 Performance measure calculation and grouping

After a confirmation that the experiments are indeed different and enough repetitions were conducted, type I performance measures are calculated. Type I performance measures are calculated using each algorithm's global grid map (GGM) denoted by *Z* in section 2.5.1, and quantify the difference between the real world map and each algorithm's GGM using equations [13]-[20]. For each experiment, all repetitions values for each performance measure are grouped together.

2.5.12 Statistical analysis

The statistical analysis includes three non-parametric tests that aim to find the best performing algorithm. The first stage is the *Friedman's test* [Hollander and Wolfe, 1973], that checks whether the algorithms performances are considered different. Friedman's test is performed separately for each performance measure in every experiment. In this test, the algorithms are ranked from the least (rank=1) to the largest (rank=4) for every repetition. The test statistic uses the rank differences. The second stage is the *multiple comparison's procedure* [Hollander d Wolfe, 1973] that picks the best performing couple of algorithms. The multiple comparisons' procedure uses the sum of ranks for each algorithm to divide the algorithms into homogenous subgroups. Two algorithms belong to the same subgroup if the difference between the sums of their ranks does not exceed a predefined critical value. The critical value is taken from table A.17 in [Hollander and Wolfe, 1973]. The significant value for this test is derived from the number of repetitions and the number of the compared algorithms, and appears in the same table. The third and final stage is the *sign test* [Hollander and Wolfe, 1973] that picks the best performing algorithm. The sign test checks the significance of difference between the medians of the two algorithms. If the p-value of this test is smaller than the desired significance level, this proves that one algorithm is superior to the other.

3. Methodology

Chapter overview

This chapter describes the methods used in this research. The basic development of this work is presented in the first section. The second section presents the information flow of the sensor fusion framework. The following sections present an overview of the methods used: performance measures that were developed to quantify the logical sensors and the fusion algorithms performances, the sensor fusion algorithms that were developed and implemented, and a statistical evaluation method to check the system's performances that was used in the analysis procedure.

3.1 General

This research is based on Cohen’s PhD thesis [Cohen, 2005] aimed to extend the previous analysis by developing an extended experimental framework. In addition, a new sensor fusion algorithm was developed and analyzed including development of new performance measures and employing a different map representation.

3.1.1 Problem definition

Map a mobile robot's environment by fusing data received from different physical sensors and evaluate fusion performances.

3.1.2 Development basics

The system was developed based on three basic concepts: *logical sensors*, *grid map* paradigm and *performance measures*. These concepts are adapted from Cohen’s work (elaborated in section 2.5) and are modified and extended to meet this research objectives.

In this work, two additional logical sensors were added to the system easily due to the use of logical sensors. The *grid map* paradigm implementation in this work assigns each cell is an integer value that indicated the number of times the logical sensor decided that this cell is occupied with an obstacle. A cell that was declared as *‘Empty’* was assigned the value '0'. This method enables to cells influence on the fusion process in direct proportion to their values. *i.e.,* the higher the value of the cell, the higher it's importance in creating the fused map.

The *performance measures* quantify the difference between two grid maps. Two types of performance measures were used, Type I and Type II. Type I was adapted from Cohen's work (detailed in section 2.5.3). Type II, which was developed in this research, considers not only the cell’s decisions, but also the value of the cell in the calculation process. Performance measures are detailed in chapter 4.

3.1.3 Assumptions

- The environment changes slowly
- Each logical sensor observes the same area
- The logical sensors work asynchronously
- Each logical sensor outputs a two dimensional grid map of the environment
- The resolution of the grid map is higher than obstacle’s resolution; therefore each obstacle occupies a group of cells in the grid map.

3.2 Mapping algorithms

The robot’s environment is represented by grid maps that are built using different mapping algorithms. In the map building process, each sensor’s readings are placed within each sensor’s grid map. Mapping algorithms describe the method for converting raw sensor data to a grid map representation. Since each sensor output has different properties such as shape, accuracy and resolution, which are derived from the sensor’s model, each sensor has a unique mapping algorithm. To create different logical sensors, several mapping algorithms were used for each physical sensor: two mapping algorithms for the ultrasonic sensor and three for the CCD camera sensor were adapted and enhanced from [Cohen, 2005], and two mapping algorithms for the laser sensor were developed in this work. Overall seven mapping algorithms were employed. The different mapping algorithms are detailed in section 6.3.

3.3 Performance measures

Performance measures are used to quantify the difference between the two grid maps as elaborated later [Cohen, 2005]. Two types of sensor performance measure were used: Type I and Type II. Type I was adapted from Cohen's work (detailed in 2.5.3) and are used in the AFL algorithm as the performance measures of the logical sensors as they quantify the difference between the logical sensor's maps (LGM) and the fused map (FGM). The AFL algorithm is detailed in section 2.5.4.2. In addition, they are used to evaluate the difference between the entire environment map (GGM) and the real world map in order to evaluate the sensor fusion algorithm's performances (see section 2.5.6).

Type II performance measures are used in the new developed Adaptive weighted average algorithm (detailed in section 4.2) as the weights of the logical sensors. Type II considers not only the binary decision, *i.e.*, whether the cell is occupied or not, but considers also the value of the cell in the performance measure calculation process.
These performance measures enable to give a higher weight to a logical sensor that occupies similar regions in the LGM and in the fused map. This is important as to differentiate these cells from those that occupy different regions in the LGM than those of the fused map. The higher the logical sensor's performance measure, the logical sensor is given more weight in the fusion process. The type II performance measure is defined as the summation over all cells that are marked as *'Occupy'* in both the logical sensor's map and the fused map of the squared distance between the corresponding cell in the logical sensor's map and the fused map, divided by the square value of the cell in the fused map. Type II performance measures calculation process is elaborated in section 4.2.

3.4 Sensor fusion algorithms

Five different fusion algorithms were used to fuse the logical sensors’ grid maps. The algorithms can be divided into two groups: logical (detailed in 2.5.4.1) and adaptive algorithms.
The adaptive algorithms uses the logical sensor's performance measures in the fusion process. The input of these algorithms are the logical sensor's grid maps (LGM) and each average logical sensor's performance measures ($p_i^{t-1,t-2}$), calculated using the previously built fused map and the LGM. The algorithm gives a higher weight to the better performing logical sensors, so they have more influence on the fusion process. Although the adaptive algorithms are computationally expensive, their ability to use the logical sensor's performances with no a-priori assumption provides an important advantage to the system.

Two adaptive algorithms were evaluated: Adaptive Fuzzy Logic (AFL, developed by Cohen), and a new developed Adaptive Weighted Average algorithm (AdpWA) developed in this research. Adaptive algorithms are detailed in section 5.2.

3.5 Evaluation

The performance of the five sensor fusion algorithms were evaluated using a statistical evaluation method detailed in section 2.5.6.

The evaluation method defines the experimental design and analysis procedure.
In order to evaluate the algorithm's performances, several different experiments must be performed. The experiments differ by changes in the input and in the sensory conditions. Each experiment is performed R times (called repetitions), under the same environmental and sensory conditions. Of course, there are some deviations from one repetition to another, due to changes in the lightning conditions (day/night), temperature, shadows etc.
The algorithms performances are quantified using type I sensor fusion algorithm performance measures. For each algorithm, in every experiment and all repetitions, four performance measures are gathered: OO, EE, OE and EO.
The preliminary step in the evaluation process is to ensure that the experiments are different enough and that there are enough repetitions. The number of repetitions is defined by the statistical parameters (mean and standard deviation) of the performance measures.
The next step is to check which algorithms perform better by applying a statistical analysis that includes three stages. The first stage is Friedman's test, which checks whether the algorithms performances are considered different. The second stage is the multiple comparisons procedure that picks the best performing couple of algorithms, and the third and final stage is the Sign test, that picks the best performing algorithm. The statistical tests also define the number of experiments. The evaluation process and results are detailed in chapter 7.

3.6 Experimental setup and analysis procedure

Cohen's work was expanded to include three physical sensors and a new adaptive weighted average. The expansion was carried out by re-programming Cohen’s framework using a new object oriented API source of libraries, ARIA (see Appendix I). Necessary modifications were made as detailed in Appendix III.

The experiments performed (detailed in chapter 6) consisted of a mobile robot (Pioneer 2-AT) equipped with an array of 16 ultrasonic sensors (8 in the front panel and 8 in back panel), a SONY CCD camera and a SICK laser rangefinder. Only six ultrasonic sensors were used. Two logical sensors were generated using the ultrasonic data. The image registered by the camera was transformed into three logical sensors. The laser scans were transformed into two logical sensors. Each of the logical sensors mapped the area using a LOGM, and the system created a GGM according to the information flow detailed in section 2.5.1.
During the experiments, the robot moved forward at a constant velocity in a controlled laboratory environment and mapped the area in front of it. The area consisted of a black path with obstacles along it. In addition, two types of decoy obstacles were set along the robot's path.
In this research, two sets of evaluations were made. The first set aimed to check the performances of Cohen's system using three physical sensors (instead of two in the original work) with four algorithms: OR, MOST, AND and the adaptive fuzzy logic algorithm. The second set aimed checks the performance of the new developed adaptive weighted average algorithm compared with the adaptive fuzzy logic algorithm.

4. Performance measures

Chapter overview

This chapter presents two types of performance measures. The first type was developed by Cohen [Cohen, 2005], and uses the difference in the binary decisions (*'Occupy'* vs. *'Empty'*) between corresponding cells in two maps. The second type was developed in this research and uses the changes in values between two corresponding cells in two maps.

4.1 General

Performance measures are used to quantify the difference between two grid maps [Cohen, 2005]. Two types of sensor performance measures were used: Type I and Type II.

Type I was adapted from Cohen's work, uses binary decisions about the cell's condition in the grid maps. This type of performance measures checks whether the cell has a positive value (*'Occupied'*) or not (*'Empty'*). Since there are four logical states of binary decisions, Type I performance measures is a vector containing four parameters that all together quantify the difference between the two maps: *OO* (indication to the number of cells that are *'Occupy'* in the first map and also in the second map), *EE* (indication to the number of cells that are *'Occupy'* in the first map and also in the second map), *OE* (indication to the number of cells that are *'Occupy'* in the first map but *'Empty'* in the second map) and *EO* (indication to the number of cells that are *'Empty'* in the first map but *'Occupy'* in the second map). The calculation process is detailed in section 2.5.3.

Type II is used in the adaptive weighted average algorithm as the weights of the logical sensors. Type II counts not only the decision about the cell's condition (*'Occupy'* or *'Empty'*) but also the value in the cell. This type is a scalar indicating the relation between two maps. A map that occupies similar area of cells as the map it was compared to, would have a higher value of this performance measure.

4.2 Type II performance measures

Type II performance measures are used in the adaptive weighted algorithm. Since the grid map paradigm was extended to represent a non-binary grid map and the values of the map represent the number of time each cell was sampled by the sensor, a reliable quantitative measure of the difference between two non-binary grid maps that considers the gap between maps' corresponding cells values is needed. Type II considers not only if the cell is occupied or not, but counts also the value of the cells in the calculation process. This type examines the difference between the occupied cells both in the logical sensor's map and in the fused map. The pseudo-code for calculating Type II performance measures is presented in Table 6 :

Table 6 Pseudo-code for calculating type II performance measures

$$
\begin{aligned}
& for\ i = 1 : MapSizeX \\
& \quad for\ j = 1 : MapSizeY \\
& \quad\quad if\ LSMap_{ij}\ \&\ FusedMap_{ij} \\
& \quad\quad\quad PM^t(i) = PM^t(i) + \left(\frac{LSMap_{ij} - FusedMap_{ij}}{FusedMap_{ij}} \right)^2
\end{aligned}
$$

The calculation process checks all the cells in the logical sensor's map and in the fused map and sums the differences in the following way: if both cells have positive values (implying that these cells are occupied by an obstacle), then the gap between their values can be used to quantify the difference between the two maps. The cells values are subtracted, and in order to normalize the difference, it is divided by the value of the cell in the fused map. To avoid the influence of negative differences, the quotient is squared. $PM^t(i)$ is calculated to each logical sensor at every time sample *t*, where $i = 1, 2, \cdots, N$ and N is the total number of the logical sensors.

As the number of two corresponding cells in both grid maps increases, the performance measure value increases. In other words, the performance measure reveals similarity between two maps that occupy similar regions. Since obstacles in this research's grid map occupy a group of cells, this performance measure enables to denote that performances are increasing when two maps agree on obstacle's location, and to decrease performances when two maps do not agree.

However, the performance measure does not consider agreement on 'Empty' cells between two maps, and thus should be enhanced.

To confirm convergence and in order to mark the best performing sensor, at every time sample *t*, all the performance measures are normalized by dividing each performance measure by the maximum value from all the calculated performance measures. Given the vector of the $PM^t = \left(PM^t(1), PM^t(2), \cdots, PM^t(N) \right)$, the performance measures PM_{II}^t are calculated by:

$$PM_{II}{}^t = \left(\frac{PM^t(1)}{\max\left(PM^t\right)}, \frac{PM^t(2)}{\max\left(PM^t\right)}, \cdots, \frac{PM^t(N)}{\max\left(PM^t\right)} \right) \qquad [20]$$

After performing this step, all the performance measures relate to the most accurate performance measure, which has the value '1'. The rest of the logical sensors have lower values according to their accuracy. These results giving in more weight to logical sensors that occupy similar areas as the fused map and less weight to logical sensors that occupy different regions then the fused map. The following section presents a numerical example of Type II calculation process.

	1	2	3	4
1	20	0	**10**	0
2	0	**12**	20	0
3	0	0	0	0
4	**4**	0	0	0
	Logical sensor 1 (LS1)			

	1	2	3	4
1	15	**17**	0	**24**
2	**12**	0	0	0
3	0	0	0	0
4	0	**20**	0	0
	Logical sensor 2 (LS2)			

	1	2	3	4
1	0	24	2	14
2	7	5	0	25
3	0	0	11	6
4	4	10	0	0
	Fused map (FM)			

Figure 4 numerical calculation example of type II performance measure

Suppose the system contains two logical sensors, with grid maps and a fused map for a certain cycle as shown in Figure 4. As explained above, Type II performance measures considers corresponding cells that are occupied in the logical sensor's map and in the fused map, *i.e.*, the bold cells in the logical sensor's map. The calculation process for the first logical sensor in the example is detailed in [20]:

$$PM_{II}^{(LS1)} = \left(\frac{LS_{13} - FM_{13}}{FM_{13}}\right)^2 + \left(\frac{LS_{22} - FM_{22}}{FM_{22}}\right)^2 + \left(\frac{LS_{41} - FM_{41}}{FM_{41}}\right)^2 = \qquad [21]$$
$$= \left(\frac{10-2}{2}\right)^2 + \left(\frac{12-5}{5}\right)^2 + \left(\frac{4-4}{4}\right)^2 = 4^2 + 1.4^2 + 0^2 = 17.96$$

The calculation process for the second logical sensor in the example is detailed in [22]:

$$PM_{II} = \left(\frac{LS_{12} - FM_{12}}{FM_{12}}\right)^2 + \left(\frac{LS_{14} - FM_{14}}{FM_{14}}\right)^2 + \left(\frac{LS_{12} - FM_{12}}{FM_{12}}\right)^2 + \left(\frac{LS_{24} - FM_{24}}{FM_{24}}\right)^2$$
$$+ \left(\frac{LS_{42} - FM_{42}}{FM_{42}}\right)^2 = \qquad [22]$$
$$= \left(\frac{17-24}{24}\right)^2 + \left(\frac{24-14}{14}\right)^2 + \left(\frac{12-7}{7}\right)^2 + \left(\frac{20-10}{10}\right)^2 =$$
$$= (-0.291)^2 + 0.714^2 + 0.714^2 + 1^2 = 2.104$$

After calculating the measure for each logical sensor separately, the final step is dividing the vector by the maximum value from all performance measures, to normalize and ensure convergence, so the performance measures vector for the given example is presented in [23], indicating that the most accurate logical sensor is LS1:

$$PM_{II} = \left(\frac{17.96}{17.96}, \frac{2.104}{17.96}\right) = (1, 0.117) \qquad [23]$$

5. Sensor fusion algorithms

Chapter overview

Two types of algorithms were evaluated: logical (OR, MOST and AND) and adaptive (fuzzy logic and weighted average). Four adaptive weighted average algorithms were developed. This chapter describes in detail the algorithms.

5.1 General

Two types of sensor fusion algorithms were evaluated. The first type consists of logical algorithms in which the logical sensor distinguishes between two basic states, *Occupy* and *Empty*, and are elaborated in section 2.5.4.1.

The second type of sensor fusion algorithms used the performance of the logical sensors in the fusion. The adaptive algorithms are considered as algorithms that have feedback and memory. In these algorithms, at each time step *t*, the i^{th} logical sensor creates its local grid map (*i.e.*, u_i^t). The fused map (*i.e.*, u_0^t) is built using the average value of the performance measures. The algorithms are considered adaptive, since these values are recalculated online. Although the adaptive algorithms are computationally expensive, their ability to consider the logical sensor's performances with no a-priori assumptions provides an important advantage to the system.

Two adaptive algorithms were evaluated. The Adaptive fuzzy logic (AFL) algorithm (detailed in 2.5.4.2) uses the performance measures as fuzzy variables with three fuzzy sets. The Adaptive weighted average (AdpWA) algorithm considers the values of the cells, (instead of the decision *'Occupy'* or *'Empty'*) and uses the type II logical sensor's performance measures as weights, giving a higher weight to the better performing logical sensor. Four versions of the AdpWA are presented, using different performance measures and a map enhancement procedure.

5.2 Adaptive weighted average algorithm

The Adaptive weighted average (AdpWA) algorithm considers the values of the cells (instead of the decision *'Occupy'* or *'Empty'*), and uses the type II logical sensor's performance measures as weights, giving a higher weight to the better performing logical sensor. Four versions of the AdpWA are presented, using different performance measures and a map enhancement procedure.

The first step in the algorithm is calculating an average map that contains for each cell within the map, the average value from all logical sensors' maps. The next step is calculating for each cell the value of the adaptive weighted average by multiplying the corresponding logical sensor's cells with the logical sensor's performance measure and dividing the product by the sum of the performance measures, for normalization. The fused map is built according to the following rule: if the corresponding cell in the average map is greater or equal to the adaptive weighted average value, the cell in the fused map is assigned the value of the average map. Else, the cell in the fused map is assigned the value '0'. After the fused map is built, the final step is calculating the performance measures for each of the logical sensor. The pseudo-code for calculating the fused map is presented in Figure 5.

Adaptive weighted average algorithm
1. $Calc\ AvgMap^t$ 2. $Build\ u_0^t$: $for\ x = 1 : MapSizeX$ $for\ y = 1 : MapSizeY$ $AdpWA^t(x,y) = \frac{\sum_{i=1}^{N} PM_i^{t-1,t-2} \cdot u_i^t(x,y)}{\sum_{i=1}^{N} PM_i^{t-1,t-2}}$ $if\ AvgMap^t(x,y) > AdpWA^t(x,y)$ $u_0^t(x,y) = AvgMap^t(x,y)$ $else$ $u_0^t(x,y) = 0$ 3. $Calc\ PM_i^{t-1,t-2}$

Figure 5 Adaptive weighted average algorithm pseudo code

The calculation of $AdpWA^t(x,y)$ is done for each cell in the grid map. The sum of products between each logical sensor's performance measure and the cell's value is divided in the sum of all performance measures to receive each cell's weighted average. Since the performance measures changes between cycles according to the sensor's performances, the average is considered adaptive. The adaptive weighted average functions as a threshold as it compared to the value of that cell in the average map. The average map can be considers as a weighted average when each logical sensor is given the same value; comparing the average value to the adaptive threshold allows to reference to each logical sensor's performances, and how the average changes with the performances. The disadvantage of the algorithm is that it requires of the weighted average for each cell separately, and therefore it is computationally expensive in comparison to logical algorithms; However, calculating a different value for each cell allows to give more weight to cells with higher value since more likely that this cell indeed contains an obstacle.

5.2.1 Map enhancement

Map enhancement was introduced into some of the fusion algorithms assuming a relation between a cell and its neighbors. According to the assumption made in the development of the framework (section 3.1.3), an obstacle occupies a group of cells in the grid map; therefore, a cell that is surrounded by occupied cells is more likely to indeed contain an obstacle, rather than a cell that most of its neighbors are empty, and is less likely to contain an obstacle. The purpose of the enhancement procedure is to strengthen occupied cells that are surrounded with occupied cells, by giving them higher values. The procedure checks the number of occupied neighbors for each cell. If the number of occupied neighbors is greater than half of the neighbors – the occupied cells average is added to the cell's value, else – the cell is assigned the value '0'. This applies only to occupied cells (*i.e.*, cells that their value is different than zero), because if the cell is already marked as 'Empty' cell, there is no need to change the sensor's decision.

The following section presents a numerical example for the enhancement calculation procedure.

	1	*2*	*3*	*4*	*5*
1	**10**	5	0	0	0
2	**5**	0	0	10	**5**
3	12	9	0	**7**	**6**
4	0	0	**5**	0	0
5	0	0	12	5	0

(a)
logical sensor's grid map (LS)

	1	*2*	*3*	*4*	*5*
1	**15**	0	0	0	0
2	**13**	0	0	0	**13**
3	0	0	0	**14**	**14**
4	0	0	**14**	0	0
5	0	0	0	0	0

(b)
Enhanced logical sensor's grid map (ELS)

Figure 6 LS and ELS grid maps

Figure 6 presents an example for a logical sensor's grid map (a) and its enhanced grid map (b). The bold cells within the logical sensor's grid map are cells that at least most of their neighbors are occupied, and therefore are strengthen in the enhanced map. The non bold cells are cells that less than half of their neighbors are occupied, and are assigned a zero value in the enhanced grid map.

For example, LS_{11} has three neighbors, and two of them are occupied, so in the enhanced map it will be added with the average value of it's occupied neighbors – 5, and we obtain that $ELS_{11} = 15$. LS_{34} has eight neighbors, and four of them are occupied, so it's value in ELS is the sum of it's value in LS and the average value of it's occupied neighbors (6.5). since the ELS, like the LS grid map contains integer values, this value is rounded up to the nearest integer, and $ELS_{34} = 14$. LS_{32} has eight neighbors, but only three of them are occupied (less then half of the neighbors), therefore we get $ELS_{32} = 0$. This is similar to LS_{53}, which only two of it's neighbors are occupied (instead of at least three), and therefore $ELS_{53} = 0$. The same procedure applies for all LS cells, and the result is presented in Figure 6.

5.2.2 AdpWA algorithms

In order to examine the influence of the enhancement procedure and the type of the performance measures, four different adaptive weighted average algorithms were developed. The algorithms differ in the enhancement procedure and in the type of the performance measures as described in Table 7.

Table 7 Adaptive weighted average algorithms

	Type I Performance measures	**Type II Performance measures**
No Enhancement	AdpWA1	AdpWA2
Enhancement	AdpWA3	AdpWA4

AdpWA1 and AdpWA3 use type I performance measures, while AdpWA2 and AdpWA4 use type II. AdpWA1 and AdpWA2 do not use the map enhancement procedure, while AdpWA3 and AdpWA4 use it. For the evaluation of the different algorithm performances see section 7.3.

6. Mobile robot experiments

Chapter overview

This chapter describes the experimental procedures design, mapping algorithms and setup for the evaluating the sensor fusion framework and the new developed adaptive weighted average algorithm.

6.1 General

The experiment consisted of a mobile robot (Pioneer 2-AT, Figure 7). The robot is equipped with an array of 16 ultrasonic sensors on the robot's front and back panel (eight sensors on each panel), one SICK Laser rangefinder mounted on top of the robot and a PTZ SONY CCD camera mounted on top of the laser sensor. All sensors were used to scan the area in front of the robot; therefore only six ultrasonic sensors from the front of the robot were used. The robot's specifications and parameters are detailed in Appendix I. Two logical sensors were generated using the ultrasonic data, two logical sensors were generated using the laser scans, and three logical sensors were used to describe to image captured by the camera. Overall the system consisted of seven logical sensors.

Two sets of experiments were conducted. The first set aimed to test Cohen's sensor fusion framework using three physical sensors, and the second set aimed to test the new developed AdpWA algorithm's performances, using the same three physical sensors and seven logical sensors configuration.

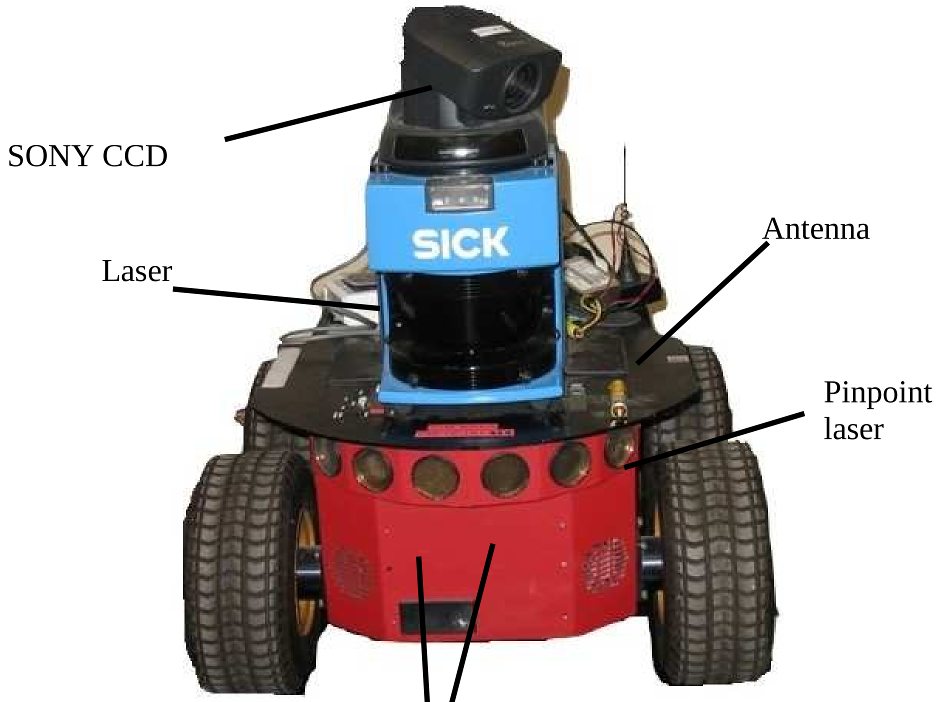


Figure 7 ARAD Pioneer 2-AT

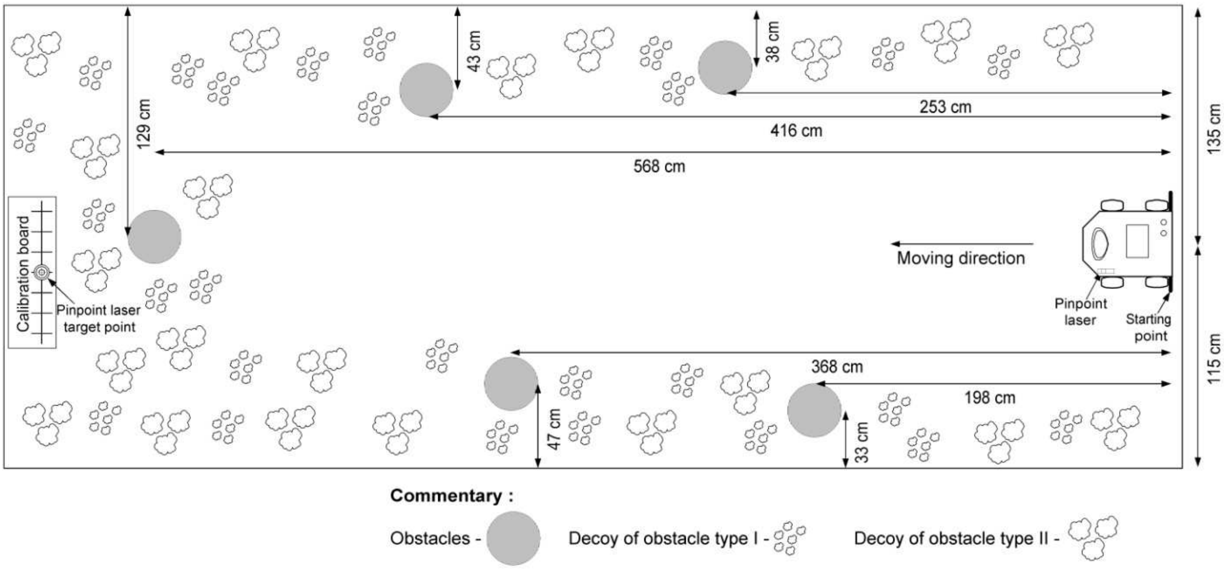


Figure 8 Schematic experimental setup (Adapted from Cohen, 2005)

6.2 Experimental Setup

The robot moved forward at a constant velocity (10 cm/sec) along a 574 cm X 240 cm path in a controlled laboratory environment. As it moved, it mapped the area in front of it. This area consisted of a black path with five obstacles (corrugated red cardboard cylinders Ø25 cm., 50 cm. height located at fixed positions along the path). To increase disagreement between logical sensors, two types of decoy obstacles were set along the robot's path. These decoys were made of light brown rug. The first type of decoy was less than 6 cm. in width and length; the size of the second type was around 25 cm. Decoys locations were randomly changed between repetitions. The schematic experimental setup is presented in Figure 8. Obstacles and decoys are presented in Figure 12. The obstacles and decoys were not always noticeable to all logical sensors because of differences in the algorithms as in the color, size and structure of the decoys themselves. These differences caused the logical sensors to disagree [Cohen, 2005].

Two logical sensors were generated using the ultrasonic data: (i) logical OR algorithm and (ii) probabilistic approach algorithm [Ribo and Pinz, 2001] denoted as US1 and US2 respectively [Cohen, 2005]. The laser scans were transformed into two logical sensors: LASER1 and LASER2. The former used all the 180° scans while the latter used every third scan. The image captured by the camera was transformed into three logical sensors for determining different types of obstacles. The first logical sensor, denoted as CAM1 was used to detect red cardboard cylinders; the second logical sensor, denoted as CAM2 was used to locate the first type of decoys and the third logical sensor, denoted as CAM3 was used to locate the second type of decoys. However, these algorithms were not optimized and their performances very much depended on the lighting conditions, which varied along the path due to external conditions (*e.g.* shadows from the ceilings and from obstacles in the room). To enhance image processing performance, the only light source was a 300W spot placed behind the camera and a sheet of aluminum foil was placed in the back of the spot to prevent light reflection.

In course of traveling 400 ± 5cm., the robot generated 38 fused grid maps. A fusion was conducted whenever a logical sensor was sampled. To eliminate influence of the robot localization problem [Lin *et al.*, 2003] the robot moved only forward. To ensure that the robot traveled straight, the robot was placed at the beginning of the path and a laser pointer mounted

on top of the robot marked the starting point on a calibration board placed at the end of the path. The robot's exact location was changed until the point on the calibration board matched the exact beginning point. At the end of the experiment the robot's location was measured again using the laser and the calibration board and if the robot diverged more than 4cm the repetition was not considered in the analysis.
The robot's software is written in VC++ version 6.0 using ARIA version 2.4 library routines. The robot's operating system is Windows 2000. The experimental software code is detailed in Appendix IV, while the ARIA library routines concept is detailed in Appendix II. During the experiments the robot was controlled and programmed using radio connection and PC Anywhere 8.0 interface via the network.

6.3 Mapping algorithms

Three physical sensors participated in the fusion system: a set of six ultrasonic sensors, a laser rangefinder and a CCD camera. A total of seven algorithms were implemented to generate seven logical sensors out of the physical sensors: two algorithms for the ultrasonic sensors, two algorithms for the laser sensor and three algorithms for the camera.
This section contains the description and pseudo-code for the mapping algorithm for the different physical sensors. The detailed functions mentioned in this section can be found in Appendix IV. The flowcharts of the mapping algorithm are described in Appendix VI.
After building the logical sensor's grid map, each local grid map is transformed relatively to the robot's location and placed in the path planning grid map (PPGM). The robot's location is checked using the robot's encoders, and is divided by the cell's size (5cm) and each cell within the local grid map is copied to the PPGM (function *CopyLBMToGGM*)

6.3.1 <u>Ultrasonic mapping algorithms</u>

Ultrasonic mapping algorithms were adapted from [Cohen, 2005] and were modified to fit the new non-binary grid map paradigm. An array of six ultrasonic sensors in the front panel of the robot was used (sensors 1-6 in Figure 9). Two logical sensors (marked as US1 and US2) describe the sensor's reading using two algorithms.

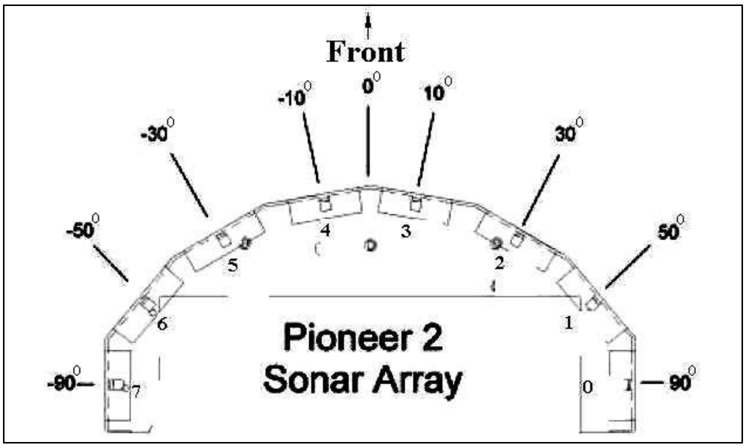


Figure 9 Ultrasonic array (Adapted from Pioneer manual)

The following steps describe the ultrasonic mapping procedure:

1. The sensors are read in a sequential order, from 1 to 6. Each sensor's reading is a value indicating the distance to the nearest obstacle relatively to the physical sensor's location on the robot, as described in Appendix I.

2. Each sensor's reading is placed within one grid map, resulting with a total of six grid maps. Since each sensor's output is one value, each grid map contains one obstacle. Each obstacle's location is calculated locally by finding it's projection on the X and Y axis, and then transformed relatively to the physical sensor's location on the robot (function *US_readDataFromUS*). Since the shape and size of the obstacles are unknown, the obstacles are described by an arc of 10 cm and 10° angle, as Figure 10 presents. The cells inside the arc are marked as empty, the arc cells are marked as occupied and the rest of the cells are marked as unknown. Grid maps cells contain one of three options: '500' indicating that the cell status is unknown, '0' indicating that this cell is empty or any integer value indicating the number of times this cell was declared as occupied, creating non-binary grid map. The grid map is initialized to the value '0' and each time the algorithm decides a cell is occupied, the cell's value is incremented by 1, unless it is assigned with the value '500', then it is assigned the value '1'.
3. Two logical sensors local grid maps are generated by fusing the sensor's maps into two grid map using two algorithms – OR and Probabilistic approach [Cohen, 2005]. The algorithms indicate logical sensors US1 and US2, respectively. The algorithms were modified to fit the non-binary grid map concept. Each algorithm serially fuses the generated grid maps of all logical sensors (one after another) using its own truth table (functions *US_SFA_LogicalOR US_SFA_ProbablisticApproach*). Table 8 and Table 9 present the OR and Probabilistic approach truth table, respectively. The 'Max' value in the tables means that the selected value is the maximum value within the occupied cells values.

500	500	500	500	500	500
500	7	3	8	2	500
500	1	4	2	7	500
500	0	0	0	0	500
500	0	0	0	0	500
500	0	0	0	0	500
500	0	0	0	0	500
500	0	0	0	500	500
500	500	0	0	500	500
500	500	0	0	500	500
500	500	0	0	500	500

Figure 10 Ultrasonic grid map model

Table 8 OR algorithm truth table (US1)

		US_j		
		Empty	**Occupied**	**Unknown**
US_i	**Empty**	Empty	Max	Empty
	Occupied	Max	Max	Max
	Unknown	Empty	Max	Empty

Table 9 Probablistic approach truth table (US2)

		US_j		
		Empty	**Occupied**	**Unknown**
US_i	**Empty**	Empty	Empty	Empty
	Occupied	Empty	Max	Max
	Unknown	Empty	Max	Empty

6.3.2 Camera algorithms

Camera's Mapping algorithms were adapted from [Cohen, 2005] and were changed to fit the environmental conditions in the lab and the new non-binary grid map paradigm.

Sampling the camera sensor means taking photos in four pan angles: -17°, 17°, 50°,-50°. The order of the pan angles changes between odd and even cycles. Three logical sensors are implemented using the camera pictures (marked as CAM1, CAM2 and CAM3); each logical sensor differs in the image processing algorithm, *i.e.*, each logical sensor's map is created using a different image processing function. The algorithms differ in the obstacles they are designed to detect. CAM1 is designed to detect the obstacles, CAM2 is designed to detect the first type of decoys and the obstacles and CAM3 is designed to detect the second type of decoys and the obstacles (see section 6.2).

The following steps describe the camera mapping procedure:

1. The camera is set to one of four possible pan angle (-17°, 17°, 50° or -50°), and takes a picture.
2. Obstacles center of mass in the picture is found using three image processing algorithm and its X and Y location in the map is calculated using the camera's calibration process and saved in a special vector for each image processing algorithm.
3. Steps 1 and 2 are taken for the next pan angle, until all four pan angles were sampled. Overall, twelve obstacle's location vectors represent the obstacle found in each picture: three image processing vectors for each one of the four pan angle (function *ImageProcessingAlgo3*).
4. For each image processing algorithm, according to the obstacle's location vectors, the obstacles from the different pan angles are placed in the grid map. Since the size and the shape of the obstacles are unknown, around every obstacle's location a circle (Ø 15cm) is drawn. The grid map is initialized to the value '0' and each time the algorithm decided a cell is occupied, the cell's value is incremented by 1. This step resulting in three local grid maps that represent three logical sensors: CAM1, CAM2 and CAM3 (function *ImageProcessingAlgo4*).

<u>*Image processing algorithms*</u>

The algorithms use *Threshold*, *iplErode*, *iplDilate* and *cvFindCountors* function from Intel's CV and IPL (see Appendix IV), and differ in the functions constants. Each algorithm is design to detect different kind of obstacles according to their area as found from the *cvFindCountors* function. The areas for each algorithm are shown in Table 10. The threshold values and the obstacle's minimum and maximum sizes were found empirically and were adapted to the lightning conditions in the lab where the experiments took place.

Algorithm 1 – CAM1

1. Convert the photo from RGB into grayscale format.
2. Run *Threshold(120)* on the grayscale photo and save in BW format.
3. Run *cvFindCountors* and find the center of mass of all obstacles.
4. Map the obstacles their area fit the range in Table 10.

Algorithm 2 – CAM2

1. Convert the picture from RGB into grayscale format.
2. Run *Threshold(120)* on the grayscale photo and save in BW format.
3. Run *iplErode(3)* on the BW photo and save it.
4. Run *iplDilate(5)* on the BW photo and save it.
5. Run *cvFindCountors* and find the center of mass of all obstacles.
6. Map the obstacles their area fit the range in Table 10.

Algorithm 3 – CAM3

1. Convert the picture from RGB into grayscale format.
2. Run *Threshold(150)* on the grayscale photo and save in BW format.
3. Run *iplErode(3)* on the BW photo and save it.
4. Run *iplDilate(4)* on the BW photo and save it.
5. Run *cvFindCountors* and find the center of mass of all obstacles.
6. Map the obstacles their area fit the range in Table 10.

Table 10 Detection area for each algorithm

	CAM1	CAM2	CAM3
Min	15,000	600	8000
Max	47,000	33,000	33,000

6.3.3 Laser algorithms

Two mapping algorithms were developed in this research. The laser sensor senses the environment in front of it at a 180° in an angular resolution of 1°. This results in an output vector with 181 readings. Each cell in the reading indicates the distance to the nearest obstacle at a specific angle. The laser logical sensors (marked as LASER1 and LASER2) were implemented using two algorithms. The two laser mapping algorithms differ by the number of readings that are marked in the grid map.

The following steps describe the ultrasonic mapping procedure:

1. The laser is sampled and a vector of readings is saved.
2. For LASER1 - all the readings are placed in a grid map according to the reading and its angle. A small circle (Ø 5cm) is placed around each reading in the grid map. For LASER2 – every 3rd reading is placed within the grid map, and a small circle (Ø 5cm) is placed around it. The local grid map is initialized to the value '0' and each time the algorithm decided a cell is occupied, the cell's value is incremented by 1 (function *ReadFromSick*).

6.4 Experimental procedure

An experiment is defined as a mapping task for specific environmental and sensory conditions. At each cycle, the robot mapped the environment using samples from six of the ultrasonic sensors in the front panel, laser scans and photos taken from the camera. In order for the camera to capture the obstacles and the decoys, the camera tilt angle was set to -25°. The camera took photos from four pan angles (-50°, -17°, 17° and 50°), as shown in Figure 11.

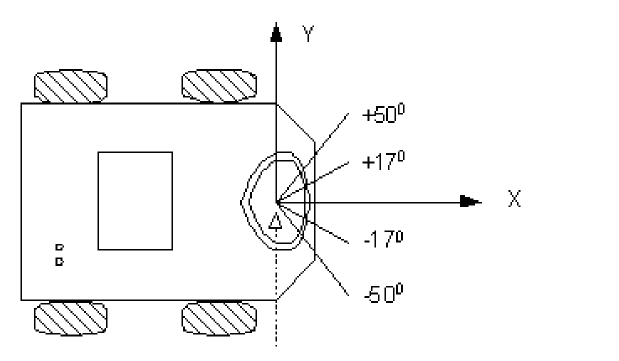
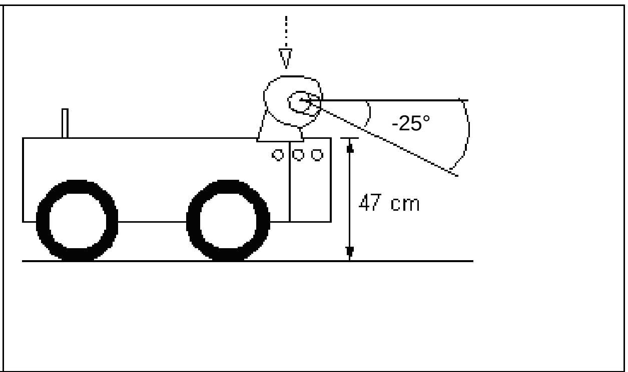
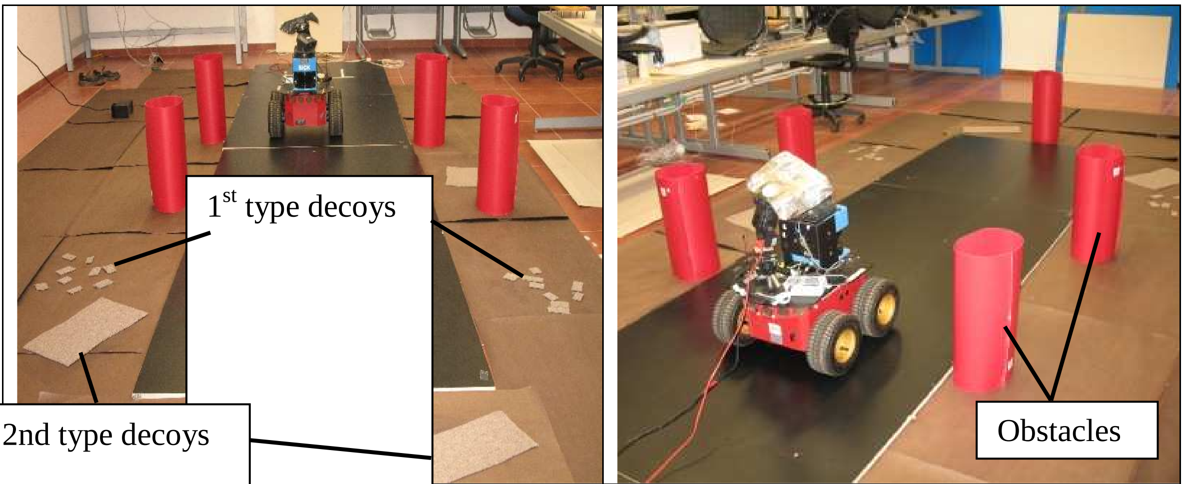
	
Pan angles	Tilt angle

Figure 11 Pan and Tilt angles (Adapted from Cohen, 2005)

The slowest sensor was the camera, due to the slow pan angles changing process. In order to increase the number of cycles in the experiment, the order of the pan angles was reversed between odd and even cycles. The pan angles order in the even cycles is -50°→-17°→17°→50° and in the odd cycles is reversed, *i.e.*, 50°→-17°→-17°→-50°. Image processing algorithms recognized the obstacles and decoys from the different angles, and placed them in the camera's logical sensor's LBM according to a calibration process detailed in Appendix V. Environmental and sensory conditions are changed for each new experiment using a methodology described in chapter 7 . The obstacle locations are constant and therefore identical for all experiments. Each experiment is performed R times (called repetitions), for identical environmental and sensory conditions. The difference between repetitions is caused by randomly changing the location of the decoys. Statistical procedures (detailed in section 2.5.6) determined the number of experiments and repetitions. At the end of each repetition, all logical sensors maps and the data received from the sensors and robot's encoders were saved on the robot's hard drive for offline analysis.



12 Experimental setup photographs

he performance measures for the AFL and AdpWA algorithms were set as 1 and in Table 12 respectively. These values were determined randomly and r all experiments.

Table 11 Experimental initial perofrmance measures for AFL algorithm

Logical sensor	Initial performance measures values			
	$OO^0(i)$	$EE^0(i)$	$EO^0(i)$	$OE^0(i)$
US1	0.85	0.9	0.1	0.15
US2	0.78	0.91	0.19	0.22
CAM1	0.8	0.7	0.3	0.2
CAM2	0.6	0.9	0.1	0.4
CAM3	0.88	0.91	0.09	0.12
LASER1	0.92	0.95	0.05	0.08
LASER2	0.93	0.95	0.05	0.07

Table 12 Experimental initial peroformance measures for AdpWA algorithm

Logical sensor	Initial performance measures values
US1	0.3
US2	0.3
CAM1	0.1
CAM2	0.1
CAM3	0.1
LASER1	1
LASER2	1

7. Evaluation and Results

Chapter overview

This chapter presents the algorithm's evaluation results. Two sets of experiments were conducted. The first set aimed to test the performance of sensor fusion algorithms using the new sensor fusion framework while the second set aimed to test the performances of the new adaptive weighted algorithms.

7.1 General

Sensor fusion algorithms were evaluated using the evaluation method developed by [Cohen *et. al.*, 2005]. To evaluate the algorithm's performances, several different experiments were performed. The experiments differ by changes in the input and in the sensory conditions. Malfunctions were created artificially by setting logical sensors to empty, full and shifting positions by a constant value. Each experiment is performed R times (called repetitions), under the same environmental and sensory conditions. Of course, there are some deviations from one repetition to another, due to changes in the lighting conditions (day/night), temperature, shadows, etc. The algorithms performances are quantified using type I sensor fusion algorithm performance measures, as detailed in section 2.5.3.2 . For each algorithm, in every experiment and all repetitions, four performance measures are gathered: OO, EE, OE and EO.

The statistical method is detailed in section 2.5.6.

The first set of experiments aimed to test the performances of Cohen's extended sensor fusion algorithm framework which uses three physical sensors instead of two, as in Cohen's work. The second set of experiments aimed to test the performances of the new developed adaptive weighted algorithm. The AdpWA performances were tested using the extended fusion framework fusing data from three physical sensors.

7.2 Extended sensor fusion framework evaluation

7.2.1 General

In this research, Cohen's fusion framework was extended to fuse data from three physical sensors. The additional physical sensor, a laser rangefinder, was added to the system, as described in chapter 6. In the extended framework, four sensor fusion algorithms were employed: OR, MOST, AND and AFL as detailed in section 2.5.4. In order to test the sensor fusion algorithms using the new extended framework, a set of experiments was conducted. The experiments design and procedure are detailed in chapter 6.

7.2.2 Experimental design

Seven different experiments were conducted (Table 13). The experiments differ in the environmental conditions and in sensory input. Different environmental conditions were chosen to ensure that the results are not specific for a dataset only. Each experiment was repeated seven times. The number of experiments and repetitions required derives from several parameters, including the statistical characteristics of the data (*e.g.,* standard deviation), the desired α value and Δ, the minimum difference to be detected [Cohen, 2005]. Hence, it is impossible to predict *a-priori* the number of experiments and repetitions required. Therefore, the initial number of experiments and repetitions was chosen arbitrary as four. Lighting conditions were changed in the third and seventh experiment. Experiment's repetitions were performed under the same conditions with natural variations such as lightning conditions, shadows and time differences. However, calculating the volume of overlap region (VOLR) showed that the experiments were not different enough (VOLR>0), therefore three additional experiments were performed. In each experiment, environmental mapping was achieved using the four different sensor fusion algorithms, resulting in a total of 196 environmental mappings (4 sensor fusion algorithms X 7 Experiments X 7 repetitions). Figure 13 presents the algorithms' map results from experiment 1, first repetition and the corresponding real world map. All logical sensors mappings from all experiments are presented in Table 22, and algorithms mappings are presented in Table 23.

Table 13 Experimental design for statistical evaluation experiment

Exp.	US1	US2	LASER1	LASER2	CAM1	CAM2	CAM3	Comments
1	Empty	Regular Algorithm	Full	Regular Algorithm	Regular Algorithm	Shift: X=X+40cm Y=Y+40cm	Shift: X=X-40cm Y=Y-40cm	
2	Full	Regular Algorithm	Empty	Regular Algorithm	Regular Algorithm	Shift: X=X-40cm Y=Y-40cm	Empty	
3	Regular Algorithm	Empty	Regular Algorithm	Full	Regular Algorithm	Regular Algorithm	Regular Algorithm	Lights off for cycles 15-end
4	Regular Algorithm	Full	Regular Algorithm	Empty	Regular Algorithm	Regular Algorithm	Full	
5	Regular Algorithm	Full	Regular Algorithm	Empty	Shift: X=X+20cm Y=Y-40cm	Full	Shift: X=X-40cm Y=Y+60cm	
6	Regular Algorithm	Empty	Regular Algorithm	Full	Regular Algorithm	Shift: X=X+60cm Y=Y+60cm	Full	
7	Regular Algorithm	Regular Algorithm	Regular Algorithm	Regular Algorithm	Regular Algorithm	Regular Algorithm	Regular Algorithm	Lights off for cycles 15-end

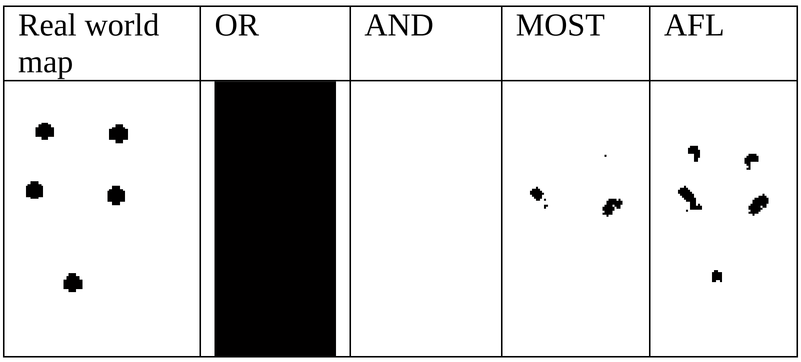


Figure 13 Map results, experiment 3, first repetition

Different experiments

For seven LS, seven repetitions and seven experiments, this result in 7,203 subtracted maps, as derived from [17] and presented in [24]. For each comparison, the worst difference of all logical sensors is saved [Cohen, 2005], resulting in 343 maps.

$$N_{Exp.} = 7 \cdot 7^2 \cdot \binom{7}{2} = 7{,}203 \qquad [24]$$

Similar repetitions

For seven LS, seven repetitions and seven experiments that were chosen arbitrarily, this result in 1,029 comparisons, as derived from [18] and presented in [25]. For each comparison, the worst difference is saved, *e.g.*, the maximum number of signed cells [Cohen, 2005], resulting in 49 maps.

$$N_{\mathrm{Re}\,p.} = 7 \cdot 7 \cdot \binom{7}{2} = 1{,}029 \qquad [25]$$

Volume of overlap region

The maximum number of signed cells was calculated for all experiments and repetitions. The volume is negative as equation [26] shows, implying that the experiments are different and repetitions are similar.

$$VOLR = \frac{MIN(5136{,}1067) - MAX(4160{,}156)}{MAX(5136{,}1067) - MIN(4160{,}156)} = -0.7114 \qquad [26]$$

Number of repetitions

The number of repetitions is based on a t-test detailed in [Cohen, 2005] and is calculated for α=0.05 and β=0.2. Standard deviation (S) and mean values are taken as the upper bound from all experiments and algorithms. In order to allow reasonable error, Δ is chosen to be 20% from the mean upper bound. This value was chosen arbitrarily. Results are presented in Table 14. Based on these results, the largest R is for the OO measure; this results in seven necessary repetitions. Since each experiment has already seven repetitions, no additional repetitions were required.

Table 14 R calculations for each performance measure

Performance Measure	S	Δ	$R_{adjusted}$	R
OO	0.068	0.072	6.220	**7**
EE	0.039	0.195	0.287	1
OE	0.009	0.200	0.016	1
EO	0.086	0.200	1.326	2

7.2.3 <u>Performance measure calculation and grouping</u>

Table 15 presents an example of raw data for one of the repetitions in one of the experiments for all sensor fusion algorithms. Raw data for the whole experiment set is detailed in Appendix VIII. An example of the resulting OO values for all repetitions is presented in Table 16.

Table 15 Sensor fusion performance measures values for experiment 2, first repetition

Algorithm	OO	EE	OE	EO
OR	0.038	0	1	0
AND	0	0.962	0	1
MOST	0.0048	0.9634	0.0001	0.8459
AFL	0.3513	0.9754	0.0001	0.0385

Table 16 OO Measure for four algorithms, seven repetitions, Experiment 7

	Repetition number						
Algorithm	**1**	**2**	**3**	**4**	**5**	**6**	**7**
OR	0.226	0.214	0.165	0.183	0.175	0.159	0.198
AND	0.001	0.005	0.001	0.001	0.004	0.004	0.005
MOST	0.295	0.374	0.278	0.252	0.180	0.230	0.193
AFL	0.416	0.395	0.401	0.361	0.220	0.305	0.234

7.2.4 Statistical analysis

Friedman's test

An example of Friedman' test ranking for the OO measure of one experiment is presented in Table 17. The entries in each row are the ranks of each algorithm within the seven replications. Friedman’s ranking for all experiments are presented in Appendix IX.
P-values for all 7 experiments for all seven experiments are presented in Table 18 . The very small p-values imply a difference between algorithms.

Table 17 Example of Friedman's test ranking, OO measure, experiment 7, seven repetitions
(Note: for OE and EO smaller values is preferable)

		Algorithm			
	Repetition	**OR**	**AND**	**MOST**	**AFL**
Rank	**1**	2	1	3	4
	2	2	1	3	4
	3	2	1	3	4
	4	2	1	3	4
	5	2	1	3	4
	6	2	1	3	4
	7	3	1	2	4
	Sum	15	7	20	28

Table 18 Friedman's test results

Experiment	Sensor fusion performance measures	p - value	Experiment	Sensor fusion performance measures	p - value
1.	OO	0.0002	**5.**	OO	0.0002
	EE	0.0001		EE	0.0001
	OE	0.0003		OE	0.0003
	EO	0.0002		EO	0.0004
2.	OO	0.0001	**6.**	OO	0.0005
	EE	0.0001		EE	0.0001
	OE	0.0001		OE	0.0002
	EO	0.0002		EO	0.0002
3.	OO	0.0004	**7.**	OO	0.0001
	EE	0.0001		EE	0.0003
	OE	0.0002		OE	0.0004
	EO	0.0005		EO	0.0001
4.	OO	0.0003			
	EE	0.0006			
	OE	0.0005			
	EO	0.0005			

Multiple comparison procedure

According to table A.17 in [Hollander and Wolfe, 1973], for a significance level of 0.02, four algorithms and seven repetitions require a difference equal or greater than 14 between their algorithm's sum of ranks in order to considered as different algorithms. Table 19 describes an example of the 28 multiple comparison procedures detailed in Appendix X. A close look at the results indicates that in most cases MOST and AFL algorithms belong to the same best subgroup and thus they are considered the two best performing algorithms.

Table 19 Multiple comparison results for all PM , experiment 7

(Note: for OE and EO smaller values is preferable)

Experiment 7								
OO measure				**EE measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups		
AFL	28	A		AFL	27	A		
MOST	20	A	B	MOST	22	A	B	
OR	15	A	B	AND	14	A	B	C
AND	7		B	OR	7			C
OE measure				**EO measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups		
AND	27	A		AFL	28	A		
AFL	20	A	B	MOST	21	A	B	
MOST	16	A	B	OR	14	A	B	
OR	7		B	AND	7		B	

Sign test

Final comparison between the two best performing algorithms (AFL *vs.* MOST) is the sign test. For each experiment, four performance measures was tested, overall 28 cases was examined. Sign test data is presented in Appendix XI. Table 20 presents the sign test data summary. The AFL algorithm outperformed the MOST algorithm in 12 cases, in 4 cases

MOST outperformed AFL and in 12 cases both of them yielded identical results. The significance level corresponding to this case is equal to 0.077, as presented in Table 21. Table 21 was generated using SPSS for windows software release 12.0.0. The small significance level implies that the AFL algorithm is the best performing algorithm.

Table 20 Sign test data

Experiment environmental conditions	Sensor fusion performance measures							
	OO		EE		OE		EO	
	MOST	**AFL**	**MOST**	**AFL**	**MOST**	**AFL**	**MOST**	**AFL**
1.	Ties	Ties	Ties	Ties	Ties	Ties	Ties	Ties
2.	0	7	0	7	0	3	0	7
3.	0	7	0	7	3	4	0	7
4.	Ties	Ties	Ties	Ties	Ties	Ties	Ties	Ties
5.	7	0	7	0	7	0	7	0
6.	Ties	Ties	Ties	Ties	Ties	Ties	Ties	Ties
7.	0	7	1	6	1	6	1	6
Total	**1**	**3**	**1**	**3**	**1**	**3**	**1**	**3**

Note: The values in this table indicate the number of times each algorithm outperforms the opponent.

Table 21 Sign test results

Frequencies

		N
MOST - AFL	Negative Differences(a)	12
	Positive Differences(b)	4
	Ties(c)	12
	Total	28

a MOST < AFL
b MOST > AFL
c MOST = AFL

Test Statistics(b)

	MOST - AFL
Exact Sig. (2-tailed)	.077(a)

a Binomial distribution used.
b Sign Test

7.2.5 Discussion

The evaluation method presented in this section indicates that the two best performing algorithms are the AFL and MOST. Of the two, the AFL is superior. OR and AND algorithms have poor performances. These evaluations correspond to previous results [Cohen, 2005] and to visual presentations of the generated maps.

Table 22 Logical sensors mapping in the extended sensor fusion framework

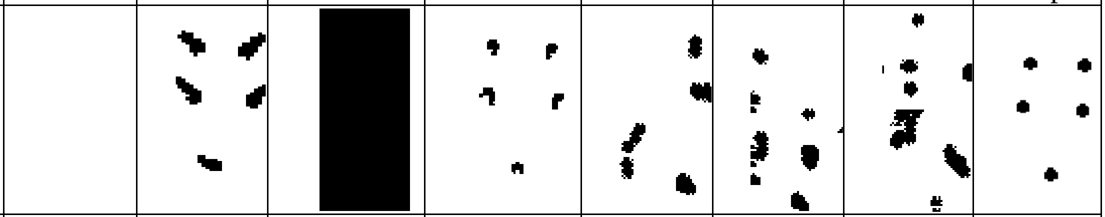
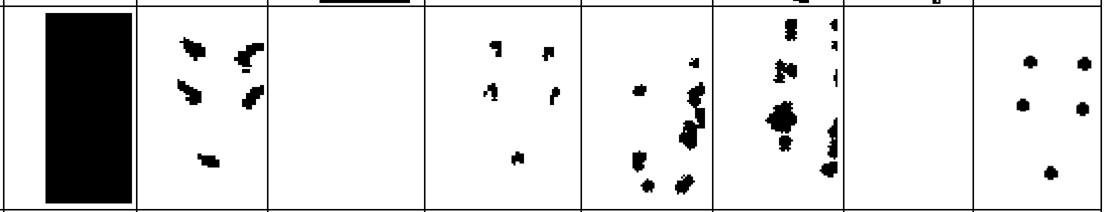
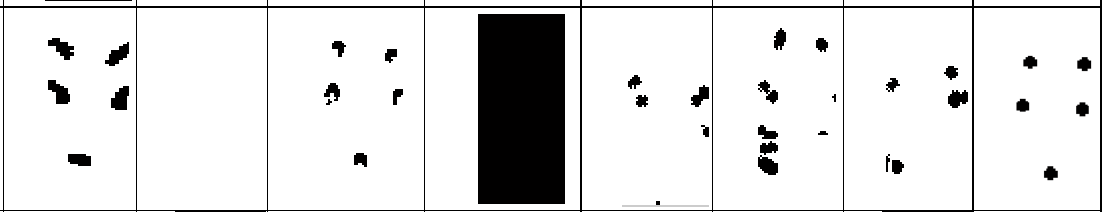
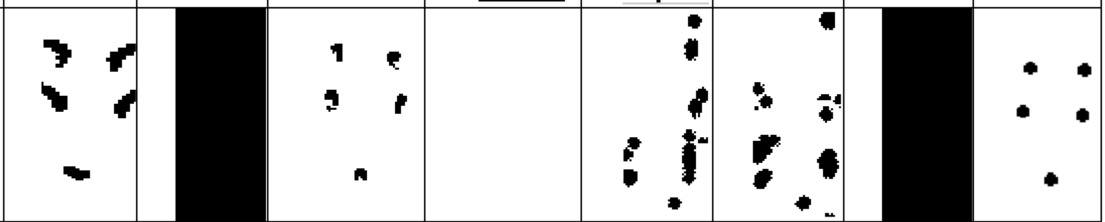
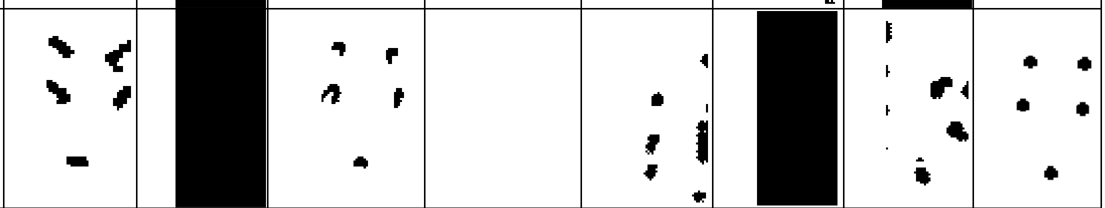
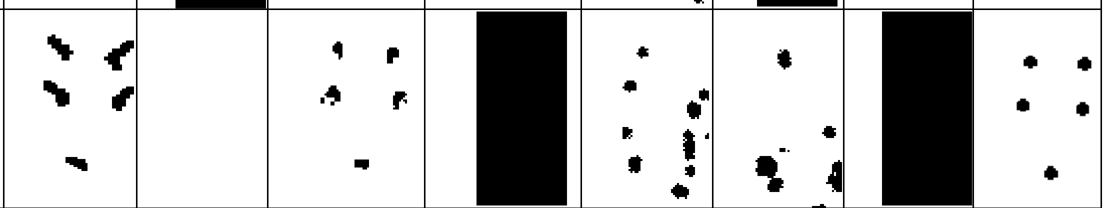
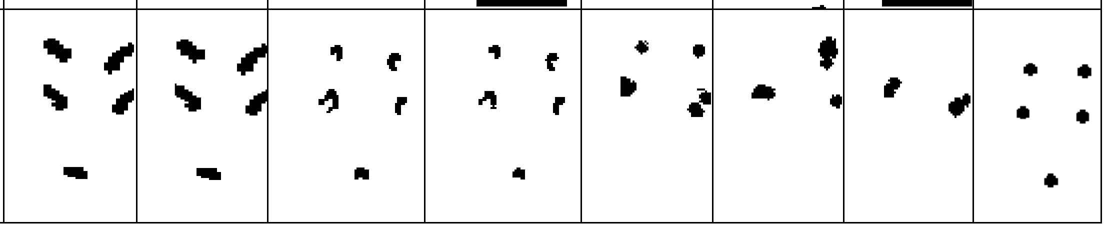
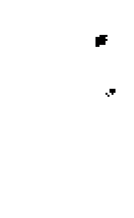
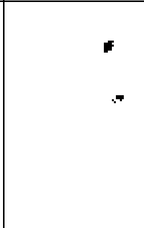
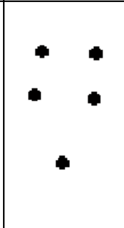
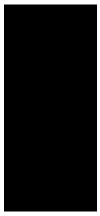
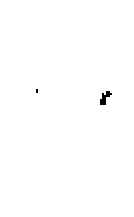
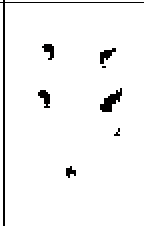
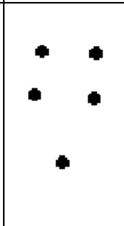
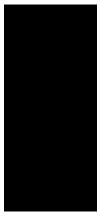
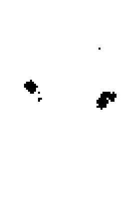
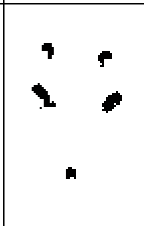
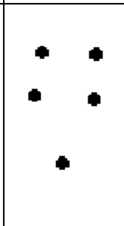
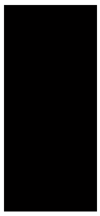
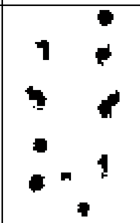
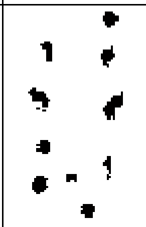
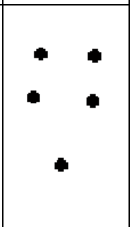
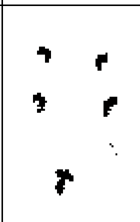
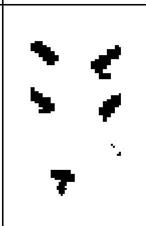
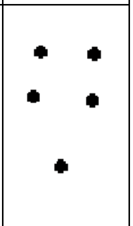
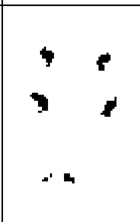
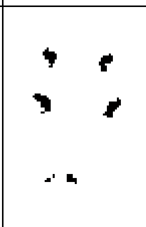
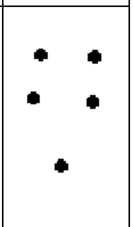
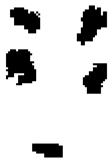
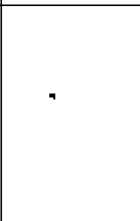
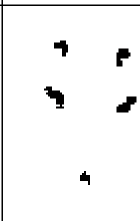
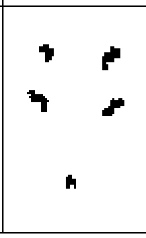
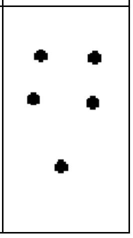
	Logical sensor							Real world map
Exp.	US1	US2	LASER1	LASER2	CAM1	CAM2	CAM3	
1								
2								
3								
4								
5								
6								
7								

Table 23 Algorithms mapping in the extended sensor fusion framework

	Sensor fusion algorithm				Real world map
Exp.	OR	AND	MOST	AFL	
1					
2					
3					
4					
5					
6					
7					

7.3 Adaptive weighted algorithm evaluation

7.3.1 General

A new adaptive weighted algorithm (AdpWA) is presented in this research. The algorithm uses the values in the maps and in performance measures for building the fused map, and was implemented in the extended sensor fusion framework. The algorithm was implemented using a set of algorithms, which differ in the performance measures type and include implementation of a map enhancement procedure. The algorithm set is fully described in section 5.2. To test the algorithms' performances, Cohen's evaluation method was applied [Cohen, 2005]. A total of five sensor fusion algorithms were employed: AdpWA1, AdpWA2, AdpWA3 ,AdpWA4 and AFL. In order to test their performances, a set of experiments was conducted. The experimental set was defined using the evaluation method developed by Cohen [Cohen, 2005].

7.3.2 Experimental design

Four different experiments were conducted (Table 24). The experiments differ in the environmental conditions and in sensory input. Different environmental conditions were chosen to ensure that the results are not only for a specific dataset. Each experiment was repeated six times. The number of experiments and repetitions required derives from several parameters, including the statistical characteristics of the data (*e.g.* standard deviation), the desired α value and Δ, the minimum difference to be detected [Cohen, 2005]. Hence, it is impossible to predict *a-priori* the number of experiments and repetitions required. Therefore, the initial number of experiments and repetitions was chosen arbitrary. Lighting conditions were changed in the fourth experiment by turning off the lights. Experiment's repetitions were performed under the same conditions with natural variations such as lightning conditions, shadows and time differences. In each experiment, environmental mapping was achieved using the five different sensor fusion algorithms, resulting in a total 120 of environmental mappings (5 sensor fusion algorithms X 4 Experiments X 6 repetitions). Figure 14 presents the algorithms' map results from experiment 1, first repetition and the corresponding real world map. All logical sensors mappings are presented in Table 33 and algorithms mappings are presented in Table 34.

Table 24 Experimental design for statistical evaluation experiments

Exp.	US1	US2	LASER1	LASER2	CAM1	CAM2	CAM3	Comments
1	**Empty**	Regular algorithm	**Full**	Regular Algorithm	Regular Algorithm	**Shift: X=X+40 Y=Y-60**	**Shift: X=X-60 Y=Y+40**	
2	Regular Algorithm	**Empty**	Regular Algorithm	**Full**	**Shift: X=X+100 Y=Y-100**	Regular Algorithm	Regular Algorithm	
3	**Empty**	Regular Algorithm	Regular Algorithm	**Empty**	**Empty**	Regular Algorithm	**Shift: X=X+100 Y=Y-120**	
4	**Empty**	Regular Algorithm	Regular Algorithm	**Empty**	Regular Algorithm	Regular Algorithm	Regular Algorithm	Lights off

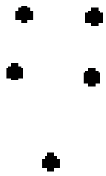
Real world map	AdpWA1	AdpWA2	AdpWA3	AdpWA4	AFL
					

Figure 14 Map results, Experiment 3, first repetition

Different experiments

For seven logical sensors, seven repetitions and four experiments that were chosen arbitrarily, these results in 1,512 subtracted maps, as derived from [17] and presented in [27]. For each comparison, the worst difference of all logical sensors is saved [Cohen, 2005], resulting in

$$N_{Exp.} = 7 \cdot 6^2 \cdot \binom{4}{2} = 1{,}512 \qquad [27]$$

Similar repetitions

For seven LS, six repetitions and four experiments, this results in 420 comparisons, as derived from [18] and presented in [28]. For each comparison, the worst difference is saved, *e.g.*, the maximum number of signed cells [Cohen, 2005], resulting in 168 maps.

$$N_{\mathrm{Re}\,p.} = 7 \cdot 4 \cdot \binom{6}{2} = 420 \qquad [28]$$

Volume of overlap region

The maximum number of signed cells was calculated for all experiment and repetitions. The volume is negative as equation [29] shows, implying that the experiments are different and repetitions are similar.

$$VOLR = \frac{MIN(5136{,}754) - MAX(877{,}78)}{MAX(5136{,}754) - MIN(877{,}78)} = -0.024 \qquad [29]$$

Number of repetitions

The number of repetitions is based on a t-test detailed in [Cohen, 2005] and is calculated for α=0.05 and β=0.2. Standard deviation (S) and mean values are taken as the upper bound from all experiments and algorithms. Δ is chosen to be 15% from the average upper bound. The results are presented in Table 25. Based on these results, the largest R is for the OO measure; this results in six necessary repetitions. Since each experiment has already six repetitions, no additional repetitions were required.

Table 25 R calculations for each performance measure

Performance Measure	S	Δ	$R_{adjusted}$	R
OO	0.0371	0.041	5.838	**6**
EE	0.0221	0.1459	0.163	1
OE	0.0011	0.1471	0.0003	1
EO	0.0625	0.15	1.237	2

7.3.3 Performance measure calculation and grouping

Table 26 presents an example of raw data for one of the repetitions in one of the experiments for all sensor fusion algorithms. Raw data for the whole experiment set is detailed in Appendix XII. An example of the resulting EE values for all repetitions is presented in Table 27.

Table 26 Sensor fusion performance measures values for experiment 3, second repetition

Algorithm	OO	EE	OE	EO
AdpWA1	0.242	0.971	0.000	0.068
AdpWA2	0.196	0.878	0.004	0.310
AdpWA3	0.115	0.969	0.000	0.320
AdpWA4	0.231	0.935	0.001	0.221
AFL	0.000	0.962	0.000	1.000

Table 27 EE Measure for five algorithms, six repetitions, Experiment 2

	Repetition number					
Algorithm	**1**	**2**	**3**	**4**	**5**	**6**
AdpWA1	0.974	0.970	0.972	0.957	0.963	0.973
AdpWA2	0.000	0.000	0.000	0.000	0.000	0.000
AdpWA3	0.971	0.969	0.970	0.970	0.968	0.970
AdpWA4	0.002	0.002	0.002	0.002	0.002	0.002
AFL	0.974	0.971	0.973	0.973	0.970	0.973

7.3.4 Statistical analysis

Friedman's test

An example of Friedman' test ranking for the EE measure of experiment 2 is presented in Table 28. The entries in each row are the ranks of each algorithm within the seven replications. Friedman's ranking for all experiments are presented in Appendix XIII.

P-values for all 7 experiments for all seven experiments are presented in Table 29. The very small p-values imply a difference between algorithms.

Table 28 Example of Friedman's test ranking, OO measure, experiment 2, seven repetitions

(Note: for OE and EO smaller values is preferable)

		Algorithm				
	Repetition	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
Rank	**1**	5	1	3	2	4
	2	5	1	3	2	4
	3	5	1	3	2	4
	4	5	1	3	2	4
	5	5	1	3	2	4
	6	5	1	3	2	4
	Sum	**30**	**6**	**18**	**12**	**24**

Table 29 Friedman's test results for AdpWA algorithm

Experiment	**Sensor fusion performance measures**	**p - value**	**Experiment**	**Sensor fusion performance measures**	**p - value**
1.	OO	0.0012	**3.**	OO	0.001
	EE	0.0005		EE	0.0005
	OE	0.0006		OE	0.0005
	EO	0.0005		EO	0.0005
2.	OO	0.0005	**4.**	OO	0.0005
	EE	0.0012		EE	0.0012
	OE	0.0012		OE	0.0012
	EO	0.0005		EO	0.0005

Multiple comparison procedure

According to table A.17 in [Hollander and Wolfe, 1973], for a significance level of 0.049, five algorithms and six repetitions require a difference equal or greater than 15 between the algorithm's sum of ranks in order to be considered as different algorithms. Table 30 describes an example of the 16 multiple comparison procedures detailed in Appendix XIV. A close look at the results indicates that in most cases AdpWA1 and AFL algorithms belong to the same best subgroup and thus they are considered the two best performing algorithms.

Table 30 Multiple comparison results for all PM , experiment 1

(Note: for OE and EO smaller values is preferable)

Experiment 1									
OO measure					**EE measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups			Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA4	29	A			AdpWA1	30	A		
AdpWA2	23	A	B		AdpWA3	24	A	B	
AdpWA1	20	A	B	C	AFL	18	A	B	C
AFL	12		B	C	AdpWA4	12		B	C
AdpWA3	6			**C**	AdpWA2	6			**C**
OE measure					**EO measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups			Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA3	30	A			AdpWA2	27	A		
AdpWA1	22	A	B		AdpWA4	27	A		
AFL	20	A	B	C	AdpWA1	18	A	B	
AdpWA4	12		B	C	AFL	12	A	B	
AdpWA2	6			C	AdpWA3	6		B	

Sign test

Final comparison between the two best performing algorithms (AdpWA1 *vs.* AFL) was done using the sign test. For each experiment, four performance measures were tested, resulting in overall 16 cases. Sign test data it presented in Appendix XV. Table 31 presents the sign test data summary. The AdpWA1 algorithm outperformed the AFL algorithm is 13 cases, and in 3 cases AFL outperformed AdpWA1. The significance level corresponding to this case is equal to 0.021, as presented in Table 32. Table 32 was generated using SPSS software for windows release 12.0.0. The small significance level implies that the AdpWA1 algorithm is the best performing algorithm.

Table 31 Sign test data

Experiment environmental conditions	**Sensor fusion performance measures**							
	OO		EE		OE		EO	
	AdpWA1	**AFL**	**AdpWA1**	**AFL**	**AdpWA1**	**AFL**	**AdpWA1**	**AFL**
1.	6	0	6	0	1	0	6	0
2.	6	0	1	4	4	1	6	0
3.	6	0	6	0	0	6	6	0
4.	6	0	6	0	1	4	6	0
Total	**4**	**0**	**3**	**1**	**2**	**2**	**4**	**0**
Note: The values in this table indicate the number of times each algorithm outperforms the opponent.								

Table 32 Sign test results

Frequencies

		N
AFL - AdpWA1	Negative Differences(a)	13
	Positive Differences(b)	3
	Ties(c)	0
	Total	16

a AFL < AdpWA1
b AFL > AdpWA1
c AFL = AdpWA1

Test Statistics(b)

	AFL - AdpWA1
Exact Sig. (2-tailed)	.021(a)

a Binomial distribution used.
b Sign Test

7.3.5 Discussion

The evaluation method presented in this section indicates that the two best performing algorithms are the AdpWA1 and AFL. Of the two, the AdpWA1 is superior. AdpWA2, AdpWA3 and AdpWA4 have poor performances. The results indicate that the suggested enhancement procedure did not improve the performances, since the best performing algorithm did not use the enhancement procedure and the algorithms that does use it, did not appear as one of the two best performing algorithms. AdpWA1 algorithm uses type I performance measures, implying that the developed type II performance measures does not quantify the difference between two maps accurately enough, and this performance measures needs to be improved. These evaluations correspond to visual presentations of the generated maps.

Table 33 Logical sensors mapping for adaptive weighted average algorithm experiments set

	Logical sensor							Real world map
Exp.	US1	US2	LASER1	LASER2	CAM1	CAM2	CAM3	
1								
2								
3								
4								

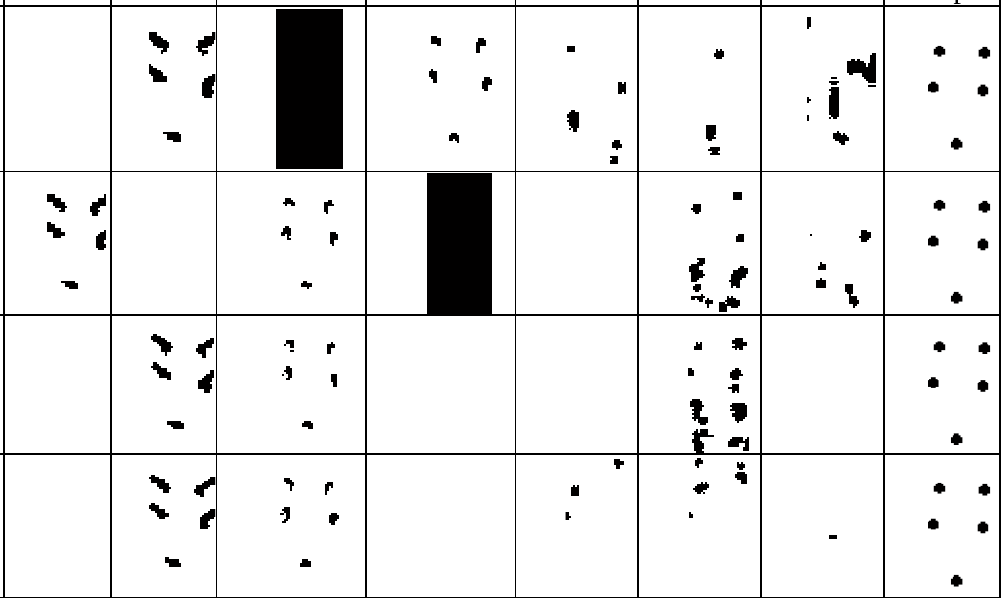
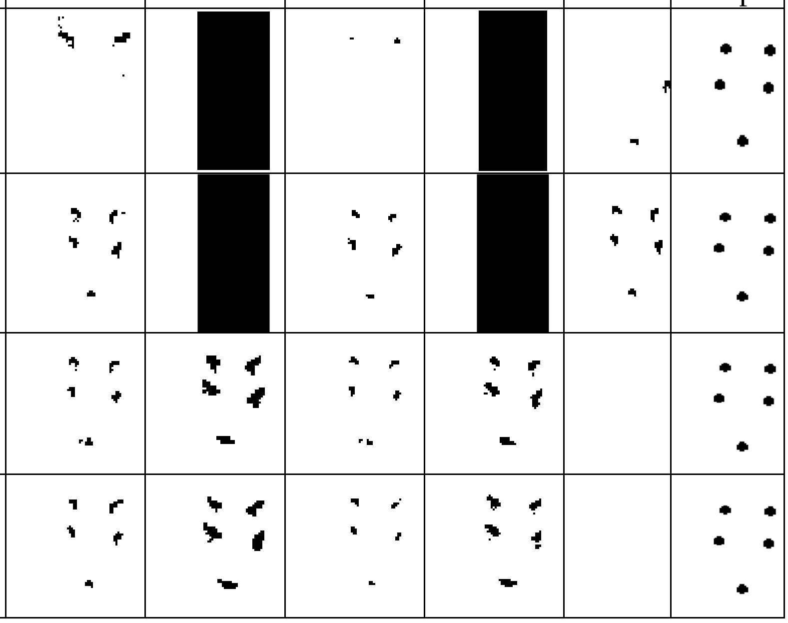


Table 34 Algorithms mapping for adaptive weighted average algorithm experiments set

	Sensor fusion algorithm					Real world map
Exp.	AdpWA1	AdpWA2	AdpWA3	AdpWA4	AFL	
1						
2						
3						
4						



8. Conclusions and future research

8.1 Conclusions

This thesis evaluates a sensor fusion framework developed in previous research for mapping the environment of a mobile robot using a grid-map representation concept. This work consists of two parts. The **first** part deals with an extended sensor fusion framework. [Cohen, 2005] sensor fusion framework was extended to fuse data from three physical sensors. The performances of the extended framework were evaluated through a statistical evaluation method. In the evaluation, four different algorithms were used to fuse the data: three logical algorithms and one adaptive algorithm. Results indicate that the adaptive algorithm is superior the logical ones by giving best results in different environmental conditions corresponding to previous results [Cohen, 2005].

The second part deals with the development of a new adaptive weighted average sensor fusion algorithm. In the process of the algorithm's development, three concepts were developed. The **first** concept is the non-binary grid map, where Cohen's binary grid-map paradigm was extended to include cells that contain integer values that indicate the number of times the sensor declared this cell as occupied (instead of binary cells, that contains only '1' and '0' values that indicates whether this cell is *occupied* or *empty*, respectively). This extension increases the amount of data in the map yielding more accurate and complete information about the robot's surroundings. The **second** concept is a new type of performance measures that was developed. This type examines changes in two corresponding cells values and detects the more accurate sensors. The new type of performance measures allows giving more weight in the fusion process to sensors with higher performances. *i.e.,* the more accurate sensor. The non-binary grid map concept allows cells with higher values (*i.e.,* the sensor declared them as *occupied* more times) to influence more on the fusion process. The **third** concept is the map enhancement procedure that was developed in order to improve maps accuracy by canceling environmental noises and sensors malfunctions. The assumption that stands in the basis of the enhancement procedure is that occupied cells that are surrounding with occupied cells are more likely to indeed contain an obstacle and therefore should be strengthened.
The new adaptive weighted average algorithm uses these three concepts when enhanced non-binary grid maps from the different logical sensors are fused to one map by considering the cells' values and the logical sensor performance measures.

The performances of the new algorithm were evaluated through the statistical evaluation method and were compared to the previously developed adaptive algorithms. Results show that the new algorithm outperforms the other algorithms, while the enhancement procedure did not affect the performances.

8.2 Future research

Several research areas remain open for future expansion of this work.

Performance measures

Type II performance measures needs to be modified. In the current definition, two identical maps do not yield maximum performance measures as should be. The performance measures can be a combination between type I and type II by considering changes in the cell's status (*i.e.*, '*Occupy*' or '*Empty*') and also changes in the cell's value. The current type II performance measures deals only in the difference between two corresponding occupied cells, and their definitions should be extended to include changes between cell's conditions, *i.e.*, corresponding cells that are marked as occupied in one map but empty in the other, and vice versa. The influence of the initial performance measures must be checked, by running simulations with several random performance measures and checking the convergence to the best performing logical sensor.

Sensors configurations

In future research, it would be beneficial to examine changes in mapping from one physical sensor only (ultrasonic or laser) as opposed to the fused map. *i.e.*, what is the different between one sensor mapping and the fusion algorithm mapping. This is important in understanding the fusion contribution and the fusion system robustness. In addition, fusion results from different sensors combinations (for example, ultrasonic and laser or camera and laser) must to examined as opposed to fusion from all available sensors in order to examine the different sensors' contribution to the fusion process.

Extended experimentation

The mobile robots experiments must be extended to include more realistic conditions with different types of interruptions such as lightning conditions or bright surfaces. Another suggestion is to examine changes in the obstacle's configuration, color or height (or all the above together) in order to check the algorithms' limitations. Checking a scenario when the robot is static and adding a random noise to the system can give new understanding about algorithm's performances. In addition, the influence of the performance measures should be tested by running different experiments.

Representation

Future research should deal with maps with uncertainty values representing the probability for an obstacle in the cell. In addition, handling three-dimensional maps can provide important additional information [Cohen, 2005]. Another direction can be to consider each cell's certainty to be 'Empty', perhaps by summing the number of times the sensor declared this cell as 'Empty'. A combination of each cell's certainty to be 'Occupy' and 'Empty' can be taken into account, and fusion based on these value can be an interesting approach.

In addition, image processing algorithms optimization for the camera different logical sensors is required.

Algorithms

Cohen's Online sensor and algorithm selection system, OLSAS [Cohen, 2005] needs to be implemented using the extended fusion framework (i.e., to fuse data from three physical sensors). In addition, the Adaptive weighted average algorithm should be integrated in the OLSAS system.

Additional applications

It can be interesting to use the fused map for other mobile robot’s applications such as navigation. Navigation through the use of the fused map and other techniques should be compared, in order to examine the fusion profits.

9. References

[1] Abidi M. A. and Gonzales R. C. 1992. Data fusion in robotics and machine intelligence, Academic Press, San Diego, CA.

[2] Amigoni, F., Gasparini, S. and Gini, M. 2006. Building segment-based maps without pose information, Proceedings of the IEEE , 94(7): 1340-1359.

[3] Arkin E. M., Fekete S. P. and Mitchell J. S. B. 2000. Approximation algorithms for lawn mowing and milling, Computational Geometry, 17(1-2): 25-50.

[4] Arras K.O., Tomaris N., Jensen B. and Siegwart R. 2001. Multisensor On-the-Fly Localization: Precision and Reliability for Applications, Robotics and Autonomous Systems, 34(2-3): 131-143.

[5] B. Solaiman, R. Debon, F. Pipelier, J.M. Cauvin and C. Roax. 1999. information fusion, application to data and model fusion for ultrasound image segmentation, IEEE transactions on biomedical engineering, 46(10):1171-1175

[6] Bank, D. and Kampke, T. 2007. High-Resolution Ultrasonic Environment Imaging, IEEE Transactions on Robotics, 23(2): 370-381.

[7] Belknap R., Riesman E. and Hanson A. 1986. The information fusion problem and rule-based hypothesis applied to complex aggregation of image events, IEEE Conference Computer Vision and Pattern Recognition: 227-234.

[8] Blum R. S., Kassam S. A. and Poor H. V. 1997. Distributed detection with multiple sensors: Part II – advanced topics, Proceedings of the IEEE, 85(1): 64-79.

[9] Borenstein, J. and Koren, Y. 1991. Histogramic in-motion mapping for mobile robot obstacle avoidance, IEEE Transactions on Robotics and Automation, 7(4):535-539.

[10] Brooks R. A. 1982. Solving the find-edge problem is good representation of free space, Proceedings AAAI-82: 381-387.

[11] Brooks, R. R. and Iyengar S. S. 1998. Multi-sensor fusion. Prentice Hall, New York, NY.

[12] Cao, A., and Borenstein, J. 2002. Experimental Characterization of Polaroid Ultrasonic Sensors in Single and Phased Array Configuration. Proceedings of the UGV Technology Conference at the SPIE AeroSense Symposium.

[13] Carson R.R., Meyer, M.P and Peters, D.J. 1996. Fusion of IFF and radar data," Data Fusion Symposium. ADFS '96, 21-22 Nov: 65-70.

[14] Castellanos J.A. and Tardos J.D. 1999. Mobile robot localization and map building: a multisensory fusion approach. Boston, MA: Kluwer Academic Publishers.

[15] Castellanos, J.A., Neira, J. and Tardos, J.D. 2001. Multisensor fusion for simultaneous localization and map building, IEEE Transactions on Robotics and Automation, 17(6): 908-914.

[16] Chen S. 1987. Multisensor fusion and navigation of mobile robots, International Journal of Intelligent Systems, 2(2): 227-251.

[17] Cohen O. 2005. Grid-Map based sensor fusion for autonomous mobile robot,Ph.d thesis, Ben-Gurion University of the Negev, Beer-Sheva p.o box 84105, ISRAEL.

[18] Costa, J., Dias, F. and Araujo, R. 2006. Simultaneous Localization and Map Building by Integrating a Cache of Features, IEEE Conference on Emerging Technologies and Factory Automation, 20-22 Sept:1036 - 1043.

[19] Cremer F., Schutte K., Schavemaker J. G. M., and E. den Breejen. 2001. A comparison of decision-level sensor-fusion methods for anti-personnel landmine detection, Information Fusion, (2)3: 187-208.
[20] Daniel F., Gamrra T., Bastos-Filho T. and Sarcinelli-Filho M. 2005. Controlling the Navigation of a Mobile Robot in a Corridor with Redundant Controllers, Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain, April 2005: 3855-3860.
[21] Davison A. J. and Kita. N. 2002. Simultaneous localization and map-building using active vision for a robot moving on undulating terrain. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7): 865 – 880.
[22] Davison, A.J. and Murray, D.W. 2002. Simultaneous localization and map-building using active vision, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7): 865-880.
[23] Durrant-Whyte H. F. 1988a. Integration, coordination and control of multisensor robot systems, Kluwer, Boston, MA.
[24] Durrant-Whyte H. F. 1988b. Sensors models and multisensor integration, The International Journal of Robotics Research, 7(6): 97-112.
[25] Elfes, A. 1987. Sonar-based real-world mapping and navigation, IEEE Journal of Robotics and Automation, 3(3):249-265.
[26] Faceli K., Andre C.P.L.F. de Carvalho and Rezende S. O. 2004. Combining intelligent techniques for sensor fusion, Applied Intelligence, 20: 199–213.
[27] Faugeras O., Vieville T., Theron E., Vuillemin J., Hotz B., Zhang Z., Moll L, Bertin P., Mathieu H., Fua P., Berry G. and Proy C. 1993. Real time correlation-based stereo: algorithm, implementations and applications, Tech. Rep. 2013,INRIA.
[28] Feng-chun Z., Yan-bing J. and Ai-hua W. 2006. Research on Integrated Navigation Technology of Field Robot, IEEE International Conference on Information Acquisition, pp.59-64.
[29] Filippidis A., Jain L.C. and Martin, N. 2000. Multisensor data fusion for surface land-mine detection, IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, 30(1):145-150.
[30] Garcia J.G., Robertsson A., Ortega J.G. and Johansson R. 2004. Sensor fusion of force and acceleration for robot force control, Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3: 3009-3014.
[31] Gon W. and Beom H. 2006. Hierarchical Sensor Fusion for Building an Occupancy Grid Map using Active Sensor Modules, Proceedings of International Joint Conference:2600 – 2605.
[32] Gonzales J., Ollero, A. and Reina, A. 1994. Map building for a mobile robot equipped with a 2D laser rangefinder, Proceedings of the IEEE International Conference on Robotics and Automation, vol. 3: 1904-1909.
[33] Groen F. C. A., Komen E. R., Vreeburg M. A. C. and Warmerdam T. P. H. 1986. Multisensor robot assembly station, Robotics, vol. 2: 205-214.
[34] Guoliang L., Wanjun H., Shizuo Y., Zengqi S. and Wenyi Q. 2006. A Fusion Algorithm for Building Maps in Confined Environments for Mobile Robots, Proceedings of IMACS Multiconference on Computational Engineering in Systems Applications:960 – 964.
[35] Hanson A. R., Risen E. M., and Williams T. D. 1988. Sensor and information fusion from knowledge-based constrains, SPIE Proceedings of Sensor Fusion. 931: 186-196.

[36] Harmon S. Y. 1986. Autonomous vehicles, Encyclopedia of Artificial Intelligence, John Wiley, New York: 39-45.
[37] Harris C.G. and Stephens M. 1988. A combined corner and edge detector. Proceedings of Fourth Alvey Vision Conference: 147-151.
[38] Hebert M. 2000. Active and passive range sensing for robotics, Proceedings of IEEE International Conference on Robotics and Automation, vol. 1: 102-110.
[39] Henderson T. and Shilcrat E. 1984. Logical sensor system, Journal of Robotic Systems, 1(2): 169-193.
[40] Hernandez A.I., Carrault G., Mora, F., Thoraval L., Passariello G. and Schleich J.M. 1999. Multisensor fusion for atrial and ventricular activity detection in coronary care monitoring, IEEE Transactions on Biomedical Engineering, 46(10):1186-1190.
[41] Hernandez A.I., Carrault G., Mora F., Thoraval L., Passariello G. and Schleich J.M., 1999. Multisensor fusion for atrial and ventricular activity detection in coronary care monitoring, IEEE Transactions on Biomedical Engineering, 46(10):1186-1190.
[42] Hollander M. and Wolfe D. A. 1973. Nonparametric statistical methods, John Wiley & Sons Inc, New York, NY.
[43] Hong W., Tian Y. and Dong Z. 2002. The approach of extracting features from the local environment for mobile robot, Proceedings of the First International Conference on Machine Learning and Cybernetics: 611-616.
[44] Hong-Ming W., Zeng-Guang H., Jia M., Yun-Chu Z., Yong-Qian Z. and Min T. .2007. Sonar Feature Map Building for a Mobile Robot, Proceedings of IEEE International Conference on Robotics and Automation: 4152 – 4157.
[45] HoseinNezhad R., Moshiri B. and Asharif M. R. 2002. Sensor fusion for ultrasonic and laser arrays in mobile robotics. Proceedings of IEEE International Conference on Sensors: 1682-1689.
[46] Huntsberger T. L. and Jayaramamurthy S. N. 1987. A framework for multisensor fusion the presence of uncertainty, Proceedings of Workshop Spatial Reasoning and Multisensor Fusion: 345-350.
[47] Hyeyeon C., JongSuk C. and Munsang K. 2006. Experimental research of probabilistic localization of service robots using range image data and indoor GPS system, Proceeding of IEEE Conference on Emerging Technologies and Factory Automation:1021 – 1027.
[48] Ivanjko E., Vasak M. and Petrovic I. 2005. Kalman filter theory based mobile robot pose tracking using occupancy grid maps, Proccedings of International Conference on Control and Automation: 869 – 874.
[49] Kamat S. J. 1985. Value function structure for multiple sensor integration, Proceedings of SPIE, Intelligence Robots and Computer Vision. 579: 432-435.
[50] Karaman, O. and Temeltas, H. 2004. Comparison of different grid based techniques for real-time map building, Proceedings of IEEE International Conference on Industrial Technology,vol. 2: 863-868.
[51] Kim G.W and Lee B.H. 2006. Hierarchical Sensor Fusion for Building an Occupancy Grid Map using Active Sensor Modules, Proceedings of the International Joint Conference SICE-ICASE: 2600-2605.
[52] Klein L. A. 1993. A Boolean algebra approach to multiple sensor voting fusion, IEEE Transactions on Aerospace and Electronic Systems, 29(2): 317-327.
[53] Kluge, K.C. 2003. SAMLOS: a 2D simultaneous localization and mapping algorithm based on lines of sight, Proceedings of IEEE Intelligent Vehicles Symposium: 438-443.

[54] Kwon, Y.D. and Lee, J.S., 1997. A stochastic environment modelling method for mobile robot by using 2-D laser scanner, Proceedings of the IEEE International Conference on Robotics and Automation, vol.2: 1688-1693.
[55] Li X., Huang X., Wang M. and Peng G. 2006. A Comparison of the effect of sonar grid map building based on dsmt and dst, Proceedings of the 6th World Congress on Intelligent Control and Automation, vol. 1: 4073:4077.
[56] Lin H. H., Tsai C. C., Hsu J. C. and Chang C. F. 2003. Ultrasonic self localization and pose tracking of an autonomous mobile robot via fuzzy adaptive extended information filtering, Proceedings of the IEEE International Conference on Robotics and Automation: 1283-1290.
[57] Liu G., Haot W., Yant S. , Sun2 Z. and Qiangt W. 2006. A Fusion Algorithm for Building Maps in Confined Environments for Mobile Robots, IMACS Multiconference on Computational Engineering in Systems Applications, vol. 1:960-964.
[58] Lozano-Perez T. 1981. Automatic planning of manipulator transfer movements, IEEE Transactions, on Systems Man and Cybernetics: 781-798.
[59] Lu Y., Zeng L. and Bone G.M. 2005. Multisensor System for Safer Human-Robot Interaction, Proceedings of the 2005 IEEE International Conference on Robotics and Automation: 1767-1772.
[60] Luo R. C. and Kay M. G. 1989. Multisensor integration and fusion in intelligent systems, IEEE Transactions on Systems, Man. and Cybernetics, 19(5): 901-931.
[61] Luo, R.C., Chih-Chen Yih and Kuo Lan Su. 2002. Multisensor fusion and integration: approaches, applications, and future research directions, IEEE Sensors Journal, 2(2):107-119.
[62] Luo K. and Lin C. 1996. An intelligent sensor fusion system for tool monitoring on a machining center. Proceedings of the International Conference of Multisensor Fusion Integration Intelligence Systems. pp. 208-214.
[63] Martin M.C. and Moravec H., Robot Evidence Grids, tech. report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, March, 1996.
[64] Metz CE. 1986. Statistical analysis of ROC data in evaluating diagnostic performance. In: Multiple Regression Analysis: Applications in the Health Sciences (D Herbert and R Myers, eds.). New York: American Institute of Physics: 365-384.
[65] Mirzaei, F.M., Mourikis, A.I. and Roumeliotis, S.I., On the Performance of Multi-robot Target Tracking, Proceedings of the IEEE International Conference on Robotics and Automation, pp.3482-3489.
[66] Miura, J., Negishi, Y. and Shirai, Y. 2002. Mobile robot map generation by integrating omnidirectional stereo and laser range finder, Proceedings of IEEE International Conference on Intelligent Robots and System, pp. 250-255.
[67] Moravec H. P. and Elfes A. E. 1985. High resolution maps for wide-angle sonar, Proceedings of the IEEE International Conference on Robotics and Automation: 116-121.
[68] Moravec H.P. 1988. Sensor fusion in certainty grids for mobile robots. Al Magazine:61-74.
[69] Moravec, H. and Elfes, A..1985. High resolution maps from wide angle sonar, Proceedings of IEEE International Conference on Robotics and Automation:116-121.
[70] Najjaran H. and Goldenberg A. 2006. Real-time motion planning of an autonomous mobile manipulator using a fuzzy adaptive Kalman filter, Robotics and Autonomous Systems, 55(2): 96-106

[71] Oriolo G., Ulivi G. and Vendittelli M. 1997. Fuzzy maps: A new tool for mobile robot perception and planning, Journal of Robotic Systems, 14(3): 179-197.
[72] Patel K., Macklem W., Thrun S. and Montemerlo M. 2005. Active Sensing for High-Speed Offroad Driving, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3162-3168.
[73] Perez-Lorenzo J.M., Vazquez-Martin R., Nunez P., Perez E.J. and Sandoval F. 2004. A Hough-based method for concurrent mapping and localization in indoor Proceedings of IEEE Conference on environments, Robotics, Automation and Mechatronics, vol. 2: 840-845.
[74] Reina A. and Gonzales J. 1997. Characterization of a radial laser scanner for mobile robot navigation, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol.2: 579-585.
[75] Ribo M. and Pinz A. 2001. A comparison of three uncertainty calculi for building sonar based occupancy grids, International Journal of Robotics and Automation Systems 35: 201-209.
[76] Schwering P. B. W., Baertlein B.A., Van den Broek S.P. and Cremer F. 2002. Evaluation methodologies for comparison of fusion algorithms in land mine detection, Detection and Remediation Technologies for Mines and Minelike Targets VII, Proc. SPIE, 4742: 847-856.
[77] Solaiman B., Pierce L.E. and Ulaby F.T. 1999. Multisensor data fusion using fuzzy concepts: application to land-cover classification using ERS-1/JERS-1 SAR composites, IEEE Transactions on Geoscience and Remote sensing. 37(3):1316-1326.
[78] Solaiman B., Koffi R.K., Mouchot M.-C., Hillion A. 1998. An information fusion method for multispectral image classification postprocessing, IEEE Transactions on Geoscience and Remote Sensing, 36(2):395-406.
[79] Stepan P., Kulich M. and Preucil L. 2005. Robust data fusion with occupancy grid, IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, 35(1):106-115.
[80] Stepan P., Kulich M. and Preucil L. 2005. Robust Data Fusion With Occupancy Grid, IEEE Transactions on Systems, Man and Cybernetics, 35(3): 106-115
[81] Sukumar S.R., Bozdogan H., Page D.L., Koschan A.F. and Abidi M.A. 2007. Sensor Selection Using Information Complexity for Multi-sensor Mobile Robot Localization. Proceedings of the IEEE International Conference on Robotics and Automation, pp.4158-4163.
[82] T.E. Bell, 1995. Remote sensing, IEEE spectrum, 32(3): 24-31.
[83] Tanaka K., Okada N. and Kondo E. 2003. Building a floor map by combining stereo vision and visual tracking of persons, Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, vol.2: 641-646.
[84] Thomas U., Molkenstruck S., Iser R. and Wahl F.M. 2007. Multi Sensor Fusion in Robot Assembly Using Particle Filters, Proceedings of the IEEE International Conference on Robotics and Automation, pp.3837-3843.
[85] Thomas U., Molkenstruck S., Iser R. and Wahl F.M. 2007. Multi Sensor Fusion in Robot Assembly Using Particle Filters, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3837-3843.
[86] Thomopoulos S. C. A., Viswanathan R. and Bougoulias D. K. 1987. Optimal decision fusion in multiple sensor systems, IEEE Transactions on Aerospace and Electronic Systems, 23(5): 644-652.

[87] Tin K. H. and Mitra B. 2002. Complexity measures of supervised classification problems, IEEE Transactions on Pattern analysis and machine intelligence 24(3): 289-300.
[88] Toledo, F.J., Luis, J.D., Tomas, L.M., Zamora, M.A. and Martinez, H. 2000. Map building with ultrasonic sensors of indoor environments using neural networks, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, vol. 2: 920-925.
[89] Tomoto M. 2005. environment modeling by a mobile robot with a laser range finder and a monocular camera, Proceedings of the IEEE Workshop on Advanced Robotics and its Social Impacts, pp. 133-138.
[90] Vandorpe J., Van Brussel H. and Xu H. 1996. Exact dynamic map building for a mobile robot using geometrical primitives produced by a 2D range finder, Proceedings of IEEE International Conference on Robotics and Automation, pp. 901-908.
[91] Vazquez J. and Malcolm C. 2005. Fusion of triangulated sonar plus infrared sensing for localization and mapping, Proceedings of the International Conference on Control and Automation, pp. 1097 – 1102.
[92] Wooden D. 2006. A guide to vision-based map building, IEEE Robotics and Automation Magazine, 13(2): 94-98.
[93] Xinde L., Xinhan H., Min W. and Gang P. 2006. A Comparison of the Effect of Sonar Grid Map Building Based on DSmT and DST, Proceedings of the The Sixth World Congress on Intelligent Control and Automation, pp. 4073 – 4077.
[94] Xue-Cheng L., Cheong-Yeen K., Shuzhi S. G. and Al M. A. 2005. Online map building for autonomous mobile robots by fusing laser and sonar data, Proceedings of the IEEE International Conference Mechatronics and Automation, pp.993 – 998.
[95] Ye C. and Borenstein, J. 2002. Characterization of a 2D laser scanner for mobile robot obstacle negotiation, Proceedings of IEEE International Conference on Robotics and Automation ,vol.3: 2512-2518.
[96] Zhen J., Balasuriya A. and Challa S. 2005. Sensor fusion based 3D Target Visual Tracking for Autonomous Vehicles with IMM, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1841-1846.
[97] Kodagoda, S., Hemachandra, E. A. S. M., Jayasekara, P. G., Peiris, R. L., De Silva, A. C. and Munasinghe, R. 2006. Obstacle Detection and Map Building with a Rotating Ultrasonic Range Sensor using Bayesian Combination, Proceedings of the International Conference on Information and Automation, 98-103.

10. Appendices

Appendix I Robot and laser– specifications and parameters

Table 35 Pioneer 2 AT Specifications
(adapted from Pioneer 2 manual)

Physical Characteristics	
Length (cm)	50
Width (cm)	49
Height (cm)	24
Clearance (cm)	5.5
Weight (kg)	14
Payload (kg)	40
Power	
Batteries 12VDC lead-acid	3
Charge (watt-hrs)	252
Run time (hrs)	4-6
with PC (hrs)	2-3
Recharge time hr/battery std charger	6
High-Speed (3 batteries)	2.4
Mobility	
Wheels	4 pneumatic
diam (mm)	220
width (mm)	75
Caster (mm)	na
Steering	Skid
Gear ratio	85.2:1
Swing (cm)	40
Turn (cm)	0 cm
Translate speed max (mm/sec)	700
Rotate speed max (deg/sec)	140
Traversable step max (mm)	89
Traversable gap max (mm)	127
Traversable slope max (grade)	40%
Traversable terrains	Unconsolidated No carpets!
Sensors	
Sonar Front Array (one each side, six forward @ 20° intervals)	8
Sonar Front Array (one each side, six forward @ 20° intervals)	8
Rear Sonar Array (one each side, six rear @ 20° intervals)	8
Top Deck Sonar (one each side, six rear @ 20° intervals)	na
Encoders (2 ea) counts/rev	34,000
counts/mm	49
counts/rotation	22,500

Microcontroller and Console Controls & Ports

Siemens 8C166 (20 MHz)	Processor
32 characters on 2 lines	LCD
4	Encoder inputs
Piezo buzzer	Audio
8 (4 user-available)	PWM outputs
2x8 (multiplexed)	Sonar inputs
16 logic ports; 8 in, 8 out	Digital I/O
5 @ 0-5 VDC; 1024- or 256-bit resolution	A/D
8 @ 1µsec resolution;	Digital timers
32 KB; P2OS and robot-specific parameters	FLASH PROM
32 KB	RAM
1 main; 1 RADIO	Power switches
2 RS-232 serial internal; 1 RS-232 external	Comm ports
12 VDC @ 1A switched; 5 VDC @ 3Aswitched	Power (internal comm. ports)
RESET and MOTORS	Logic pushbuttons
Main power; RADIO power; Host SERIAL RxD and TxD	Indicator LEDs

Table 36 Laser's technical data
(Adapted from laser's manual)

Laser Measurement Sensors

Indoor

Model Name LMS 200-30106
Part Number 1015850

Technical data	
Field of view:	180 °
Angular resolution:	1 ... 0.25 °
Response time:	13 ... 53 ms
Resolution:	10 mm
Systematic error:	+/- 15 mm
Statistical error (1 sigma):	5 mm
Laser class:	1
Enclosure rating:	IP 65
Ambient operating temperature:	0 °C ... +50 °C
Scanning range:	80 m
Data interface:	RS-232, RS-422
Data transmission rate:	9,6 / 19,2 / 38,4 / 500 kBaud
Switching outputs:	3 x PNP
Supply voltage:	24 V DC +/- 15%
Power consumption:	20 W
Storage temperature:	-30 °C ... +70 °C
Weight:	4.5 kg
Dimensions (L x W x H):	156 x 155 x 210 mm

Table 37 Pioneer 2 AT parameters
(adapted from Pioneer 2 manual)

```
;;
;; Parameters for the Pioneer 2 AT Mobile Robot (adapted from the Pioneer Manual)
;;
AngleConvFactor 0.001534 ;radians per angular unit (2PI/4096)
DistConvFactor 1.303 ; mm returned by P2
VelConvFactor 1.0 ; mm/sec returned by P2
RobotRadius 500.0 ; radius in mm
RobotDiagonal 120.0 ; half-height to diagonal of octagon
Holonomic 1 ; turns in own radius
MaxRVelocity 300.0 ; degrees per second
MaxVelocity 1200.0 ; mm per second
RangeConvFactor 0.268 ; sonar range returned in mm
;;
;; Robot class, subclass
;;
Class Pioneer
Subclass p2at
SonarNum 16 ; 16 total sonars
;; These are for the eight front sonars: six front, two sides
;;
;; Sonar parameters
;; SonarNum N is number of sonars
;; SonarUnit I X Y TH is unit I (0 to N-1) description
;; X, Y are position of sonar in mm, THETA is bearing in degrees
;;
;;          #  X    Y  THETA
;;-------------------------
SonarUnit 0 145 130 90
SonarUnit 1 185 115 50
SonarUnit 2 220 80 30
SonarUnit 3 240 25 10
SonarUnit 4 240 -25 -10
SonarUnit 5 220 -80 -30
SonarUnit 6 185 -115 -50
SonarUnit 7 145 -130 -90
;; These are for the eight rear sonars: six back, two sides
;;          #  X    Y  THETA
;;----------------------------
SonarUnit 8 -145 -130 -90
SonarUnit 9 -185 -115 -130
SonarUnit 10 -220 -80 -150
SonarUnit 11 -240 -25 -170
SonarUnit 12 -240 25 170
SonarUnit 13 -220 80 150
SonarUnit 14 -185 115 130
SonarUnit 15 -145 130 90
;; Number of readings to keep in circular buffers
FrontBuffer 20
SideBuffer 40
```

Appendix II ARIA API

This appendix is based on ARIA version 2.4.0 manual, downloaded from the Activemedia website (http://www.activrobots.com/SOFTWARE/aria.html).

ARIA is an object-oriented, application-programming interface for ActivMedia Robotics' line of intelligent mobile robots, including Pioneer, Pioneer 2/3, PeopleBot, PowerBot, and AmigoBot mobile robots. Written in the C++ language, ARIA is client-side software for easy, high-performance access to and management of the robot server, as well as to the many accessory robot sensors and effectors. Its versatility and flexibility makes ARIA an excellent foundation for higher-level robotics applications.

ARIA can be run multi- or single-threaded, using its own wrapper around Linux pthreads and WIN32 threads. Use ARIA in many different ways, from simple command-control of the robot server for direct-drive navigation, to development of higher-level intelligent actions (behaviors).

At its heart, ARIA's **ArRobot** class collects and organizes the robot's operating states, and provides clear and convenient interface for other ARIA components, as well as upper-level applications, to access that robot state-reflection information for assessment, planning, and ultimately, intelligent, purposeful control of the platform and its accessories. Figure 15 presents ARIA's schematic architecture.

ARIA also includes clear and convenient interface for applications to access and control ActivMedia Robotics accessory sensors and devices, including operation and state reflection for sonar and laser range finders, pan-tilt units, arms, inertial navigation devices, and many others.
The versatility and ease of access to ARIA code (sources included!) makes it the ideal platform for robotics client applications development.

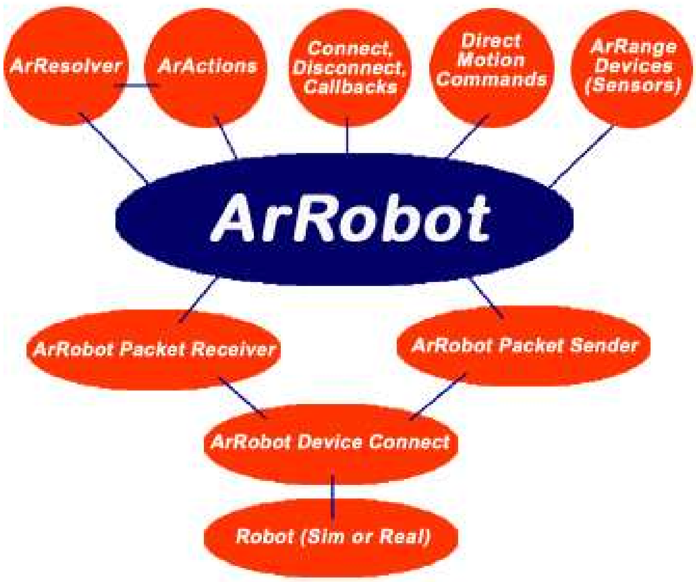


Figure 15 ARIA's schematic architecture

(Adapted from ARIA 2.4.0 manual)

Appendix III Modifications to the research of Cohen [Cohen, 2005]

In this research, Cohen's PhD work [Cohen, 2005] was extended and modified.

- A new type of performance measures (type II) was developed and implemented (section 4.2).
- The grid map paradigm that was implemented in Cohen's work was extended from a binary grid map (where each cell has to possible values: '1' represents that the cell is Occupied or '0' represents that the cell is Empty) to a non-binary grid map (where cells contains an integer value that represents the number of time the logical sensor decided that this cell is occupied). The extension was carried out by code modifications in all the mapping functions. In addition, Cohen's sensor fusion algorithms were modified to fit the new concept. The extension enables to consider the values within the cells, and was the base upon the type of performance measure and the new algorithms were developed.
- A new adaptive sensor fusion algorithm was developed and implemented – adaptive weighted average (section 5.2) and it's performances was compared to previous algorithms using Cohen's statistical evaluation procedure (section 7.3).
- Cohen's original code was developed using robot's interface *Saphira*. In this work, the robot's API was changed to ARIA. The *Saphira* architecture is designed to operate with a robot server, that is, a mobile robot platform that provides a set of robotic services in a standard format. ARIA is a newer and powerful robot's interface that replaces *Saphira* .As a result, the complete system's code was re-programmed using the new API. ARIA API is detailed in Appendix II.
- The robot's system was extended by adding a third physical, a laser range finder (Laser specifications are detailed in Appendix I). The laser sensor was implemented using two logical sensors that were created through two new mapping algorithms that were developed (see section 6.3). The system's code was adjusted to include three physical sensors (instead of two) and seven logical sensors (instead of five). All the matrices and variables were changed to fit the extended system. The adjusted code is detailed in Appendix VII. The algorithms performances were evaluated using the new physical sensor.
- The code for the sampling the camera was modified in order to prevent time lag and enhance the system performances. In Cohen's work, the camera took pictures in four pan angles at a constant sequence: -17°→17°→50°→-50°. Due to the camera's structure, a lot of time was wasted in shifting the camera's pan angle from 50° to -50°. This procedure was enhanced by using different sequences to odd and even cycles. In even cycles, the camera samples in the following order: -50°→-17°→17°→50°, and in odd cycles the sampling order is: 50°→17°→-17→-50°. The change required code modification, and caused a higher number of cycles during the robot's course (38 instead of 23), as a result from the increase in the number of cycles, the sensors map are more accurate.
- The laser sensor was placed on the robot's base, and the camera is mounted on top of the laser. As a result, the camera's new location is higher than the previous location by 20 cm. this influenced the camera calibration parameters, and a new calibration was performed (see Appendix V).
- The lab for the experiments was changed due to technical and administrative changes, which caused difference in the lightning and environmental conditions. Hence, the image processing algorithms were adjusted to fit the new conditions.

Appendix IV Software code

The experiments software is written in VC++™ (version 6.0), using ARIA™ (version 6.4) library routines, under Windows™ 2000 (version 4.0).
The image processing functions that were used are taken from Intel's OpenCV and IPL (manuals can be found at http://www.intel.com/technology/computing/opencv/ and http://www.intel.com/software/products/perlib/ipl/iplrelnotes_test.htm).

The system consists of the following files.

System files	
ConstantParameters.h	Contains the system constant parameters.
GlobalParameters.h	Contains the system global parameters.
StaticParameters.h	Contains the system static parameters.
main.cpp	This file starts the ARIA™ and the program.
InitiationFile.h ***InitiationFile.cpp***	These files contain the functions to initialize the parameters and arrays and check the system for errors within the constant and global parameters.
LogicalSensor.h ***LogicalSensor.cpp***	These files contain all the information related to logical sensors (*e.g.*, sensor fusion algorithms and transformations).

Camera and image processing algorithms files	
PXC_Camera_Dll_Load.h ***PXC_Camera_Dll_Load.cpp***	These files contain the information related to the camera and the PXC200 frame grabber.
Vision_Class.h ***Vision_Class.cpp***	These files contain the information related to the image processing algorithms.

Ultrasonic files	
UltraSonic_Class.h ***UltraSonic_Class.cpp***	These files contain all the information related to the physical ultrasonic sensors (*e.g.*, algorithms transformations).

Laser files	
Sick_Class.h ***Sick_Class.cpp***	These files contain all the information related to the physical laser sensor (*e.g.*, algorithms transformations).

Fuzzy Logic algorithm files	
FuzzyLogic_Algorithm.h ***FuzzyLogic_Algorithm.cpp***	These files contain the information related to the fuzzy logic algorithms (adaptive and logical).

Figure 16 presents a schematic diagram of main program information flow.

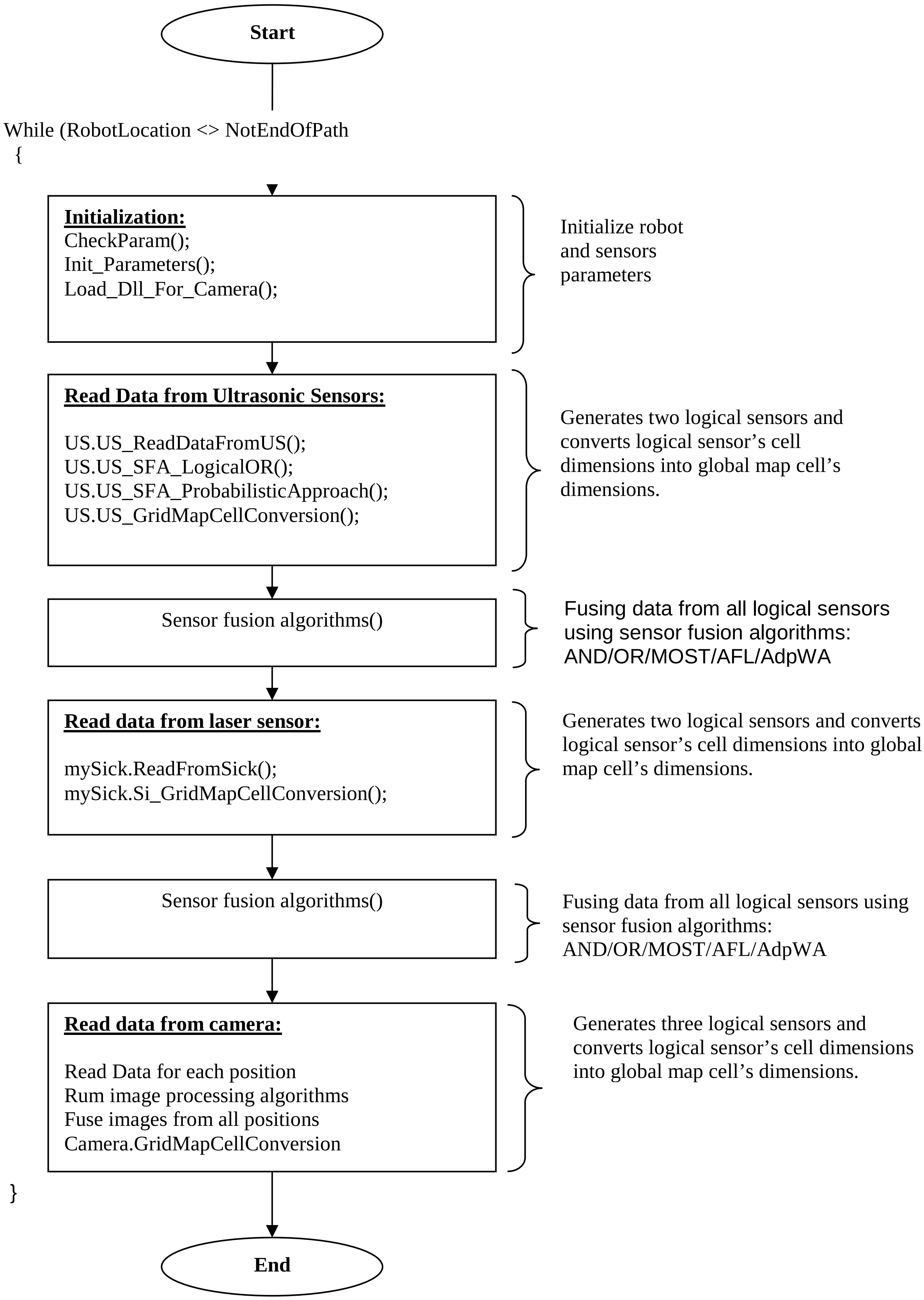


Figure 16 Schematic diagram of main program flow

Table 38 contains a list of the main functions in the system's files, their types and a brief explanation about their purpose. The table is followed by the full system code.

Table 38 list of functions and explanations

File	Function type	Function name	Explanations
collector.cpp	int	main	This function is the main of the sensor fusion system. The robot moves at a straight line while fusing information from the physical sensors. Schematic diagram of the program flow presented in Figure 16. The function returns 0 when it ends.
	void	initDLL	Initializing the camera's dll files
		Init_Parameters	This function initializes the parameters and sets all array to zeros
LogicalSensor.cpp	void	SFA_AND	This function fuses the data between all LBMs using the AND method
		SFA_OR	This function fuses the data between all LBM using the OR method
		SFA_REGULAR_MOST	This function fuses the data between all LBM using the regular MOST method.
		SFA_REGULAR_AFL	This function fuses the data between all LBM using the regular AFL method
		SFA_AdpWA1	This function fuses the data between all LBM using AdpWA1 algorithm, which means without Enhancement, binary PM.
		SFA_AdpWA2	This function fuses the data between all LBM using AdpWA1 algorithm, which means without Enhancement, New PM.
		SFA_AdpWA3	This function fuses the data between all LBM using AdpWA1 algorithm, which means with Enhancement, Binary PM.
		SFA_AdpWA4	This function fuses the data between all LBM using AdpWA1 algorithm, which means without Enhancement, new PM.
		CopyLBM2GGM(int)	This function copies the local binary maps (LBMs) to the global grid maps.
		CreateLS_PPGM(int)	This function creates the PPGM matrix for each LS using the LBM

		SaveGGM	This function saves all the data into the hard disk.
		Call_LS_Func	This function is used to fuse the data using the all algorithms methods
		Calculating_FL_TruthTable(int)	Calculating the truth table
		FuzzyLogicAlgorithm(int)	This function is an algorithm base on the FL theory for fusing the data.
		CalculatingTrueAndFalseValues(int)	This function compares the new data at this level with the integrated data This function is the adaptive part of the system and determine the following parameters SFS_True_False: The Local Map Found True But the fused map determined False SFS_True_True: The Local Map Found True And the fused map determined True SFS_False_False: The Local Map Found False And the fused map determined False SFS_False_True: The Local Map Found False But the fused map determined True
		SFA_Calc_PM(int)	This function calculates for each LS its reliability according to the generated map by each algorithm
		LGM_Transformation	This function transforms the logical sensors' maps that were not update
PXC_Camera_Dll_Load.cpp	bool	AppInit	This function initializes and allocates the Frame grabber PXC200.
	void	ImageProcessingAlgo1	The function has three steps:1. Capturing the image.2. Image processing algorithm (has two stages).2.1 Simple Threshold. 2.2 Two level threshold.3. Finding the center of mass (COM) for each obstacle, and calculate the real distance from the camera.
Vision_Class.cpp	void	ImageProcessingAlgo3(int)	The heart of the image processing, here we do the Erode Dilate for each photo according to the algorithm number, We find the center of mass for each algorithm and finds the location of the algorithm according to the calibration process made earlier.

		ImageProcessingAlgo4(int)	This function transforms the maps and built for each obstacle a circle around it.
		Vision_GridMapCellConversion	This function converts the maps into a one grid cell size.
UltraSonic_Class.cpp	void	US_ReadDataFromUS	This function reads the data form the sonar.
		US_SFA_LogicalOR	This function fuse the data between the physical US sensors based on the OR method.
		US_SFA_ProbabilisticApproach	This function fuses the data between the physical US sensors based on the algorithm which is based on the paper of Miguel Ribo and Axel Pinz, 2001.
		US_GridMapCellConversion	This function converts cell size from US to LBM.
FuzzyLogic_Algorithm.h	FuzzyLogic	FL_Crisp2Fuzzy	This function: FL_Crisp2Fuzzy calculate the FUZZY value for each crispy value.
		operator>>(FuzzyLogic &FL_Source1, FuzzyLogic &FL_Target1)	This Operator: >> Means 'Then' at the IF....THEN fuzzy rules
		operator+(const FuzzyLogic &FL1,const FuzzyLogic &FL2)	This Operator: + Means 'OR' at the IF....THEN fuzzy rules
		operator*(const FuzzyLogic &FL1,const FuzzyLogic &FL2)	This Operator: * Means 'AND' at the IF....THEN fuzzy rules
Sick_Class.h	void	ReadFromSick	This function reads the data from the laser sensor and generates two logical sensors from this data.
		Si_GridMapCellConversion	This function converts cell size from laser to LBM.

```
/**
** ConstantParameters.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail: oprc@bgumail.bgu.ac.il
**
**/

#ifndef __ConstantParameters_h__
#define __ConstantParameters_h__


const unsigned short g_usNumOfCamPos=4;  // Total number of Cam positions
const int g_TotalNumberOfAlgorithms=6; // OR, AND, OLSAS, MOST, AFL(STAM), AFL
const double g_pi=3.1415926535;

const int LBM_cm_SizeX=140; // Local Binary Map (LBM) X Direction (Forward)
const int LBM_cm_SizeY=240; // Local Binary Map (LBM) Y Direction (Side)
const int PPGM_cm_SizeX=800; // Path Planning Grid Map (PPGM) X Direction (Forward)
const int PPGM_cm_SizeY=LBM_cm_SizeY;

const int g_iTotalNumOfCamLS=3; // Total number of camera LSs in the system
const int g_CamCellSize=5; //cell size in camera local grid map [cm]
const int g_CamGridSizeX=LBM_cm_SizeX/g_CamCellSize; //number of cells in camera local grid map - X axis
const int g_CamGridSizeY=LBM_cm_SizeY/g_CamCellSize; //number of cells in camera local grid map - Y axis

const int g_iTotalNumOfUsLS=2;
const int g_USCellSize=10; //cell size in US local grid map [cm]
const int g_USGridSizeX=LBM_cm_SizeX/g_USCellSize; //number of cells in US local grid map - X axis
const int g_USGridSizeY=LBM_cm_SizeY/g_USCellSize; //number of cells in US local grid map - Y axis

const int g_iTotalNumOfSiLS=2;  // Total number of Sick LSs in the system
const int g_SickCellSize=5; //cell size in camera local grid map [cm]
const int g_SickGridSizeX=LBM_cm_SizeX/g_SickCellSize; //number of cells in camera local grid map - X axis
const int g_SickGridSizeY=LBM_cm_SizeY/g_SickCellSize; //number of cells in camera local grid map - Y axis


const int g_iTotalNumOfLS=g_iTotalNumOfCamLS+g_iTotalNumOfUsLS+g_iTotalNumOfSiLS; // Toatal number
fo LSs in the system
const int g_LBMCellSize=5; // cell size in LBM [cm]
const int g_iX_LBM_MapSize=LBM_cm_SizeX/g_LBMCellSize; // Number of cells of the LBM X direction
const int g_iY_LBM_MapSize=LBM_cm_SizeY/g_LBMCellSize; // Number of cells of the LBM Y direction

const int g_PPGMCellSize=g_LBMCellSize; // cell size in PPGM [cm]
const int g_iX_PPGM_MapSize=PPGM_cm_SizeX/g_PPGMCellSize;
const int g_iY_PPGM_MapSize=PPGM_cm_SizeY/g_PPGMCellSize;

const int g_iMaxNumOfObstacle=60; // Max number of obstacle

#endif
```

```
/**
** GlobalParameters.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail: oprc@bgumail.bgu.ac.il
**
**/
#include "ConstantParameters.h"

#ifndef __GlobalParameters_h__
#define __GlobalParameters_h__


 struct BlackBoard // Declare g_BlackBoard Structure
{
	// **************  System parameters **************
	int iCycle;				// system cycle counter
	int iLBM_X_Old;			// LBM old X Location
	int iLBM_Y_Old;			// LBM old Y location
	int iLBM_Theta_Old;		// LBM old Theta angle [Deg]

	int iLBM_X_New;			// LBM new X Location
	int iLBM_Y_New;			// LBM new Y location
	int iLBM_Theta_New;		// LBM new Theta angle [Deg]

	int iPPGM_X;			// PPGM X Location
	int iPPGM_Y;			// PPGM Y location
	int iPPGM_Theta;		// PPGM Theta angle [Deg]

	bool bLGM_NewDataFlag[g_iTotalNumOfLS+1]; // Flag to determine if the LBM has been updated

	float faRobotPos[100][2];		// Robot positions based on encoders X, Y

	int iaPI[g_iX_PPGM_MapSize][g_iY_PPGM_MapSize]; // Array which counts how many times each cell
has been sampled

	// Array which saves all algorithms PPGMs
	int iaPPGM[g_iX_PPGM_MapSize][g_iY_PPGM_MapSize][g_TotalNumberOfAlgorithms+1];
	/*
	Level 0 - OR
	Level 1 - AND
	Level 2 - OLSAS
	Level 3 - MOST
	Level 4 – EMPTY
	Level 5 - AFL
	*/
	int iaLS_PPGM[g_iX_PPGM_MapSize][g_iY_PPGM_MapSize][g_iTotalNumOfLS+1];

	// iaLBM: Local Binary Map that includes the all LGMs contains the fused map at level 0
	int iaLBM[g_iX_LBM_MapSize][g_iY_LBM_MapSize][1+g_iTotalNumOfLS];

	float fFL_TruthTable[64]; //unique array for the FL algorithm (NOT for Adaptive algorithm)
	float faTrueFalseRegular[(1+g_iTotalNumOfLS)][7]; // For the regular algorithm

	float faTrueFalse[(1+g_iTotalNumOfLS)][7];
```

```
/* the calculated data from the CalculatingTrueAndFalseValues function entered to this array.
	Explanation about the BB_faTrueFalse[(1+g_NumberOfModules)][7] array:
	Cell number 0 is for: Free
	Cell number 1 is for: TT Value
	Cell number 2 is for: FF Value
	Cell number 3 is for: TF Value
	Cell number 4 is for: FT Value
	Cell number 5 is for: TRUE Value
	Cell number 6 is for: FALSE Value*/

	float fTrueAccuracy[(1+g_iTotalNumOfLS)];			// the 'True' value for each sensor
	float fFalseAccuracy[(1+g_iTotalNumOfLS)];			// the 'False' value for each sensor
	float fTTValue[(1+g_iTotalNumOfLS)][64];
/*
In this table we enter the results 'WHAT WOULD BE THE CELL VALUE' IF (FOR EXAMPLE) sensor number1
	'says' 'T' number three and four says 'F' etc.
	Explanation about: BB_fTTValue[7][64].
	Cell number 0 is for the calculated Value.
	Cell number 1 is for the LS number 1.
	Cell number 2 is for the LS number 2.
	Cell number 3 is for the LS number 3.
	Cell number 4 is for the LS number 4.
	Cell number 5 is for the LS number 5.
	Cell number 6 is for the LS number 6.
	*/

	int iaLogicalSensorMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfLS][50]; // Array
that saves all the LSs maps during the experiment for future research

	int SFAOutput[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfLS+1][50];

	/*
	Explanations for the fSFA_PM 4D array:[i][j][k][l]
	i - stands for maximum number of cycles
	j - stands for number of 5 SFA (,i.e.,, OR/0/, AND/1/, OLSAS /2/, MOST_REGULAR /3/,AFL/5/)
	k - stands for PM: TT, FF, TF, FT, Fused measure(0.5*(TT+FF-TF-FT))
	l - stands for total number of LSs
	*/
	float fSFA_PM[100][6][5][g_iTotalNumOfLS];

	/*
Explanations for the fSFA_FL 3D array:[i][j][k]
	i - stands for maximum number of cycles
	j - stands for number of 5 SFA (,i.e.,, OR/0/, AND/1/, OLSAS /2/, MOST_REGULAR /3/,AFL/5/)
	k - stands for PM: TT, FF, TF ,FT
	*/

	float fSFA_FL_Regular[100][5][4];			// Regular FL algorithm

	//**************************************************
	//Adpative weighted average algorithm
	//**************************************************
	int AvgMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][60];
	int EnLSMap1[g_iX_LBM_MapSize][g_iX_LBM_MapSize][g_iTotalNumOfLS+1][60];
	float DiffMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][60];
	float AdpThr[g_iX_LBM_MapSize][g_iY_LBM_MapSize][60];
	float NewPM[g_iTotalNumOfLS+1][5][5][60]; //2nd dimension - AlgCode, 3rd - type of PM
	float NewPM1[g_iTotalNumOfLS+1][5][60]; //2nd dimension - AlgCode,
};
#endif
```

```
/**
** StaticParameters.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail: oprc@bgumail.bgu.ac.il
**
**/

#ifndef __StaticParameters_h__
#define __StaticParameters_h__

#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"

// structure for the saphira sensors array
//extern struct sfprocess *sfpMainLoop;

// Camera and OpenCV Parameters
extern int videotype;
extern int grab_type;
extern int ImageMaxX,
           ImageMaxY,
           WindowX,
           WindowY;
/*
#define PIXEL_TYPE PBITS_RGB24
#define PXC_NAME "pxc_95.dll"
#define FRAME_NAME "frame_32.dll"
#define PXC_NT  "pxc_nt.dll"
*/
static int videotype;
static int grab_type;
static int ImageMaxX,
                        ImageMaxY,
                        WindowX,
                        WindowY;
#endif
```

```
/**
** collector.h
**
** Copyright 2007 by Keren Kapach
**
** E-mail: kapach@bgu.ac.il
**
**/
#include "Aria.h"
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "UltraSonic_Class.h"
#include "Sick_Class.h"
//#include "InitiationFile.h"
#include "PXC_Camera_Dll_Load.h"
#include "LogicalSensor.h"
#include "image.h"
#include "ipl.h"
#include "cv.h"
#include <windows.h>
#include <cvlgrfmts.h>

#define move 1
#define stop 0
#define SPEED 40
#define PATH_LENGTH 1500
#define ReadDataFrom_US_Sensor 2
#define ReadDataFrom_Sick_Sensor 3
#define ReadDataFromCamera1 4
#define ReadDataFromCamera2 5
#define ReadDataFromCamera3 6
#define ReadDataFromCamera4 7
#define ReadDataFromCameraF 8


PXC pxc;
FRAMELIB frame;

// Camera and OpenCV Parameters
//extern PXC pxc ;
long fgh;
FRAME __PX_FAR *frh;
extern int videotype;
extern int grab_type;
extern int ImageMaxX,
            ImageMaxY,
            WindowX,
            WindowY;

   BlackBoard g_BB;
   ArRobot robot(NULL, false);
   ArSick *sick;

   Vision_Class CAM;
   UltraSonic_Class US;
   Sick_Class mySick;
   ArSonyPTZ myCam(&robot);

void initDLL();
```

```
/*******************************************************************************
* Name: main     *
* Description: This function is the main of the sensor fusion system       *
*******************************************************************************/

int main(int argc, char **argv) {

  // just some stuff for returns
  std::string str;
  int ret,process_state=1,sum=0;

 initDLL();
  Init_Parameters();
  int tCam,tPic ; // the camera
  tCam =1100  ; tPic = 0 ;
  ArTime start;
  ArSerialConnection con;
  ArSerialConnection conL;
  Aria::init();  // mandatory init

  sick = new ArSick;
  if ((ret = conL.open("COM3")) != 0) {   // opens the connection, if it fails, exit
    str = conL.getOpenMessage(ret);
    printf("Open failed: %s\n", str.c_str());
    Aria::shutdown();
    return 1;   }

   sick->configure(false, true, false, ArSick::BAUD38400,
   ArSick::DEGREES180, ArSick::INCREMENT_ONE);
   sick->setDeviceConnection(&conL);

   sick->runAsync();
   ArUtil::sleep(100);
   sick->lockDevice();
   sick->asyncConnect();
   sick->unlockDevice();

    while (!sick->isConnected())
   ArUtil::sleep(100);
    printf("Connected\n");

  if ((ret = con.open()) != 0){  // opens the connection, if it fails, exit
    str = con.getOpenMessage(ret);
    printf("Open failed: %s\n", str.c_str());
    Aria::shutdown();
    return 1;   }
    robot.setDeviceConnection(&con);  // set the connection on the robot

  if (!robot.blockingConnect())  {   // connect, if we fail, exit
    printf("Could not connect to robot... exiting\n");
    Aria::shutdown();
    return 1;   }

  robot.comInt(ArCommands::SONAR, 1);  // turn on the sonar, enable the motors, turn off amigobot sounds
  robot.comInt(ArCommands::ENABLE,1);

  robot.runAsync(true); // run, if we lose connection to the robot, exit
  Init_Parameters();
  myCam.tilt(-40);
  ArUtil::sleep(500);
```

```
  myCam.pan(-50);
  ArUtil::sleep(800);

 while (process_state){
  switch (process_state){
  case move:
            printf("Starting to move...\n\n");
            robot.setVel2(30,30);
            g_BB.iCycle=0;
            process_state = ReadDataFrom_US_Sensor;
            continue;

  case ReadDataFrom_US_Sensor:
            printf("Reading data from US sensor\n\n");
            start.setToNow();
            US.US_ReadDataFromUS();
            US.US_SFA_LogicalOR();
            US.US_SFA_ProbabilisticApproach();
            US.US_GridMapCellConversion();
            if (g_BB.iCycle>=0)
                Call_LS_Func();
            else
                g_BB.iCycle++;
            if (robot.getX()>4000) { //end of the line
                printf("Path Ended\n\n");
                robot.setVel2(0,0); // Set velocity for each wheel side independently.
                robot.comInt(ArCommands::SONAR, 0);
                SaveGGM();
                robot.comInt(ArCommands::SONAR, 0);
                process_state=stop;
                break;}
            else{
                process_state = ReadDataFrom_Sick_Sensor;
                continue;}

  case ReadDataFrom_Sick_Sensor:
             printf("Reading data from Sick #%d\n\n",sum);
            mySick.ReadFromSick();
            mySick.Si_GridMapCellConversion();
            if (g_BB.iCycle>=0)
                Call_LS_Func();
            else
                g_BB.iCycle++;

            if (robot.getX()>4000) {//end of the line
                printf("Path Ended\n\n");
                robot.setVel2(0,0); // Set velocity for each wheel side independently.
                robot.comInt(ArCommands::SONAR, 0);
                SaveGGM();
                process_state=stop;}
            else{
                if (g_BB.iCycle%2==0)
                    process_state = ReadDataFromCamera1;
                else
                    process_state = ReadDataFromCamera4;
                continue;}

  case ReadDataFromCamera1:  //-50
            printf("Angle -50\n");
            myCam.pan(-50);
```

```
            ArUtil::sleep(tCam);
            CAM.iVision_CameraAngleCode=1;
            ImageProcessingAlgo1(); // Take photo, update location and convert to gray scale
            CAM.iVision_CameraAngleCode=2;
            ImageProcessingAlgo3(0);        // Input: First  algorithm
            ImageProcessingAlgo3(1);        // Input: second  algorithm
            ImageProcessingAlgo3(2);        // Input: Third  algorithm
            if (g_BB.iCycle%2==0)
                        process_state = ReadDataFromCamera2;
                  else
                        process_state = ReadDataFromCameraF;
                  continue;



 case ReadDataFromCamera2: //-17
            printf("Angle -17\n");
            myCam.pan(-17);
            ArUtil::sleep(tCam);
            CAM.iVision_CameraAngleCode=2;
            ImageProcessingAlgo1();  // Take photo, update location and convert to gray scale
            CAM.iVision_CameraAngleCode=3;

            ImageProcessingAlgo3(0);        // Input: First  algorithm
            ImageProcessingAlgo3(1);        // Input: second  algorithm
            ImageProcessingAlgo3(2);        // Input: Third  algorithm
            if (g_BB.iCycle%2==0)
                        process_state = ReadDataFromCamera3;
                  else
                        process_state = ReadDataFromCamera1;
                  continue;
case ReadDataFromCamera3: //17
            printf("Angle 17\n");
            myCam.pan(17);
            ArUtil::sleep(tCam);
            //printf("my angle is 17 deg\n");
            CAM.iVision_CameraAngleCode=3;
            ImageProcessingAlgo1(); // Take photo, update location and convert to gray scale
            CAM.iVision_CameraAngleCode=4;
            ImageProcessingAlgo3(0);        // Input: First  algorithm
            ImageProcessingAlgo3(1);        // Input: second  algorithm
            ImageProcessingAlgo3(2);        // Input: Third  algorithm
            if (g_BB.iCycle%2==0)
                        process_state = ReadDataFromCamera4;
                  else
                        process_state = ReadDataFromCamera2;
                  continue;
 case ReadDataFromCamera4: //50
            printf("Angle 50\n");
            myCam.pan(50);
            ArUtil::sleep(tCam);
            CAM.iVision_CameraAngleCode=4;
            ImageProcessingAlgo1();  // Take photo, update location and convert to gray scale
            CAM.iVision_CameraAngleCode=1;

            ImageProcessingAlgo3(0);        // Input: First  algorithm
            ImageProcessingAlgo3(1);        // Input: second  algorithm*/
            ImageProcessingAlgo3(2);        // Input: Third  algorithm
            if (g_BB.iCycle%2==0)
                        process_state = ReadDataFromCameraF;
```

```
                else
                        process_state = ReadDataFromCamera3;
                continue;

 case ReadDataFromCameraF:
        printf("FUSING CAMERA DATA\n");
        ImageProcessingAlgo4(0);        // Converting into pic number 1 position
        ImageProcessingAlgo4(1);        // Converting into pic number 1 position
        ImageProcessingAlgo4(2);        // Converting into pic number 1 position
        CAM.Vision_GridMapCellConversion();
        if (g_BB.iCycle>=0){
                Call_LS_Func();
                g_BB.iCycle++;}
        else
                g_BB.iCycle++;

        if (robot.getX()>4000) {//end of the line
                printf("Path Ended\n\n");
                robot.setVel2(0,0); // Set velocity for each wheel side independently.
                robot.comInt(ArCommands::SONAR, 0);
                SaveGGM();
                process_state=stop;
                continue;}
        else{
                printf (" in cycle %d the time is %f\n", g_BB.iCycle,(double)start.mSecSince() );
                printf("The robots velocity is: %f\n\n", robot.getVel());
                process_state = ReadDataFrom_US_Sensor;
                continue;}

}//while

        printf("Stopping!\n\n\n");
        robot.comInt(ArCommands::SONAR, 0);
        robot.comInt(ArCommands::ENABLE, 0);
        robot.unlock();// shutdown and go away
        Aria::shutdown();
        return 0;
    }//switch
 return 0;}
```

```
/*******************************************************************************
* Name: initDLL            *
* Description: This function initializes the camera's dll files     *
*******************************************************************************/

void initDLL()
{
	if (!imagenation_OpenLibrary(PXC_NAME,&pxc,sizeof(pxc)))
	{
		if (!imagenation_OpenLibrary(PXC_NT,&pxc,sizeof(pxc)))
		{
		    printf("no load dll");
		}
	}

	if (!imagenation_OpenLibrary(FRAME_NAME,&frame,sizeof(frame)))
	{
	      printf("no load dll");
	}

	fgh = pxc.AllocateFG(-1);
	//sleep(2500); // wait for CCIR auto detect
	videotype = pxc.VideoType(fgh);

	switch(videotype) {
case 0:   // no video
case 1:   // NTSC
		grab_type = 0;
		ImageMaxX = 640;
		ImageMaxY = 486;
		break;
case 2:   // CCIR
		grab_type = 0;
		ImageMaxX = 768;
		ImageMaxY = 576;
		break;
	}

	pxc.SetWidth(fgh,(short)ImageMaxX);
	pxc.SetHeight(fgh,(short)ImageMaxY);
	pxc.SetLeft(fgh,0);
	pxc.SetTop(fgh,0);
	pxc.SetXResolution(fgh,(short)ImageMaxX);
	pxc.SetYResolution(fgh,(short)ImageMaxY);


	frh = pxc.AllocateBuffer((short)ImageMaxX, (short)ImageMaxY, PIXEL_TYPE);
}
```

```
/*****************************************************************************
* Name: Init_Parameters             *
* Description: This function initializes the parameters and sets all array to zeros  *
*****************************************************************************/
void Init_Parameters()
{
	int i, j, m;
	printf("Initiating Parameters\n\n");

	// Camera parameters
	g_BB.iLBM_X_New=0;
	g_BB.iLBM_Y_New=0;
	g_BB.iLBM_Theta_New=0;

// Initial performance measures parameters

	// Initialization of the PMs for the different LSs
	for (i=1; i<=g_iTotalNumOfLS; i++)
	{
		if (i==1)
		{
			g_BB.faTrueFalse[i][1]=0.85; //TT Value
			g_BB.faTrueFalse[i][2]=0.9; //FF Value
			g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
			g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
		}
		if (i==2)
		{
			g_BB.faTrueFalse[i][1]=0.78; //TT Value
			g_BB.faTrueFalse[i][2]=0.91; //FF Value
			g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
			g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
		}

		if (i==3)
		{
			g_BB.faTrueFalse[i][1]=0.8; //TT Value
			g_BB.faTrueFalse[i][2]=0.7; //FF Value
			g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
			g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
		}

		if (i==4)
		{
			g_BB.faTrueFalse[i][1]=0.6; //TT Value
			g_BB.faTrueFalse[i][2]=0.9; //FF Value
			g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
			g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
		}

		if (i==5)
		{
			g_BB.faTrueFalse[i][1]=0.88; //TT Value
			g_BB.faTrueFalse[i][2]=0.91; //FF Value
			g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
			g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
		}
		if (i==6)
		{
			g_BB.faTrueFalse[i][1]=0.92; //TT Value
```

```
            g_BB.faTrueFalse[i][2]=0.95; //FF Value
            g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
            g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
        }
        if (i==7)
        {
            g_BB.faTrueFalse[i][1]=0.93; //TT Value
            g_BB.faTrueFalse[i][2]=0.95; //FF Value
            g_BB.faTrueFalse[i][3]=1-g_BB.faTrueFalse[i][2]; //TF=1-TT Value
            g_BB.faTrueFalse[i][4]=1-g_BB.faTrueFalse[i][1]; //FT=1-FF Value
        }
}


g_BB.iCycle=0; // Initiating the cycle counter

for (m=1; m<=g_iTotalNumOfLS; m++)
        g_BB.baLS_Flag [m]=1; // initiating the LS flag to 1.

//Initiating the UM PM for the AdpWA algorithm
for (i=1; i<=g_iTotalNumOfLS; i++)
{
        if (i==1)
            g_BB.NewPM[i][1][4][0]=0.3;
        if (i==2)
            g_BB.NewPM[i][1][4][0]=0.3;
        if (i==3)
            g_BB.NewPM[i][1][4][0]=1;
        if (i==4)
            g_BB.NewPM[i][1][4][0]=1;
        if (i==5)
            g_BB.NewPM[i][1][4][0]=0.1;
        if (i==6)
            g_BB.NewPM[i][1][4][0]=0.1;
        if (i==7)
            g_BB.NewPM[i][1][4][0]=0.1;
}//for i

//Initiating the UM PM for the SFA_EnNEW algorithm
for (i=1; i<=g_iTotalNumOfLS; i++)
{
        if (i==1)
            g_BB.NewPM[i][3][4][0]=0.3;
        if (i==2)
            g_BB.NewPM[i][3][4][0]=0.3;
        if (i==3)
            g_BB.NewPM[i][3][4][0]=1;
        if (i==4)
            g_BB.NewPM[i][3][4][0]=1;
        if (i==5)
            g_BB.NewPM[i][3][4][0]=0.1;
        if (i==6)
            g_BB.NewPM[i][3][4][0]=0.1;
        if (i==7)
            g_BB.NewPM[i][3][4][0]=0.1;
}//for i

//Initiating the PM for the SFA_NEW1 algorithm
for (i=1; i<=g_iTotalNumOfLS; i++)
{
```

```
			if (i==1)
				g_BB.NewPM1[i][2][0]=0.3;
			if (i==2)
				g_BB.NewPM1[i][2][0]=0.3;
			if (i==3)
				g_BB.NewPM1[i][2][0]=1;
			if (i==4)
				g_BB.NewPM1[i][2][0]=1;
			if (i==5)
				g_BB.NewPM1[i][2][0]=0.1;
			if (i==6)
				g_BB.NewPM1[i][2][0]=0.1;
			if (i==7)
				g_BB.NewPM1[i][2][0]=0.1;

		}//for i

//Initiating the PM for the SFA_EnNEW1 algorithm
		for (i=1; i<=g_iTotalNumOfLS; i++)
		{
			if (i==1)
				g_BB.NewPM1[i][4][0]=0.3;
			if (i==2)
				g_BB.NewPM1[i][4][0]=0.3;
			if (i==3)
				g_BB.NewPM1[i][4][0]=1;
			if (i==4)
				g_BB.NewPM1[i][4][0]=1;
			if (i==5)
				g_BB.NewPM1[i][4][0]=0.1;
			if (i==6)
				g_BB.NewPM1[i][4][0]=0.1;
			if (i==7)
				g_BB.NewPM1[i][4][0]=0.1;
		}//for i
}
```

```
/**
** LogicalSensor.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail: oprc@bgumail.bgu.ac.il
**/
#ifndef __LogicalSensor_h__
#define __LogicalSensor_h__
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"
#include "FuzzyLogic_Algorithm.h"

extern PXC pxc;
extern FRAMELIB frame;
extern long fgh;
extern FRAME __PX_FAR *frh;
extern int videotype;
extern int grab_type;
extern int ImageMaxX,
        ImageMaxY,
        WindowX,
        WindowY;
extern BlackBoard g_BB;

void SFA_AND();
void SFA_OR();
void SFA_MOST();
void SFA_REGULAR_MOST();
void SFA_REGULAR_AFL();
void SFA_AdpWA1(); //New algorithm
void SFA_AdpWA2();
void SFA_AdpWA3(); //EnNEW - With Enhance, Binary PM, level 3
void SFA_AdpWA4(); //EnNEW1 - With Enhance, New PM, level 4
void CopyLBM2GGM(int);
void CreateLS_PPGM(int);
void SaveGGM();
void Call_LS_Func();
void Calculating_FL_TruthTable();
void FuzzyLogicAlgorithm(int);
void CalculatingTrueAndFalseValues(int);
void SFA_Calc_PM(int); // Calculates PM
void LGM_Transformation();

class LogicalSensor
{
private:

public:
        LogicalSensor();
        ~LogicalSensor();
};
#endif
```

```
/**
** LogicalSensor.cpp
**
** Copyright 2001 by Ofir Cohen
**
** E-mail: oprc@bgumail.bgu.ac.il
**
**/
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "LogicalSensor.h"
#include "InitiationFile.h"
#include "UltraSonic_Class.h"
#include "FuzzyLogic_Algorithm.h"
#include "STDIO.H"
#include <math.h>


extern PXC pxc;
extern FRAMELIB frame;

static long fgh;
static FRAME __PX_FAR *frh;

extern int videotype;
extern int grab_type;
extern int ImageMaxX,
ImageMaxY,
WindowX,
WindowY;

extern UltraSonic_Class US;

/*******************************************************************************
* Name: LGM_Transformation      *
* Description: This function transforms the logical sensors’ maps that were not update      *
*******************************************************************************/
void LGM_Transformation()
{ // 0
	int i, j, k;
	double DeltaX;
	double DeltaY;
	int i_New, j_New;

	g_BB.faRobotPos[g_BB.iCycle][0]=g_BB.iLBM_X_New;
	g_BB.faRobotPos[g_BB.iCycle][1]=g_BB.iLBM_Y_New;

	DeltaX=g_BB.iLBM_X_New-g_BB.iLBM_X_Old;
	DeltaY=g_BB.iLBM_Y_New-g_BB.iLBM_X_Old;

	for (k=1; k<=g_iTotalNumOfLS; k++)
	{ // 1
		if (g_BB.bLGM_NewDataFlag[k]) // If the LS has new data - FLAG =1
		{ // 2
			g_BB.bLGM_NewDataFlag[k]=0; // Set NewDataFlag to 0
			//sfSMessage("DeltaX= %d",DeltaX);
			for (i=0; i<g_iX_LBM_MapSize ; i++)
			{ // 3
				for(j=0; j<g_iY_LBM_MapSize ; j++)
```

```
					{ // 4
						// Save the New data in an array for the Off-Line simulation
						g_BB.iaLogicalSensorMap[i][j][k-
1][g_BB.iCycle]=g_BB.iaLBM[i][j][k];

					} // 4
				} // 3
			} // 2
			else
			{ // 5
				for (i=0; i<g_iX_LBM_MapSize ; i++)
				{ // 6
					for(j=0; j<g_iY_LBM_MapSize ; j++)
					{ // 7
					i_New=(int)(i-(int)(DeltaX/(double)g_LBMCellSize));
					j_New=j;
				if ((i_New>=0)&&(i_New<g_iX_LBM_MapSize)
&&(j_New>=0)&&(j_New<g_iY_LBM_MapSize))
						{ // 8
							g_BB.iaLBM[i_New][j_New][k]=g_BB.iaLBM[i][j][k];

						g_BB.iaLogicalSensorMap[i_New][j_New][k-
1][g_BB.iCycle]=g_BB.iaLBM[i][j][k];
							if(i>i_New) // For moving robot case
								g_BB.iaLBM[i][j][k]=0;
						} // 8

					}// 7
				}// 6
			}// 5
		}// 1
//		sfSMessage("Cycle %d",g_BB.iCycle); // print out the system's cycle
} // 0

/*********************************************************************************
* Name: SFA_AND		*
* Description: This function fuses the data between all LBMs using the AND method	*
*********************************************************************************/

void SFA_AND()
{ // 0
	int i,j,k,counter;

	for (i=0;i<g_iX_LBM_MapSize;i++)
	{ // 1
		for (j=0;j<g_iY_LBM_MapSize;j++)
		{ // 2
			counter=0;
			for (k=1;k<=g_iTotalNumOfLS;k++)
			{ // 3
				if(g_BB.iaLBM[i][j][k])
					counter++;
			} // 3
			if(counter==g_iTotalNumOfLS)
				g_BB.iaLBM[i][j][0]=1;
		} // 2
	} // 1
	// Calculate the PM for each LS by comparing the results to the fused AND map
	SFA_Calc_PM(1);
	CopyLBM2GGM(1);
```

```
} // 0

/*********************************************************************************
* Name: SFA_OR            *
 * Description: This function fuses the data between all LBM using the OR method          *
*********************************************************************************/
void SFA_OR()
{
	int i,j,k;
	for (i=0;i<g_iX_LBM_MapSize;i++)
	{
		for (j=0;j<g_iY_LBM_MapSize;j++)
		{
			for (k=1;k<=g_iTotalNumOfLS;k++)
			{
				if(g_BB.iaLBM[i][j][k])
				{
					g_BB.iaLBM[i][j][0]=1;
					k=g_iTotalNumOfLS;
				}
			}
		}
	}
	// Calculate the PM for each LS by comparing the results to the fused OR map
	SFA_Calc_PM(0);
	CopyLBM2GGM(0);
}

/*********************************************************************************
* Name: SFA_AdpWA1    *
* Description: This function fuses the data between all LBM using AdpWA1 algorithm, which means without * *
Enhancment, binary PM.   *
*********************************************************************************/

void SFA_AdpWA1()
{
	int AvgMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize]={0},i,j,k,sum,count;
	float AdpThr;

	//Enhancing the maps of the LS

	int EnLSMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfLS+1]={0};
	int mean[g_iTotalNumOfLS]={0};
	char string[30];
	char cLocNum[10];
	int m;

	for (k=1; k<=g_iTotalNumOfLS; k++)
	{
		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
			{
				EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k];
			}
		}
	}


	//Calculating the Avg. of number of samples
```

```
		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
			{
				sum=0;
				count=0;
				for (k=1; k<=g_iTotalNumOfLS; k++)
					if (EnLSMap[i][j][k])
					{
						count++;
						sum=sum+EnLSMap[i][j][k];
					}
					if (count)
						AvgMap[i][j]=sum/count;
				g_BB.AvgMap[i][j][g_BB.iCycle]=AvgMap[i][j];
			}//for j
		}//for i


	//Calculating the fused map according to the adp. thr.
	for (i=0; i<g_iX_LBM_MapSize; i++)
	{
		for (j=0; j<g_iY_LBM_MapSize; j++)
		{
			AdpThr=0;
			sum=0;

			for (k=1; k<=g_iTotalNumOfLS; k++)
			{
					sum=sum+g_BB.NewPM[k][1][4][g_BB.iCycle];
					if (g_BB.iCycle)
						AdpThr=AdpThr+g_BB.NewPM[k][1][4][g_BB.iCycle-
1]*EnLSMap[i][j][k];
					else

	AdpThr=AdpThr+g_BB.NewPM[k][1][4][0]*EnLSMap[i][j][k];

			}
			if (sum)
				AdpThr=AdpThr/sum;

			//Saving AdpThr and Diff for off line testing
			g_BB.AdpThr[i][j][g_BB.iCycle]=AdpThr;


			if (AdpThr>=AvgMap[i][j])
				g_BB.iaLBM[i][j][0]=AvgMap[i][j];
			else
				g_BB.iaLBM[i][j][0]=0;
		}//for j
	}//for i

	SFA_Calc_PM(1);
	CopyLBM2GGM(1);
}
```

```
/***********************************************************************************
* Name: SFA_AdpWA2
* Description: This function fuses the data between all LBM using AdpWA1 algorithm, which means without * *
Enhancment, New PM.
***********************************************************************************/

void SFA_AdpWA2()
{
	int AvgMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize]={0},i,j,k,sum,count;
	float AdpThr;
	static iCycle;
	char string[60];
	int SumSquaredError, SquaredFusedSum;

	int EnLSMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfLS+1]={0};
	int mean[g_iTotalNumOfLS]={0};
	char cLocNum[10],Counter[10];
	int m;


	//Copying LS maps to EnLSMap array
	for (k=1; k<=g_iTotalNumOfLS; k++)
	{
		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
				EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k];
		}
	}

	//Calculating the Avg. of number of samples
		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
			{
				sum=0;
				count=0;
				for (k=1; k<=g_iTotalNumOfLS; k++)
					if (EnLSMap[i][j][k])
					{
						count++;
						sum=sum+EnLSMap[i][j][k];
					}
					if (count)
						AvgMap[i][j]=sum/count;
				g_BB.AvgMap[i][j][g_BB.iCycle]=AvgMap[i][j];

			}//for j
		}//for i
	//Calculating the fused map according to the adp. thr.
	for (i=0; i<g_iX_LBM_MapSize; i++)
	{
		for (j=0; j<g_iY_LBM_MapSize; j++)
		{
			AdpThr=0;
			sum=0;

			for (k=1; k<=g_iTotalNumOfLS; k++)
			{
```

```
					sum=sum+g_BB.NewPM1[k][2][g_BB.iCycle];
					if (g_BB.iCycle)
						AdpThr=AdpThr+g_BB.NewPM1[k][2][g_BB.iCycle-
1]*EnLSMap[i][j][k];
					else
						AdpThr=AdpThr+g_BB.NewPM1[k][2][0]*EnLSMap[i][j][k];
			}
			if (sum)
				AdpThr=AdpThr/sum;

			//Saving AdpThr and Diff for off line testing
			g_BB.AdpThr[i][j][g_BB.iCycle] =AdpThr;

			if (AdpThr>=AvgMap[i][j])
				g_BB.iaLBM[i][j][0]=AvgMap[i][j];
			else
				g_BB.iaLBM[i][j][0]=0;
		}//for j

	}//for i

	int LSSquaredSum,FusedSquaredSum;
	float PM_Old=0, PM_New=0;

	//Calculating New PM
	int Temp;
	float ErrorCellRatio,ErrorSquaredSum;
	for (k=1; k<=g_iTotalNumOfLS; k++)
	{
		ErrorSquaredSum=0;
		FusedSquaredSum=0;
		LSSquaredSum=0;


		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
			{

				LSSquaredSum=LSSquaredSum+pow(EnLSMap[i][j][k],2);
				if (g_BB.iaLBM[i][j][0] && EnLSMap[i][j][k])
				{
					ErrorCellRatio=(float)(EnLSMap[i][j][k]-
g_BB.iaLBM[i][j][0])/(float)g_BB.iaLBM[i][j][0];
					ErrorSquaredSum=ErrorSquaredSum+pow(ErrorCellRatio,2);
					FusedSquaredSum=FusedSquaredSum+pow(g_BB.iaLBM[i][j][0],2);
				}
			}
		}
		if (g_BB.iCycle)
			PM_Old=g_BB.NewPM1[k][2][g_BB.iCycle-1];
		else
			PM_Old=g_BB.NewPM1[k][2][g_BB.iCycle];

		if (ErrorSquaredSum)
			PM_New=ErrorSquaredSum;
		else
			PM_New=PM_Old;

		g_BB.NewPM1[k][2][g_BB.iCycle]=0.5*(PM_New+PM_Old);
```

```
	}//for k
	//finding the maximum PM for normalization
		float max=0;
		for (k=1; k<=g_iTotalNumOfLS; k++)
			if (g_BB.NewPM1[k][2][g_BB.iCycle]>max)
				max=g_BB.NewPM1[k][2][g_BB.iCycle];

		for (k=1; k<=g_iTotalNumOfLS; k++)
		{
			g_BB.NewPM1[k][2][g_BB.iCycle]=g_BB.NewPM1[k][2][g_BB.iCycle]/max;
		}

	CopyLBM2GGM(2);
}

/******************************************************************************
* Name: SFA_AdpWA3    *
* Description: This function fuses the data between all LBM using AdpWA1 algorithm, which means withEnhancment,
Binary PM*
******************************************************************************/

void SFA_AdpWA3()
{
	int AvgMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize]={0},i,j,k,sum,count;
	float AdpThr;
	static iCycle;
	char string[60];
	int SumSquaredError, SquaredFusedSum;

	//Enhancing the maps of the LS
	int EnLSMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfLS+1]={0};
	int mean[g_iTotalNumOfLS]={0};
	char cLocNum[10];
	int m;

//Enahncing the LS Maps
	for (k=1; k<=g_iTotalNumOfLS; k++)
	{
		for (i=1; i<g_iX_LBM_MapSize-1; i++)
		{
			for (j=1; j<g_iY_LBM_MapSize-1; j++)
			{
				count=0;
				sum=0;
				if (g_BB.iaLBM[i-1][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j-1][k];}
				if (g_BB.iaLBM[i-1][j][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j][k];}
				if (g_BB.iaLBM[i-1][j+1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j+1][k];}
				if (g_BB.iaLBM[i][j+1][k]){ count++; sum=sum+g_BB.iaLBM[i][j+1][k];}
				if (g_BB.iaLBM[i+1][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[i+1][j+1][k];}
				if (g_BB.iaLBM[i+1][j][k]){ count++; sum=sum+g_BB.iaLBM[i+1][j][k];}
				if (g_BB.iaLBM[i+1][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i+1][j-1][k];}
				if (g_BB.iaLBM[i][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i][j-1][k];}

				if (count>4 && g_BB.iaLBM[i][j][k])
					EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
				else
```

```
                        EnLSMap[i][j][k]=0;
            }//for i
        }//for j

        //First row
        for (j=1; j<g_iY_LBM_MapSize-1; j++)
        {
            sum=0;
            count=0;
            if (g_BB.iaLBM[0][j+1][k]){ count++; sum=sum+g_BB.iaLBM[0][j+1][k];}
            if (g_BB.iaLBM[1][j+1][k]){count++; sum=sum+g_BB.iaLBM[1][j+1][k];}
            if (g_BB.iaLBM[1][j][k]){ count++; sum=sum+g_BB.iaLBM[1][j][k];}
            if (g_BB.iaLBM[1][j-1][k]){count++; sum=sum+g_BB.iaLBM[1][j-1][k];}
            if (g_BB.iaLBM[0][j-1][k]){ count++; sum=sum+g_BB.iaLBM[0][j-1][k];}

            if (count>3 && g_BB.iaLBM[0][j][k])
                        EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
                else
                        EnLSMap[i][j][k]=0;
        }//for j

        //Last row
        for (j=1; j<g_iY_LBM_MapSize-1; j++)
        {
            sum=0;
            count=0;
            if (g_BB.iaLBM[g_iX_LBM_MapSize-1][j-1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][j-1][k];}
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][j-1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][j-1][k];}
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][j][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][j][k];}
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][j+1][k];}
            if (g_BB.iaLBM[g_iX_LBM_MapSize-1][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][j+1][k];}

            if (count>3 && g_BB.iaLBM[g_iX_LBM_MapSize-1][j][k])
                        EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
                else
                        EnLSMap[i][j][k]=0;
        }//for j
        //First Col.
        for (i=1; i<g_iX_LBM_MapSize-1; i++)
        {
            sum=0;
            count=0;
            if (g_BB.iaLBM[i-1][0][k]){ count++; sum=sum+g_BB.iaLBM[i-1][0][k];}
            if (g_BB.iaLBM[i-1][1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][1][k];}
            if (g_BB.iaLBM[i][1][k]){ count++; sum=sum+g_BB.iaLBM[i][1][k];}
            if (g_BB.iaLBM[i+1][1][k]){ count++; sum=sum+g_BB.iaLBM[i+1][1][k];}
            if (g_BB.iaLBM[i+1][0][k]){ count++; sum=sum+g_BB.iaLBM[i+1][0][k];}

            if (count>3 && g_BB.iaLBM[i][0][k])
                        EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
                else
                        EnLSMap[i][j][k]=0;
        }//for i

        //Last Col.
```

```
		for (i=1; i<g_iX_LBM_MapSize-1; i++)
		{
			sum=0;
			count=0;
			if (g_BB.iaLBM[i-1][g_iY_LBM_MapSize-1][k]){ count++; sum=sum+g_BB.iaLBM[i-
1][g_iY_LBM_MapSize-1][k];}
			if (g_BB.iaLBM[i-1][g_iY_LBM_MapSize-2][k]){ count++; sum=sum+g_BB.iaLBM[i-
1][g_iY_LBM_MapSize-2][k];}
			if (g_BB.iaLBM[i][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[i][g_iY_LBM_MapSize-2][k];}
			if (g_BB.iaLBM[i+1][g_iY_LBM_MapSize-2][k]){count++;
sum=sum+g_BB.iaLBM[i+1][g_iY_LBM_MapSize-2][k];}
			if (g_BB.iaLBM[i+1][g_iY_LBM_MapSize-1][k]){ count++;
sum=sum+g_BB.iaLBM[i+1][g_iY_LBM_MapSize-1][k];}

			if (count>3 && g_BB.iaLBM[i][g_iY_LBM_MapSize-1][k])
					EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
				else
					EnLSMap[i][j][k]=0;
		}//for i

	//Corners - Top left
		sum=0;
		count=0;
		if (g_BB.iaLBM[0][1][k]){ count++; sum=sum+g_BB.iaLBM[0][1][k]; }
		if (g_BB.iaLBM[1][1][k]){ count++; sum=sum+g_BB.iaLBM[1][1][k]; }
		if (g_BB.iaLBM[1][0][k]){ count++; sum=sum+g_BB.iaLBM[1][0][k]; }
		if (count>2 && g_BB.iaLBM[0][0][k])
			EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
		else
			EnLSMap[i][j][k]=0;

		//Bottom left
		sum=0;
		count=0;
		if (g_BB.iaLBM[g_iX_LBM_MapSize-2][0][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][0][k]; }
		if (g_BB.iaLBM[g_iX_LBM_MapSize-2][1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][1][k];}
		if (g_BB.iaLBM[g_iX_LBM_MapSize-1][1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][1][k]; }
		if (count>2 && g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-1][k])
			EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
		else
			EnLSMap[i][j][k]=0;

		//Top right
		sum=0;
		count=0;
		if (g_BB.iaLBM[0][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[0][g_iY_LBM_MapSize-2][k];}
		if (g_BB.iaLBM[1][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[1][g_iY_LBM_MapSize-2][k];}
		if (g_BB.iaLBM[1][g_iY_LBM_MapSize-1][k]){ count++;
sum=sum+g_BB.iaLBM[1][g_iY_LBM_MapSize-1][k]; }
		if (count>2 && g_BB.iaLBM[0][g_iY_LBM_MapSize-1])
			EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
		else
			EnLSMap[i][j][k]=0;
```

```
			//Bottom right
			sum=0;
			count=0;
			if (g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-1][k]; }
			if (g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-2][k];  }
			if (g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-2][k]; }
			if (count>2 && g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-1][k])
				EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
			else
				EnLSMap[i][j][k]=0;
		}//for k

		//Calculating the Avg. of number of samples
			for (i=0; i<g_iX_LBM_MapSize; i++)
			{
				for (j=0; j<g_iY_LBM_MapSize; j++)
				{
					sum=0;
					count=0;
					for (k=1; k<=g_iTotalNumOfLS; k++)
						if (EnLSMap[i][j][k])
						{
							count++;
							sum=sum+EnLSMap[i][j][k];
						}
						if (count)
							AvgMap[i][j]=sum/count;
					g_BB.AvgMap[i][j][g_BB.iCycle]=AvgMap[i][j];
				}//for j

			}//for i

		//Calculating the fused map according to the adp. thr.
		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
			{
				AdpThr=0;
				sum=0;

				for (k=1; k<=g_iTotalNumOfLS; k++)
				{
						sum=sum+g_BB.NewPM[k][3][4][g_BB.iCycle];
						if (g_BB.iCycle)
							AdpThr=AdpThr+g_BB.NewPM[k][3][4][g_BB.iCycle-
1]*EnLSMap[i][j][k];
						else

		AdpThr=AdpThr+g_BB.NewPM[k][3][4][0]*EnLSMap[i][j][k];
				}
				if (sum)
					AdpThr=AdpThr/sum;

				//Saving AdpThr and Diff for off line testing
				g_BB.AdpThr[i][j][g_BB.iCycle] =AdpThr;

				if (AdpThr>=AvgMap[i][j])
```

```
				g_BB.iaLBM[i][j][0]=AvgMap[i][j];
			else
				g_BB.iaLBM[i][j][0]=0;

		}//for j

	}//for i

	SFA_Calc_PM(3);
	CopyLBM2GGM(3);
}


/***************************************************************************
* Name: SFA_AdpWA4    *
* Description: This function fuses the data between all LBM using AdpWA1 algorithm, which means without Enhancment, new PM.            *
***************************************************************************/

void SFA_AdpWA4()
{
	int AvgMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize]={0},i,j,k,sum,count;
	float AdpThr;
	static iCycle;
	char string[60];
	int SumSquaredError, SquaredFusedSum;

	//Enhancing the maps of the LS

	int EnLSMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfLS+1]={0};
	int mean[g_iTotalNumOfLS]={0};
	char cLocNum[10];
	int m;

//Enahncing the LS Maps
	for (k=1; k<=g_iTotalNumOfLS; k++)
	{
		//all together
		for (i=0; i<g_iX_LBM_MapSize-1; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize-1; j++)
			{
				count=0;
				sum=0;
				if (i>=1 && j>=1) {
					if (g_BB.iaLBM[i-1][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j-
1][k];}
				}
				if (i>=1) {
					if (g_BB.iaLBM[i-1][j][k]){ count++; sum=sum+g_BB.iaLBM[i-
1][j][k];}

					if (g_BB.iaLBM[i-1][j+1][k]){ count++; sum=sum+g_BB.iaLBM[i-
1][j+1][k];}
				}
				if (g_BB.iaLBM[i][j+1][k]){ count++; sum=sum+g_BB.iaLBM[i][j+1][k];}
				if (g_BB.iaLBM[i+1][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[i+1][j+1][k];}
				if (g_BB.iaLBM[i+1][j][k]){ count++; sum=sum+g_BB.iaLBM[i+1][j][k];}
				if (j>=1) {
```

```
					if (g_BB.iaLBM[i+1][j-1][k]){ count++;
sum=sum+g_BB.iaLBM[i+1][j-1][k];}
					if (g_BB.iaLBM[i][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i][j-
1][k];}
				}

				if (count>4 && g_BB.iaLBM[i][j][k])
					EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
				else
					EnLSMap[i][j][k]=0;
			}//for i
		}//for j

		for (i=1; i<g_iX_LBM_MapSize-1; i++)
		{
			for (j=1; j<g_iY_LBM_MapSize-1; j++)
			{
				count=0;
				sum=0;
				if (g_BB.iaLBM[i-1][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j-1][k];}
				if (g_BB.iaLBM[i-1][j][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j][k];}
				if (g_BB.iaLBM[i-1][j+1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][j+1][k];}
				if (g_BB.iaLBM[i][j+1][k]){ count++; sum=sum+g_BB.iaLBM[i][j+1][k];}
				if (g_BB.iaLBM[i+1][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[i+1][j+1][k];}
				if (g_BB.iaLBM[i+1][j][k]){ count++; sum=sum+g_BB.iaLBM[i+1][j][k];}
				if (g_BB.iaLBM[i+1][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i+1][j-1][k];}
				if (g_BB.iaLBM[i][j-1][k]){ count++; sum=sum+g_BB.iaLBM[i][j-1][k];}

				if (count>4 && g_BB.iaLBM[i][j][k])
					EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
				else
					EnLSMap[i][j][k]=0;
			}//for i
		}//for j

		//First row
		for (j=1; j<g_iY_LBM_MapSize-1; j++)
		{
			sum=0;
			count=0;
			if (g_BB.iaLBM[0][j+1][k]){ count++; sum=sum+g_BB.iaLBM[0][j+1][k];}
			if (g_BB.iaLBM[1][j+1][k]){count++; sum=sum+g_BB.iaLBM[1][j+1][k];}
			if (g_BB.iaLBM[1][j][k]){ count++; sum=sum+g_BB.iaLBM[1][j][k];}
			if (g_BB.iaLBM[1][j-1][k]){count++; sum=sum+g_BB.iaLBM[1][j-1][k];}
			if (g_BB.iaLBM[0][j-1][k]){ count++; sum=sum+g_BB.iaLBM[0][j-1][k];}

			if (count>3 && g_BB.iaLBM[0][j][k])
					EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
				else
					EnLSMap[i][j][k]=0;
		}//for j

		//Last row
		for (j=1; j<g_iY_LBM_MapSize-1; j++)
		{
			sum=0;
			count=0;
			if (g_BB.iaLBM[g_iX_LBM_MapSize-1][j-1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][j-1][k];}
```

```
				if (g_BB.iaLBM[g_iX_LBM_MapSize-2][j-1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][j-1][k];}
				if (g_BB.iaLBM[g_iX_LBM_MapSize-2][j][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][j][k];}
				if (g_BB.iaLBM[g_iX_LBM_MapSize-2][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][j+1][k];}
				if (g_BB.iaLBM[g_iX_LBM_MapSize-1][j+1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][j+1][k];}

				if (count>3 && g_BB.iaLBM[g_iX_LBM_MapSize-1][j][k])
						EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
					else
						EnLSMap[i][j][k]=0;
			}//for j
			//First Col.
			for (i=1; i<g_iX_LBM_MapSize-1; i++)
			{
				sum=0;
				count=0;
				if (g_BB.iaLBM[i-1][0][k]){ count++; sum=sum+g_BB.iaLBM[i-1][0][k];}
				if (g_BB.iaLBM[i-1][1][k]){ count++; sum=sum+g_BB.iaLBM[i-1][1][k];}
				if (g_BB.iaLBM[i][1][k]){ count++; sum=sum+g_BB.iaLBM[i][1][k];}
				if (g_BB.iaLBM[i+1][1][k]){ count++; sum=sum+g_BB.iaLBM[i+1][1][k];}
				if (g_BB.iaLBM[i+1][0][k]){ count++; sum=sum+g_BB.iaLBM[i+1][0][k];}

				if (count>3 && g_BB.iaLBM[i][0][k])
						EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
					else
						EnLSMap[i][j][k]=0;
			}//for i

			//Last Col.
			for (i=1; i<g_iX_LBM_MapSize-1; i++)
			{
				sum=0;
				count=0;
				if (g_BB.iaLBM[i-1][g_iY_LBM_MapSize-1][k]){ count++; sum=sum+g_BB.iaLBM[i-
1][g_iY_LBM_MapSize-1][k];}
				if (g_BB.iaLBM[i-1][g_iY_LBM_MapSize-2][k]){ count++; sum=sum+g_BB.iaLBM[i-
1][g_iY_LBM_MapSize-2][k];}
				if (g_BB.iaLBM[i][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[i][g_iY_LBM_MapSize-2][k];}
				if (g_BB.iaLBM[i+1][g_iY_LBM_MapSize-2][k]){count++;
sum=sum+g_BB.iaLBM[i+1][g_iY_LBM_MapSize-2][k];}
				if (g_BB.iaLBM[i+1][g_iY_LBM_MapSize-1][k]){ count++;
sum=sum+g_BB.iaLBM[i+1][g_iY_LBM_MapSize-1][k];}

				if (count>3 && g_BB.iaLBM[i][g_iY_LBM_MapSize-1][k])
						EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
					else
						EnLSMap[i][j][k]=0;
			}//for i

		//Corners - Top left
			sum=0;
			count=0;
			if (g_BB.iaLBM[0][1][k]){ count++; sum=sum+g_BB.iaLBM[0][1][k]; }
			if (g_BB.iaLBM[1][1][k]){ count++; sum=sum+g_BB.iaLBM[1][1][k]; }
			if (g_BB.iaLBM[1][0][k]){ count++; sum=sum+g_BB.iaLBM[1][0][k]; }
			if (count>2 && g_BB.iaLBM[0][0][k])
```

```
                EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
            else
                EnLSMap[i][j][k]=0;

            //Bottom left
            sum=0;
            count=0;
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][0][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][0][k]; }
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][1][k];}
            if (g_BB.iaLBM[g_iX_LBM_MapSize-1][1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][1][k]; }
            if (count>2 && g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-1][k])
                EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
            else
                EnLSMap[i][j][k]=0;

            //Top right
            sum=0;
            count=0;
            if (g_BB.iaLBM[0][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[0][g_iY_LBM_MapSize-2][k];}
            if (g_BB.iaLBM[1][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[1][g_iY_LBM_MapSize-2][k];}
            if (g_BB.iaLBM[1][g_iY_LBM_MapSize-1][k]){ count++;
sum=sum+g_BB.iaLBM[1][g_iY_LBM_MapSize-1][k]; }
            if (count>2 && g_BB.iaLBM[0][g_iY_LBM_MapSize-1])
                EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
            else
                EnLSMap[i][j][k]=0;

            //Bottom right
            sum=0;
            count=0;
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-1][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-1][k]; }
            if (g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-2][g_iY_LBM_MapSize-2][k];  }
            if (g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-2][k]){ count++;
sum=sum+g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-2][k]; }
            if (count>2 && g_BB.iaLBM[g_iX_LBM_MapSize-1][g_iY_LBM_MapSize-1][k])
                EnLSMap[i][j][k]=g_BB.iaLBM[i][j][k]+sum/count;
            else
                EnLSMap[i][j][k]=0;
        }//for k

        //Calculating the Avg. of number of samples
            for (i=0; i<g_iX_LBM_MapSize; i++)
            {
                for (j=0; j<g_iY_LBM_MapSize; j++)
                {
                    sum=0;
                    count=0;
                    for (k=1; k<=g_iTotalNumOfLS; k++)
                        if (EnLSMap[i][j][k])
                        {
                            count++;
                            sum=sum+EnLSMap[i][j][k];
                        }
```

```
					if (count)
						AvgMap[i][j]=sum/count;
				g_BB.AvgMap[i][j][g_BB.iCycle]=AvgMap[i][j];
			}//for j

		}//for i

	//Calculating the fused map according to the adp. thr.
	for (i=0; i<g_iX_LBM_MapSize; i++)
	{
		for (j=0; j<g_iY_LBM_MapSize; j++)
		{
			AdpThr=0;
			sum=0;

			for (k=1; k<=g_iTotalNumOfLS; k++)
			{
					sum=sum+g_BB.NewPM1[k][4][g_BB.iCycle];
					if (g_BB.iCycle)
						AdpThr=AdpThr+g_BB.NewPM1[k][4][g_BB.iCycle-
1]*EnLSMap[i][j][k];
					else
						AdpThr=AdpThr+g_BB.NewPM1[k][4][0]*EnLSMap[i][j][k];
			}
			if (sum)
				AdpThr=AdpThr/sum;

			//Saving AdpThr and Diff for off line testing
			g_BB.AdpThr[i][j][g_BB.iCycle] =AdpThr;

			if (AdpThr>=AvgMap[i][j])
				g_BB.iaLBM[i][j][0]=AvgMap[i][j];
			else
				g_BB.iaLBM[i][j][0]=0;

		}//for j

	}//for i

	int LSSquaredSum,FusedSquaredSum;
	float PM_Old=0, PM_New=0;

	//Calculating New PM
	int Temp;
	float ErrorCellRatio,ErrorSquaredSum;
	for (k=1; k<=g_iTotalNumOfLS; k++)
	{
		ErrorSquaredSum=0;
		FusedSquaredSum=0;
		LSSquaredSum=0;


		for (i=0; i<g_iX_LBM_MapSize; i++)
		{
			for (j=0; j<g_iY_LBM_MapSize; j++)
			{

				LSSquaredSum=LSSquaredSum+pow(EnLSMap[i][j][k],2);
				if (g_BB.iaLBM[i][j][0] && EnLSMap[i][j][k])
				{
```

```
                                        ErrorCellRatio=(float)(EnLSMap[i][j][k]-
g_BB.iaLBM[i][j][0])/(float)g_BB.iaLBM[i][j][0];
                                        ErrorSquaredSum=ErrorSquaredSum+pow(ErrorCellRatio,2);
                                        FusedSquaredSum=FusedSquaredSum+pow(g_BB.iaLBM[i][j][0],2);
                                }
                        }
                }

                if (g_BB.iCycle)
                        PM_Old=g_BB.NewPM1[k][2][g_BB.iCycle-1];
                else
                        PM_Old=g_BB.NewPM1[k][2][g_BB.iCycle];

                if (ErrorSquaredSum)
                        PM_New=ErrorSquaredSum;
                else
                        PM_New=PM_Old;


                g_BB.NewPM1[k][4][g_BB.iCycle]=0.5*(PM_New+PM_Old);


        }//for k

        //finding the maximum PM for normalization
                float max=0;
                for (k=1; k<=g_iTotalNumOfLS; k++)
                        if (g_BB.NewPM1[k][4][g_BB.iCycle]>max)
                                max=g_BB.NewPM1[k][4][g_BB.iCycle];

                for (k=1; k<=g_iTotalNumOfLS; k++)
                        g_BB.NewPM1[k][4][g_BB.iCycle]=g_BB.NewPM1[k][4][g_BB.iCycle]/max;

        CopyLBM2GGM(4);
}


/*******************************************************************************
* Name: SFA_REGULAR_MOST  *
* Description: This function fuses the data between all LBM using the regular MOST method        *
*******************************************************************************/
void SFA_REGULAR_MOST()
{
        int i,j,k, counter, MOST_US_Camera;

        // Definitions of the MOST, using the 'ceil' function
        MOST_US_Camera=(int)(ceil(0.5*(float)g_iTotalNumOfLS));
        //sfSMessage("MOST_US_Camera %d", MOST_US_Camera);
        for (i=0;i<g_iX_LBM_MapSize;i++)
        {
                for (j=0;j<g_iY_LBM_MapSize;j++)
                {
                        counter=0;
                        for (k=1;k<=g_iTotalNumOfLS;k++)
                        {
                                if(g_BB.iaLBM[i][j][k])
                                        counter++;
                                if(counter>=MOST_US_Camera)
                                {
                                        g_BB.iaLBM[i][j][0]=1;
```

```
                                        k=g_iTotalNumOfLS;
                                    }
                                }
                        }
            }
            // Calculate the PM for each LS by comparing the results to the fused MOST map
            SFA_Calc_PM(0);
            CopyLBM2GGM(0);
}

/*******************************************************************************
* Name: SFA_REGULAR_AFL     *
* Description: This function fuses the data between all LBM using the regular AFL method          *
*******************************************************************************/
void SFA_REGULAR_AFL()
{
            int i,j,k, m,Temp;
            int iCellValue;

            g_BB.iAFL_Flag=0; // 0 for regular AFL, 1 for MOST+AFL
            // Calculating the TRUE and FALSE value for each LS
            for (m=1; m<=g_iTotalNumOfLS; m++)
                        FuzzyLogicAlgorithm(m);

            // Calculating truth table for each LS
            Calculating_FL_TruthTable();
            // Based on the Truth table the fused map is built
            for ( i=0;i<g_iX_LBM_MapSize;i++)
            {
                        for ( j=0;j<g_iY_LBM_MapSize;j++)
                        {
                                    iCellValue=0;
                                    for ( k=0;k<g_iTotalNumOfLS;k++)
                                    {
                                                if (g_BB.iaLBM[i][j][k+1])
                                                            Temp=1;
                                                else
                                                            Temp=0;
                                                iCellValue= // The value for the cell at the Truth table [0/1]
                                                            iCellValue+(Temp*pow(2,k));
                                    }// for (int k=1;k<2+g_NumberOfModules;k++)
                                    g_BB.iaLBM[i][j][0]=g_BB.fTTValue[0][iCellValue];
                        }//for (int j=0;j<2*g_SensorYLength;j++)
            }// for (int i=0;i<g_SensorXLength;i++)*/

            for (m=1; m<=g_iTotalNumOfLS; m++)
                        CalculatingTrueAndFalseValues(m);
            // Calculate the PM for each LS by comparing the results to the fused OLSAS map
            SFA_Calc_PM(5);
            CopyLBM2GGM(5);
}


/*******************************************************************************
* Name: Call_LS_Func     *
* Description: This function is used to fuse the data using the all algorithms methods        *
*******************************************************************************/
void Call_LS_Func()
{ //1
            float fSpecialMeasure, fTT, fFF, fTF, fFT;
```

```
	int i,j,k;

	LGM_Transformation(); // map transformation for the maps that were not updated

	//Saving PPGM for the Logical sensor's maps
	CreateLS_PPGM(1); //US1
	CreateLS_PPGM(2); //US2
	CreateLS_PPGM(3); //LASER1
	CreateLS_PPGM(4); //LASER2
	CreateLS_PPGM(5); //CAM1
	CreateLS_PPGM(6); //CAM2
	CreateLS_PPGM(7); //CAM3
	//********************
// Fusing Data using the sensor fusion algorithms: OR/AND/MOST/ADS/AFL
	// It would help us to compare the results with what would happen if we used algorithm
	// instead of the other

	SFA_OR();          // OR writing to level 0
	SFA_AND();         // AND writing to level 1
	SFA_REGULAR_MOST(); // Regular MOST writing to level 0

	SFA_AdpWA1(); // NEW  -Without Enhance, Binary PM ,level 1
	SFA_AdpWA2(); //NEW1  - Without Enhance, New PM, level 2
	SFA_AdpWA3(); //EnNEW - With Enhance, Binary PM, level 3
	SFA_AdpWA4(); //EnNEW1 - With Enhance, New PM, level 4
	SFA_REGULAR_AFL(); // Level 5
} // 1
```

```
/*****************************************************************************
* Name: CopyLBM2GGM             *
* Description: This function copy the local binary maps (LBMs) to the global grid maps   *
*****************************************************************************/
void CopyLBM2GGM(int iAlgCode)
{
	int i, j;
	int iNew, jNew;
	for (i=0; i<g_iX_LBM_MapSize; i++)
	{
		for (j=0; j<g_iY_LBM_MapSize; j++)
		{
			iNew=i+(int)((double)g_BB.iPPGM_X/(double)g_LBMCellSize);
			jNew=j+(int)(((double)g_iY_PPGM_MapSize-(double)g_iY_LBM_MapSize)/2);
			if ((iNew>=0)&&(jNew< g_iY_PPGM_MapSize)&&
				(jNew>=0)&&(iNew< g_iX_PPGM_MapSize)&&(g_BB.iaLBM[i][j][0]))
					g_BB.iaPPGM[iNew][jNew][iAlgCode]=g_BB.iaLBM[i][j][0];
			if (iAlgCode==1)
				g_BB.iaPI[iNew][jNew]++;
		}
	}
	for (i=0; i<g_iX_LBM_MapSize; i++)// Save LBM maps for OffLine simulation
	{
		for (j=0; j<g_iY_LBM_MapSize; j++)
		{
			g_BB.SFAOutput[i][j][iAlgCode][g_BB.iCycle]=g_BB.iaLBM[i][j][0];
			g_BB.iaLBM[i][j][0]=0;
		}
	}
}

/*****************************************************************************
* Name: CreateLS_PPGM *
* Description: This function creates the PPGM matrix for each LS using the LBM          *
*****************************************************************************/
void CreateLS_PPGM(int iLSNum)
{
	int i, j;
	int iNew, jNew;
	for (i=0; i<g_iX_LBM_MapSize; i++)
	{
		for (j=0; j<g_iY_LBM_MapSize; j++)
		{
			iNew=i+(int)((double)g_BB.iPPGM_X/(double)g_LBMCellSize);
			jNew=j+(int)(((double)g_iY_PPGM_MapSize-(double)g_iY_LBM_MapSize)/2);
			if ((iNew>=0)&&(jNew< g_iY_PPGM_MapSize)&&
				(jNew>=0)&&(iNew<
g_iX_PPGM_MapSize)&&(g_BB.iaLBM[i][j][iLSNum]))
			{
				g_BB.iaLS_PPGM[iNew][jNew][iLSNum]=g_BB.iaLBM[i][j][iLSNum];
			}
//			if (iAlgCode==1)
//				g_BB.iaPI[iNew][jNew]++;


		}
	}
}
```

```
/*******************************************************************************
* Name: SFA_Calc_PM    *
* Description: This function calculates for each LS its reliability according         *
* to the generated map by each algorithm      *
*******************************************************************************/
void SFA_Calc_PM(int iAlg_Index)
{
	int i,j,k;
	int iAlg,Num,SumSquaredError,SquaredFusedSum;
	float fCounterT, fCounterF;

	// If the value at the BB_iaSensorArray[][][SFS_level] array is true then its value is 1
	float fLevelCell;
	// If the value at the BB_iaTemporarySensorArray array is true then the its value is 1
	float fFusedCell;

	float PM_True_False;  // Found True but was False.
	float PM_True_True;   // Found True And was True.
	float PM_False_False; // Found False And was False.
	float PM_False_True;  // Found False but was true.
	float PM_UM, NewPM1, NewPM1_Old;
	float OldUM=0;


	iAlg=iAlg_Index;

	for (k=1;k<=g_iTotalNumOfLS;k++)
	{
		PM_True_False=0; // The Local Map Found True  But the fused map determined False.
		PM_True_True=0;  // The Local Map Found True  And the fused map determined True.
		PM_False_False=0;// The Local Map Found False And the fused map determined False.
		PM_False_True=0; // The Local Map Found False But the fused map determined True.


		for ( i=0;i<g_iX_LBM_MapSize;i++)
		{
			for ( j=0;j<g_iY_LBM_MapSize;j++)
			{
				fLevelCell=0;
				fFusedCell=0;

				if (g_BB.iaLBM[i][j][0]) //fused map
				{
					fFusedCell=1;

				}


				//	g_BB.BB_iaTemporarySensorArray[i][j]=0;// Set the array values to 0
				if(g_BB.iaLBM[i][j][k]) // LSv  Map
				{
					fLevelCell=1;
				}

				if(fLevelCell>fFusedCell)//  Found True but was False.
					PM_True_False++;

				if((fLevelCell==fFusedCell)&&(fLevelCell==1))//   Found True And was True.
					PM_True_True++;
```

```
                    if((fLevelCell==fFusedCell)&&(fLevelCell==0))// Found False And was False.
                        PM_False_False++;

                    if(fLevelCell<fFusedCell)//  Found False but was true
                        PM_False_True++;
                } // End (for (j))
            } // End (for (i))
            fCounterT=(PM_True_True+PM_False_True);
            fCounterF=(PM_False_False+PM_True_False);

            if (fCounterT>0)
            {
                PM_True_True=PM_True_True/fCounterT;
                PM_False_True=PM_False_True/fCounterT;
            }

            if (fCounterF>0)
            {
                PM_False_False=PM_False_False/fCounterF;
                PM_True_False=PM_True_False/fCounterF;
            }
            if (fCounterT==0)
            {
                PM_True_True= PM_False_False;
                PM_False_True=1- PM_False_False;
            }

            if (fCounterF==0)
            {
                PM_False_False= PM_True_True;
                PM_True_False=1- PM_True_True;
            }

            /*
            Explanations for the fSFA_PM 4D array:[i][j][k][l]
            i - stands for maximum number of robot positions
            j - stands for number of 5 SFA (,i.e.,, OR, AND , MOST, FL, AFL)
            k - stands for PM: TT, FF, TF ,FT,  Fused measure (0.5*(TT+FF-TF-FT))
            l - stands for total number of LSs
            */

            g_BB.fSFA_PM[g_BB.iCycle][iAlg][0][k-1]=PM_True_True;   // TT
            g_BB.fSFA_PM[g_BB.iCycle][iAlg][1][k-1]=PM_False_False; // FF
            g_BB.fSFA_PM[g_BB.iCycle][iAlg][2][k-1]=PM_True_False;  // TF
            g_BB.fSFA_PM[g_BB.iCycle][iAlg][3][k-1]=PM_False_True;  // FT
            g_BB.fSFA_PM[g_BB.iCycle][iAlg][4][k-1]=
            0.5*(PM_True_True+PM_False_False-PM_True_False-PM_False_True); // Fused measure

            if (g_BB.NewPM[k][4][g_BB.iCycle]>=0)
                OldUM=g_BB.NewPM[k][iAlg][4][g_BB.iCycle];
            PM_UM=0.5*(PM_True_True+PM_False_False-PM_True_False-PM_False_True);


            if (iAlg==1 ||iAlg==3)
            g_BB.NewPM[k][iAlg][4][g_BB.iCycle]=0.5*(OldUM+PM_UM);

        } // End (for (k))
}
```

```
/*********************************************************************************
* Name: SaveGGM          *
* Description: This function save all the data into the hard disk            *
*********************************************************************************/
void SaveGGM()
{

	FILE *f;
	char string[40];
	char cLocNum[6];
	char Counter[6];

	int i, j,k,l;
	ofstream output;
	char fname[60];

	// Saving all path planning grid maps
	for (k=0; k<(g_TotalNumberOfAlgorithms+1); k++)
	{
		_itoa(k, cLocNum, 10 ); // Converting pic number (int) into string.
		_itoa(g_BB.iCycle, Counter, 10 ); // Converting pic number (int) into string.
		strcpy( string, "PPGM");
		strcat( string, cLocNum);
		strcat( string, ".data" );
		f=fopen(string,"w");

		for(i = 0; i<g_iX_PPGM_MapSize ; i++ )
		{
			for(j = 0; j <g_iY_PPGM_MapSize; j++ )
			{
				fprintf(f,"%d",g_BB.iaPPGM[i][j][k]);
				fprintf(f," ");
			 }//j
		   fprintf(f,"\n");
		 }//i
		fclose(f);
	}//k


	// Saving all path planning grid maps for each LS
	for (k=0; k<(g_iTotalNumOfLS+1); k++)
	{
		_itoa(k, cLocNum, 10 ); // Converting pic number (int) into string.
		_itoa(g_BB.iCycle, Counter, 10 ); // Converting pic number (int) into string.
		strcpy( string, "LS_PPGM");
		strcat( string, cLocNum);
		strcat( string, ".data" );
		f=fopen(string,"w");

		for(i = 0; i<g_iX_PPGM_MapSize ; i++ )
		{
			for(j = 0; j <g_iY_PPGM_MapSize; j++ )
			{
				fprintf(f,"%d",g_BB.iaLS_PPGM[i][j][k]);
				fprintf(f," ");
			 }//j
		   fprintf(f,"\n");
		 }//i
		fclose(f);
	}
```

```
// Saving the local binary maps
for (k=0; k<g_BB.iCycle; k++)
{
	for (l=0; l<g_iTotalNumOfLS; l++)
	{
		_itoa(k, Counter, 10);			// Converting pic number (int) into string.
		_itoa(l, cLocNum, 10);			// Converting pic number (int) into string.
		strcpy( string, "LBM");
		strcat( string, Counter);//k - counter
		strcat( string, "_");
		strcat( string, cLocNum);//l - g_iTotalNumOfLS
		strcat( string, ".data" );
		f=fopen(string,"w");
		for(i = 0; i <g_iX_LBM_MapSize ; i++ )
		{
			for(j = 0; j < g_iY_LBM_MapSize; j++ )
			{
				fprintf(f,"%d",g_BB.iaLogicalSensorMap[i][j][l][k]);
				fprintf(f," ");
			}
			fprintf(f,"\n");
		}
		fclose(f);
	}
}
for (k=0; k<g_BB.iCycle; k++)// Saving the sensor fusion algorithms maps (during the process)
{
	for (l=0; l<5; l++) // l - algorithm{
		_itoa(k, Counter, 10); // Converting pic number (int) into string.
		_itoa(l, cLocNum, 10); // Converting pic number (int) into string.
		strcpy( string, "SFA");
		strcat( string, Counter);//k - counter
		strcat( string, "_");
		strcat( string, cLocNum);//l - algorithm
		strcat( string, ".data" );
		f=fopen(string,"w");
		for(i = 0; i <g_iX_LBM_MapSize ; i++ )
		{
			for(j = 0; j < g_iY_LBM_MapSize; j++ )
			{
				fprintf(f,"%d",g_BB.SFAOutput[i][j][l][k]);
				fprintf(f," ");
			}
			fprintf(f,"\n");
		}
		fclose(f);
	}
}
// Saving the robot position
f=fopen("RobotPos.data","w");
for (i=0; i<(g_BB.iCycle); i++)
{
	fprintf(f,"%f ",g_BB.faRobotPos[i][0]);
	fprintf(f," %f",g_BB.faRobotPos[i][1]);
	fprintf(f,"\n");
}
fclose(f);

//Saving data of the new algorithm
```

```
//Saving the AvgMaps
	for (k=0; k<g_BB.iCycle;k++)
	{
		strcpy(string, "AvgMap");
		_itoa(k,Counter,10);
		strcat(string,Counter);
		strcat(string,".data");
		f=fopen(string,"w");
		for (i=0;i<g_SickGridSizeX ; i++)
		{
			for (j=0;j<g_SickGridSizeY; j++)
				fprintf(f,"%d " ,g_BB.AvgMap[i][j][k]);
			fprintf(f,"\n");
		}
		fclose(f);
	}
}

/*******************************************************************************
* Name: FuzzyLogicAlgorithm       *
* Description: This function is an algorithm base on the FL theory for fusing the data.       *
*******************************************************************************/
void FuzzyLogicAlgorithm(int plevel){
	float fFalseAreaTotal=0;
	float fTrueAreaTotal=0;
	float fFalseCOMValue=0;
	float fTrueCOMValue=0;
	int level=plevel;

	FuzzyLogic FL_TT(cf_TT);
	FuzzyLogic FL_FF(cf_FF);
	FuzzyLogic FL_FT(cf_FT);
	FuzzyLogic FL_TF(cf_TF);
	FuzzyLogic FL_TRUE(cf_TRUE);
	FuzzyLogic FL_FALSE(cf_FALSE);

	FL_TT.FLInsCrispVal(g_BB.faTrueFalse[level][1]); // SFS_True_True
	FL_FF.FLInsCrispVal(g_BB.faTrueFalse[level][2]); // SFS_False_False
	FL_TF.FLInsCrispVal(g_BB.faTrueFalse[level][3]); // SFS_True_False
	FL_FT.FLInsCrispVal(g_BB.faTrueFalse[level][4]); // SFS_False_True

	//**************   The Rules    ****************
	// [F/F,High]=>[False,High]
	FL_FF.FL_Crisp2Fuzzy("High")>>FL_FALSE.FLInsFuzzyName("High");
	//fFalseAreaTotal=fFalseAreaTotal+(FL_FALSE.FuzzyLogicGetAraeValue());
	fFalseCOMValue=fFalseCOMValue+
		FL_FALSE.FuzzyLogicGetAraeValue()*FL_FALSE.FuzzyLogicGetCenterOfMassCrisp();

	// [F/F,Avarage]=>[False,Avarage]
	FL_FF.FL_Crisp2Fuzzy("Avarage")>>FL_FALSE.FLInsFuzzyName("Avarage");
	//fFalseAreaTotal=fFalseAreaTotal+(FL_FALSE.FuzzyLogicGetAraeValue());
	fFalseCOMValue=fFalseCOMValue+
		FL_FALSE.FuzzyLogicGetAraeValue()*FL_FALSE.FuzzyLogicGetCenterOfMassCrisp();

	// [F/F,Low]=>[False,Low]
	FL_FF.FL_Crisp2Fuzzy("Low")>>FL_FALSE.FLInsFuzzyName("Low");
	//fFalseAreaTotal=fFalseAreaTotal+(FL_FALSE.FuzzyLogicGetAraeValue());
	fFalseCOMValue=fFalseCOMValue+
		FL_FALSE.FuzzyLogicGetAraeValue()*FL_FALSE.FuzzyLogicGetCenterOfMassCrisp();
```

```
// [F/T,High]=>[False,Low]
FL_FT.FL_Crisp2Fuzzy("High")>>FL_FALSE.FLInsFuzzyName("Low");
//fFalseAreaTotal=fFalseAreaTotal+(FL_FALSE.FuzzyLogicGetAraeValue());
fFalseCOMValue=fFalseCOMValue+
        FL_FALSE.FuzzyLogicGetAraeValue()*FL_FALSE.FuzzyLogicGetCenterOfMassCrisp();

// [F/T,Avarage]=>[False,Avarage]
FL_FT.FL_Crisp2Fuzzy("Avarage")>>FL_FALSE.FLInsFuzzyName("Avarage");
//fFalseAreaTotal=fFalseAreaTotal+(FL_FALSE.FuzzyLogicGetAraeValue());
fFalseCOMValue=fFalseCOMValue+
        FL_FALSE.FuzzyLogicGetAraeValue()*FL_FALSE.FuzzyLogicGetCenterOfMassCrisp();

// [F/T,Low]=>[False,High]
FL_FT.FL_Crisp2Fuzzy("Low")>>FL_FALSE.FLInsFuzzyName("High");
//fFalseAreaTotal=fFalseAreaTotal+(FL_FALSE.FuzzyLogicGetAraeValue());
fFalseCOMValue=fFalseCOMValue+
        FL_FALSE.FuzzyLogicGetAraeValue()*FL_FALSE.FuzzyLogicGetCenterOfMassCrisp();

// [T/T,High]=>[True,High]
FL_TT.FL_Crisp2Fuzzy("High")>>FL_TRUE.FLInsFuzzyName("High");
//fTrueAreaTotal=fTrueAreaTotal+(FL_TRUE.FuzzyLogicGetAraeValue());
fTrueCOMValue=fTrueCOMValue+
        FL_TRUE.FuzzyLogicGetAraeValue()*FL_TRUE.FuzzyLogicGetCenterOfMassCrisp();

// [T/T,Avarage]=>[True,Avarage]
FL_TT.FL_Crisp2Fuzzy("Avarage")>>FL_TRUE.FLInsFuzzyName("Avarage");
//fTrueAreaTotal=fTrueAreaTotal+(FL_TRUE.FuzzyLogicGetAraeValue());
fTrueCOMValue=fTrueCOMValue+
        FL_TRUE.FuzzyLogicGetAraeValue()*FL_TRUE.FuzzyLogicGetCenterOfMassCrisp();

// [T/T,Low]=>[True,Low]
FL_TT.FL_Crisp2Fuzzy("Low")>>FL_TRUE.FLInsFuzzyName("Low");
//fTrueAreaTotal=fTrueAreaTotal+(FL_TRUE.FuzzyLogicGetAraeValue());
fTrueCOMValue=fTrueCOMValue+
        FL_TRUE.FuzzyLogicGetAraeValue()*FL_TRUE.FuzzyLogicGetCenterOfMassCrisp();

// [T/F,High]=>[True,Low]
FL_TF.FL_Crisp2Fuzzy("High")>>FL_TRUE.FLInsFuzzyName("Low");
//fTrueAreaTotal=fTrueAreaTotal+(FL_TRUE.FuzzyLogicGetAraeValue());
fTrueCOMValue=fTrueCOMValue+
        FL_TRUE.FuzzyLogicGetAraeValue()*FL_TRUE.FuzzyLogicGetCenterOfMassCrisp();

// [T/F,Avarage]=>[True,Avarage]
FL_TF.FL_Crisp2Fuzzy("Avarage")>>FL_TRUE.FLInsFuzzyName("Avarage");
//fTrueAreaTotal=fTrueAreaTotal+(FL_TRUE.FuzzyLogicGetAraeValue());
fTrueCOMValue=fTrueCOMValue+
        FL_TRUE.FuzzyLogicGetAraeValue()*FL_TRUE.FuzzyLogicGetCenterOfMassCrisp();

// [T/F,Low]=>[True,High]
FL_TF.FL_Crisp2Fuzzy("Low")>>FL_TRUE.FLInsFuzzyName("High");
//fTrueAreaTotal=fTrueAreaTotal+(FL_TRUE.FuzzyLogicGetAraeValue());
fTrueCOMValue=fTrueCOMValue+
        FL_TRUE.FuzzyLogicGetAraeValue()*FL_TRUE.FuzzyLogicGetCenterOfMassCrisp();


g_BB.fFalseAccuracy[level]=fFalseCOMValue; // Updating the data at the BB.
g_BB.fTrueAccuracy[level]=fTrueCOMValue; // Updating the data at the BB.

}
```

```
/*******************************************************************************
* Name: Calculating_FL_TruthTable          *
* Description: Calculating the truth table,     *
*******************************************************************************/
void Calculating_FL_TruthTable()
{
	int i,j,iTempValue;//,iTempTTValue;
	char buffer[10];
	int iTempTable[128];
	float fTrueValue,fFalseValue;

	// This loop calculte the cell number in a binary mode
	for (i=0;i<pow(2,g_iTotalNumOfLS);i++)
	{
		_itoa(i,buffer,2);
		iTempTable[i]=atoi(buffer);
	}

	// This loop distribute the binary numbers in to single one '0' and '1'.
	for (i=0;i<pow(2,g_iTotalNumOfLS);i++)
	{
		for (j=1;j<(g_iTotalNumOfLS+1);j++)
		{
			if((iTempTable[i]%10)==0)
				iTempValue=0;
			else
				iTempValue=1;
			g_BB.fTTValue[j][i]=(float)iTempValue;
			iTempTable[i]=iTempTable[i]/10;
		}//	for (j=1;j<6;j++)
	}//	for (i=0;i<32;i++)

	// Calculating the total values as function of the sensors outputs and the rules
	for (i=0;i<pow(2,g_iTotalNumOfLS);i++)
	{
		//iTempTTValue=g_BB.BB_fTTValue[1][i];
		fTrueValue=0;
		fFalseValue=0;

		for (j=1;j<=g_iTotalNumOfLS;j++)
		{
			if(g_BB.fTTValue[j][i]==0)
				fFalseValue=fFalseValue+g_BB.fFalseAccuracy[j];
			else
				fTrueValue=fTrueValue+g_BB.fTrueAccuracy[j];
		}
		if (i==0)
			g_BB.fTTValue[0][i]=0;
		else if (i==(pow(2,g_iTotalNumOfLS)-1))
			g_BB.fTTValue[0][i]=1;
		else //(i>0)
		{
			if(fTrueValue<fFalseValue)
				g_BB.fTTValue[0][i]=0;
			else
				g_BB.fTTValue[0][i]=1;
		}
	}//	for (i=0;i<64;i++)
}
```

```
/*******************************************************************************
* Name: CalculatingTrueAndFalseValues     *
* Description: This function Compare the new data at this level with the integrated data    *
* This function is the adaptive part of the system  and determine the following parameters*
* SFS_True_False The Local Map Found True But the fused map determined False         *
* SFS_True_True The Local Map Found True And the fused map determined True          *
* SFS_False_False The Local Map Found False And the fused map determined False       *
* SFS_False_True The Local Map Found False But the fused map determined True         *
*******************************************************************************/
void CalculatingTrueAndFalseValues(int SFS_Level)
{
	unsigned short int i,j;
	float fCounterT,fCounterF;
	float fLevelCell; // If the value at the BB_iaSensorArray[][][SFS_level] array is true then its value is 1
	float fFusedCell; // If the value at the BB_iaTemporarySensorArray array is true then the its value is 1
	float fOldFF,fOldTT,fOldTF,fOldFT;

	float SFS_True_False;  // Found True but was False.
	float SFS_True_True;   // Found True And was True.
	float SFS_False_False; // Found False And was False.
	float SFS_False_True;  // Found Fasle but was true.


	SFS_True_False=0; // The Local Map Found True  But the fused map determined False.
	SFS_True_True=0;  // The Local Map Found True  And the fused map determined True.
	SFS_False_False=0;// The Local Map Found False And the fused map determined False.
	SFS_False_True=0; // The Local Map Found Fasle But the fused map determined True.

	if(g_BB.iAFL_Flag==0) // Regular AFL
	{
		if (g_BB.faTrueFalseRegular[SFS_Level][1]>=0)
			fOldTT=g_BB.faTrueFalseRegular[SFS_Level][1];
		if (g_BB.faTrueFalseRegular[SFS_Level][2]>=0)
			fOldFF=g_BB.faTrueFalseRegular[SFS_Level][2];
		if (g_BB.faTrueFalseRegular[SFS_Level][3]>=0)
			fOldTF=g_BB.faTrueFalseRegular[SFS_Level][3];
		if (g_BB.faTrueFalseRegular[SFS_Level][4]>=0)
			fOldFT=g_BB.faTrueFalseRegular[SFS_Level][4];
	}
	else
	{
		if (g_BB.faTrueFalse[SFS_Level][1]>=0)
			fOldTT=g_BB.faTrueFalse[SFS_Level][1];
		if (g_BB.faTrueFalse[SFS_Level][2]>=0)
			fOldFF=g_BB.faTrueFalse[SFS_Level][2];
		if (g_BB.faTrueFalse[SFS_Level][3]>=0)
			fOldTF=g_BB.faTrueFalse[SFS_Level][3];
		if (g_BB.faTrueFalse[SFS_Level][4]>=0)
			fOldFT=g_BB.faTrueFalse[SFS_Level][4];

	}

	for ( i=0;i<g_iX_LBM_MapSize;i++)
	{
		for ( j=0;j<g_iY_LBM_MapSize;j++)
		{
			fLevelCell=0;
			fFusedCell=0;
			if(g_BB.iaLBM[i][j][0]) // Fused Map
				fFusedCell=1;
```

```
			if(g_BB.iaLBM[i][j][SFS_Level])
				fLevelCell=1;
			if(fLevelCell>fFusedCell)//  Found True but was False.
				SFS_True_False++;
			if((fLevelCell==fFusedCell)&&(fLevelCell==1))//   Found True And was True.
				SFS_True_True++;
			if((fLevelCell==fFusedCell)&&(fLevelCell==0))// Found False And was False.
				SFS_False_False++;
			if(fLevelCell<fFusedCell)//  Found Fasle but was true
				SFS_False_True++;
		}
}
fCounterT=(SFS_True_True+SFS_False_True);
fCounterF=(SFS_False_False+SFS_True_False);

if (fCounterT>0)
{
   SFS_True_True=SFS_True_True/fCounterT;
		SFS_False_True=SFS_False_True/fCounterT;
}

if (fCounterF>0)
{
		SFS_False_False=SFS_False_False/fCounterF;
		SFS_True_False=SFS_True_False/fCounterF;
}
if (fCounterT==0)
{
		SFS_True_True= SFS_False_False;
		SFS_False_True=1- SFS_False_False;
}

if (fCounterF==0)
{
		SFS_False_False= SFS_True_True;
		SFS_True_False=1- SFS_True_True;
}

/*Explanation about the BB_faTrueFalse[(1+g_NumberOfModules)][7] array:
Cell number 0 is for: Free
Cell number 1 is for: TT Value
Cell number 2 is for: FF Value
Cell number 3 is for: TF Value
Cell number 4 is for: FT Value
Cell number 5 is for: TRUE Value
Cell number 6 is for: FALSE Value   */

if(g_BB.iAFL_Flag==0) // Regular AFL
{
g_BB.faTrueFalseRegular[SFS_Level][1]=0.5*(SFS_True_True+fOldTT);
g_BB.faTrueFalseRegular[SFS_Level][2]=0.5*(SFS_False_False+fOldFF);
g_BB.faTrueFalseRegular[SFS_Level][3]=0.5*(SFS_True_False+fOldTF);
g_BB.faTrueFalseRegular[SFS_Level][4]=0.5*(SFS_False_True+fOldFT);
g_BB.fSFA_FL_Regular[g_BB.iCycle][SFS_Level][0]=g_BB.faTrueFalseRegular[SFS_Level][1];   // TT
g_BB.fSFA_FL_Regular[g_BB.iCycle][SFS_Level][1]=g_BB.faTrueFalseRegular[SFS_Level][2]; // FF
g_BB.fSFA_FL_Regular[g_BB.iCycle][SFS_Level][2]=g_BB.faTrueFalseRegular[SFS_Level][3];  // TF
g_BB.fSFA_FL_Regular[g_BB.iCycle][SFS_Level][3]=g_BB.faTrueFalseRegular[SFS_Level][4];  // FT
}

else
```

```
        {
        g_BB.faTrueFalse[SFS_Level][1]=0.5*(SFS_True_True+fOldTT);
        g_BB.faTrueFalse[SFS_Level][2]=0.5*(SFS_False_False+fOldFF);
        g_BB.faTrueFalse[SFS_Level][3]=0.5*(SFS_True_False+fOldTF);
        g_BB.faTrueFalse[SFS_Level][4]=0.5*(SFS_False_True+fOldFT);

        g_BB.fSFA_FL[g_BB.iCycle][SFS_Level][0]=g_BB.faTrueFalse[SFS_Level][1];   // TT
        g_BB.fSFA_FL[g_BB.iCycle][SFS_Level][1]=g_BB.faTrueFalse[SFS_Level][2]; // FF
        g_BB.fSFA_FL[g_BB.iCycle][SFS_Level][2]=g_BB.faTrueFalse[SFS_Level][3];  // TF
        g_BB.fSFA_FL[g_BB.iCycle][SFS_Level][3]=g_BB.faTrueFalse[SFS_Level][4];  // FT
        }
}
```

```
/**
 ** PXC_Camera_Dll_Load.h
 **
 ** Copyright 2001 by Ofir Cohen
 **
 ** E-mail:  oprc@bgumail.bgu.ac.il
 **
 **/

#ifndef __PXC_Camera_Dll_Load_h__
#define __PXC_Camera_Dll_Load_h__

#include <windows.h>
#include <commdlg.h>
#include "ipl.h"

#include "cv.h"
#include "image.h"

#include "pxc.h"
#include "iframe.h"
#include "StaticParameters.h"
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"
#include "Vision_Class.h"

#define PIXEL_TYPE PBITS_RGB24
#define PXC_NAME    "C:\\PXC2\\bin\\pxc_95.dll"
#define FRAME_NAME  "C:\\PXC2\\bin\\frame_32.dll"
#define PXC_NT      "C:\\PXC2\\bin\\pxc2_nt.dll"
extern int videotype;
extern int grab_type;
extern int  ImageMaxX,ImageMaxY,WindowX,WindowY;
extern long fgh;
extern FRAME __PX_FAR *frh;
extern HINSTANCE hLib;
extern PXC pxc;
extern FRAMELIB frame;
extern Vision_Class CAM; // Crearting the CAMERA object
extern BlackBoard g_BB;

// Fuctions definitions
bool AppInit();
void ImageProcessingAlgo1();

#endif
```

```
/**
** PXC_Camera_Dll_Load.cpp
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**
**/
#include "Aria.h"
#include <math.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include "ipl.h"
#include "pxc.h"
#include "iframe.h"
#include <cvlgrfmts.h>
#include "StaticParameters.h"
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "PXC_Camera_Dll_Load.h"
#include <windows.h>
#include "Vision_Class.h"


#define PIXEL_TYPE PBITS_RGB24
#define PXC_NAME  "pxc_95.dll"
#define FRAME_NAME  "frame_32.dll"
#define PXC_NT    "pxc_nt.dll"
extern int videotype;
extern int grab_type;
extern int  ImageMaxX,
ImageMaxY,
WindowX,
WindowY;
extern long fgh;
extern FRAME __PX_FAR *frh;
extern PXC pxc;
extern FRAMELIB frame;
extern ArRobot robot ;
CImage gray; // OpenCV generating the gary CImage type


/*****************************************************************************
* Name:   ImageProcessingAlgo1    *
* Description:  The function has three steps: *
*1. Capturing the image.    *
*2. Image processing algorithm (has two stages).        *
*     2.1 Simple Threshold. *
*     2.2 Two level threshold.           *
*3. Finding the center of mass (COM) for each obstacle, and calculate the           *
*real distance from the camera.       *
*****************************************************************************/
void ImageProcessingAlgo1()
{
	gray.Create(768,576,8);

	IplImage *i_gray = gray.GetImage();

	float temp;
```

```
        temp=float(robot.getY());

        CAM.iaVision_X[CAM.iVision_CameraAngleCode]=(int)(robot.getX()/10);
        CAM.iaVision_Y[CAM.iVision_CameraAngleCode]=(int)(temp*0.231);
        CAM.iaVision_Theta[CAM.iVision_CameraAngleCode]=(int)(robot.getTh());//[Red]

        pxc.Grab(fgh, frh, (short)grab_type);
        IplImage *i_part=iplCreateImageHeader(3,0,IPL_DEPTH_8U,"RGB","RGB",
                IPL_DATA_ORDER_PIXEL,IPL_ORIGIN_TL,  // top left orientation
    IPL_ALIGN_QWORD,768,576,NULL,NULL,NULL,NULL); // not tiled

        int i=CAM.iVision_CameraAngleCode;
        i_part->imageData =(char *)frame.FrameBuffer(frh);
  iplColorToGray(i_part,i_gray); //convert into grayscale
}

/*******************************************************************************
* Name: AppInit  *
* Description:  This function initializes and allocates the Frame grabber PXC200            *
*******************************************************************************/
//BOOL
bool AppInit()
{
        fgh = 0;
        frh = 0L;
        //----------------------------------------------------------------------
        //initialize the library
        //----------------------------------------------------------------------
        if (!imagenation_OpenLibrary(PXC_NAME,&pxc,sizeof(pxc)))
        {
                if (!imagenation_OpenLibrary(PXC_NT,&pxc,sizeof(pxc)))
                {
                        return false;
                }
        }

        if (!imagenation_OpenLibrary(FRAME_NAME,&frame,sizeof(frame)))
        {
                return false;
        }

        //----------------------------------------------------------------------
        //allocate any frame grabber
        //----------------------------------------------------------------------
        fgh = pxc.AllocateFG(-1);
        videotype = pxc.VideoType(fgh);
        switch(videotype) {
case 0:    // no video
case 1:    // NTSC
                grab_type = 0;
                ImageMaxX = 640;
                ImageMaxY = 486;
                break;
case 2:    // CCIR
                grab_type = 0;
                ImageMaxX = 768;
                ImageMaxY = 576;
                break;
        }
        if(GetSystemMetrics(SM_CXSCREEN) <= ImageMaxX) {
```

```
		ImageMaxX/=2;
		ImageMaxY/=2;
	}
	pxc.SetWidth(fgh,(short)ImageMaxX);
	pxc.SetHeight(fgh,(short)ImageMaxY);
	pxc.SetLeft(fgh,0);
	pxc.SetTop(fgh,0);
	pxc.SetXResolution(fgh,(short)ImageMaxX);
	pxc.SetYResolution(fgh,(short)ImageMaxY);

	//-------------------------------------------------------------------------
	//allocate a frame buffer
	//-------------------------------------------------------------------------
	frh = pxc.AllocateBuffer((short)ImageMaxX, (short)ImageMaxY, PIXEL_TYPE);
	return true;
}
```

```
/**
** Vision_Class.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**/

#ifndef __Vision_Class_h__
#define __Vision_Class_h__
#include <time.h>
#include <conio.h>
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"

extern PXC pxc;
extern FRAMELIB frame;
extern  long fgh;
extern FRAME __PX_FAR *frh;
extern int videotype;
extern int grab_type;
extern int  ImageMaxX,
            ImageMaxY,
            WindowX,
            WindowY;

extern BlackBoard g_BB;
void ImageProcessingAlgo3(int);
void ImageProcessingAlgo4(int);

class Vision_Class
{
public:
            void Vision_GridMapCellConversion();
            int iVision_CameraAngleCode;
            int iaVision_X[g_usNumOfCamPos+1]; // Robot X Location
            int iaVision_Y[g_usNumOfCamPos+1]; // Robot Y location
            int iaVision_Theta[g_usNumOfCamPos+1]; // Robot Theta angle [Deg or Rad]
            int iaVision_Phi[g_usNumOfCamPos+1]; // Camera angle [Deg] (Cell number 0 is not in use)
            int
iaVision_NumberOfObstacle[g_usNumOfCamPos+1][g_iTotalNumOfCamLS]; // (Cell number 0 is not in use)
            int
iaVision_XY_CAM_Position[g_usNumOfCamPos+1][g_iMaxNumOfObstacle][2*g_iTotalNumOfCamLS];
// X and Y obstacle location for each camera position ,[g_usNumOfCamPos+1]  - Number of camera position , [10] -
Number of obstacles (10 is MAX),[2]  - Two coordinates for Y and X obstacle's COM
int  iaVision_LocalGridMap[g_CamGridSizeX][g_CamGridSizeY][g_iTotalNumOfCamLS];
            Vision_Class();
            ~Vision_Class();
};
#endif
```

```
/**
** Vision_Class.cpp
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**/

#include <math.h>
#include "Vision_Class.h"
#include "PXC_Camera_Dll_Load.h"

extern PXC pxc;
extern FRAMELIB frame;
extern  long fgh;
extern FRAME __PX_FAR *frh;
extern int videotype;
extern int grab_type;
extern int  ImageMaxX,
ImageMaxY,
WindowX,
WindowY;

extern CImage gray;

CImage bw1;
CImage bw2;
CImage bw3;
CImage Temp;

/**********************************************************************************
* Name: Vision_Class::Vision_Class            *
* Description: Default Constructor  *
**********************************************************************************/
Vision_Class::Vision_Class()
{
	int i,j,k;
	iVision_CameraAngleCode=2;

	for (i=0;i<=g_usNumOfCamPos;i++)
	{
		iaVision_X[i]=0; // Robot X Location
		iaVision_Y[i]=0; // Robot Y location
		iaVision_Theta[i]=0; // Robot Theta angle [Deg or Rad]
		iaVision_Phi[i]=0; // Camera angle [Deg] (Cell number 0 is not in use)
		iaVision_NumberOfObstacle[i][0]=0; // (Cell number 0 is not in use)
		iaVision_NumberOfObstacle[i][1]=0; // (Cell number 0 is not in use)

	}

	for (i=0;i<=g_usNumOfCamPos;i++)
	{
		for (j=0;j<=g_iMaxNumOfObstacle;j++)
		{
			for (k=0; k<2*g_iTotalNumOfCamLS; k++)
			{
				iaVision_XY_CAM_Position[i][j][k]=0;
			}
		}
	}
```

```
	for (i=0; i<g_CamGridSizeY; i++)
	{
		for (j=0; j<g_CamGridSizeX; j++)
		{
			for(k=0; k<g_iTotalNumOfCamLS; k++)
				iaVision_LocalGridMap[i][j][k]=0;
		}
	}
}

/*******************************************************************************
* Name: Vision_Class::~Vision_Class		*
* Description: Default Destructor	*
*******************************************************************************/
Vision_Class::~Vision_Class(){;}

/*******************************************************************************
* Name: ImageProcessingAlgo3	*
* Description: The heart of the image processing, here we do the Erode Dilate	*
* for each photo according to the algorithm number, We find the center of mass for each	*
* algorithm and finds the location of the algorithm according to the calibration process made earlier*
*******************************************************************************/
void ImageProcessingAlgo3(int Alg_Code)
{
	double fCenterOfMassRow;			//center of mass for an obstacle (row)
	double fCenterOfMassCol;			//center of mass for an obstacle (Col)
	long double Xp5=-6.17e-12;			//calibration parameter
	long double Xp4=1.59e-8;		//calibration parameter
	double Xp3=-1.54e-5;		//calibration parameter
	double Xp2=0.007846;			//calibration parameter
	double Xp1=-2.3348;			//calibration parameter
	double Xp0=406.14;			//calibration parameter
	double fTanAlfa;
	double fDisX;				//obstacle distance from camera pivot in X axis
	float fDisY;				//obstacle distance from camera pivot in Y axis
	float fR;				//obstacle distance from camera pivot
	double fRealAngle;			//angle between obstacle
	int AngleCode;
	float CameraAngle;
	int ObsX;
	int ObsY;
	int iObsArea_Min[3]={15000,20000,20000};
	int iObsArea_Max[3]={47000,33000,33000};
	int iDecoyArea_Min[3]={0,600,8000};
	int iDecoyArea_Max[3]={0,1200,20000};

	CvMoments moments;
	double m00;
	CvSeq *contour = NULL;
	CvSeq* copycontour;
	CvMemStorage *storage =cvCreateMemStorage(0);
	int counter;// How many obstacles for each picture
	counter=0;

	int iObsTH_Min,iObsTH_Max;

	if(Alg_Code==0)
	{
		bw1.Create(768,576,8);
```

```
		IplImage *i_gray = gray.GetImage();
		IplImage *i_bw1 = bw1.GetImage();
		iplThreshold(i_gray, i_bw1, 120);
		cvFindContours(i_bw1, storage, &contour, sizeof(CvContour),
				CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
	}

	if(Alg_Code==1)
	{
		iObsTH_Min=120;
		bw2.Create(768,576,8);
		Temp.Create(768,576,8);
		IplImage *i_Temp = Temp.GetImage();
		IplImage *i_gray = gray.GetImage();
		IplImage *i_bw2 = bw2.GetImage();
		iplThreshold(i_gray, i_bw2, iObsTH_Min);
		iplErode(i_bw2, i_bw2, 3); // Clear the obstacle border
		//bw2.Save("d:/users/oren/data/alg1/bw2erode1.bmp");
		//iplDilate(i_bw2, i_bw2,g_BB.Dilate2);
		iplDilate(i_bw2, i_bw2,5);// Make the object thiner
		//iplErode(i_bw2, i_bw2, g_BB.Erode2); // Make the object
		cvFindContours(i_bw2, storage, &contour, sizeof(CvContour),
		CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
	}


	if(Alg_Code==2)
	{
		iObsTH_Min=120;
		//iObsTH_Min=g_BB.iTresholdValue3_Min;
		//iObsTH_Max=g_BB.iTresholdValue3_Max;
		bw3.Create(768,576,8);
		Temp.Create(768,576,8);
		IplImage *i_gray = gray.GetImage();
		IplImage *i_bw3 = bw3.GetImage();
		iplThreshold(i_gray, i_bw3, iObsTH_Min);
		iplErode(i_bw3, i_bw3, 3); // Clear the obstacle border
		iplDilate(i_bw3, i_bw3,  4); // Make the object theaker
		cvFindContours(i_bw3, storage, &contour, sizeof(CvContour),
				CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
	}

	if (CAM.iVision_CameraAngleCode==1)
	{
		AngleCode=4;
		CameraAngle=(float)(50*g_pi/180);
	}
	else
	{
		AngleCode=CAM.iVision_CameraAngleCode-1;
		CameraAngle=(float)((AngleCode*33.4-83.5)*g_pi/180);
	}

	//  Stage 3 - find center of mass for each obstacle in the image, and its real distance from the robot.
	double ContourArea;
	if (contour)
	{//5
		for(copycontour=contour; copycontour!=0; copycontour=copycontour->h_next)
		{//6
			cvContourArea(copycontour, &ContourArea);
```

```
                ContourArea=ContourArea*(-1);
                //Checking if this is an obstacle or decoy

                /*if(ContourArea>iObsArea_Min[Alg_Code] &&
ContourArea<iObsArea_Max[Alg_Code])
                        iObs_Flag=1;

                if(Alg_Code!=0 && ContourArea>iDecoyArea_Min[Alg_Code] &&
ContourArea<iDecoyArea_Max[Alg_Code])
                        iDecoy_Flag=1;
                        if (iObs_Flag==1 || iDecoy_Flag==1)
                        */

                if (ContourArea>iObsArea_Min[Alg_Code] &&
ContourArea<iObsArea_Max[Alg_Code] ||
                        (Alg_Code!=0 && ContourArea>iDecoyArea_Min[Alg_Code] &&
ContourArea<iDecoyArea_Max[Alg_Code]) )
                {//7
                        cvContourMoments(copycontour, &moments);
                        m00=cvGetSpatialMoment(&moments,0,0);
                        fCenterOfMassCol=(cvGetSpatialMoment(&moments, 1,0)/m00)*(-1);
                        fCenterOfMassRow=(cvGetSpatialMoment(&moments, 0,1)/m00)*(-1);
                        // calculating X and Y distance relative to the picture axis
                        if
(!((fCenterOfMassRow<50)||(fCenterOfMassRow>566)||(fCenterOfMassCol<10)||(fCenterOfMassCol>758)))
                        {//8
                                counter++;
                                fTanAlfa= (fCenterOfMassCol-385)/(fCenterOfMassRow+117);
                                fDisX=Xp5*pow(fCenterOfMassRow, 5);
                                fDisX=fDisX+Xp4*pow(fCenterOfMassRow,4);
                                fDisX=fDisX+Xp3*pow(fCenterOfMassRow,3);
                                fDisX=fDisX+Xp2*pow(fCenterOfMassRow,2);
                                fDisX=fDisX+ Xp1*fCenterOfMassRow;
                                fDisX=fDisX+Xp0;
                                fDisY = float(60.459*fTanAlfa+0.2418);
                                fR=float(pow(fDisX,2)+pow(fDisY,2));
                                fR=(float)sqrt(fR);
                                fRealAngle=float(atan(fDisY/fDisX)); //[Rad]

                                ObsX=(int)(cos(fRealAngle+CameraAngle)*fR);
                                ObsY=(int)(sin(fRealAngle+CameraAngle)*fR);
        CAM.iaVision_XY_CAM_Position[AngleCode][counter][Alg_Code*2]=ObsX;

        CAM.iaVision_XY_CAM_Position[AngleCode][counter][Alg_Code*2+1]=ObsY;

                        }//8
                }//7
            }//  6 end for
        }// 5 end if
        CAM.iaVision_NumberOfObstacle[AngleCode][Alg_Code]=counter;
        counter=0;
}
```

```
/*******************************************************************************
* Name: ImageProcessingAlgo4      *
* Description: This function transform the maps and built for each obstacle a circle around it.       *
*******************************************************************************/

void ImageProcessingAlgo4(int Alg_Code)
{
        double Theta1;                          // Vehicle Old Theta Position
        double Theta2;                          // Vehicle New Theta Position
        double DeltaX;
        double DeltaY;
        double DeltaTheta;
        double X1_obj;          // Rotation of the object from old position to New position
        double Y1_obj;          // Rotation of the object from old position to New position

        double X12;             // Rotation of the vehicle from old position to new position
        double Y12;             // Rotation of the vehicle from old position to new position

        double Xm1;             // Object Old X Position (Relative to the OLD vehicle)
        double Ym1;             // Object Old Y Position (Relative to the OLD vehicle)
        double Xm2;             // Object new X Position (Relative to the NEW vehicle)
        double Ym2;             // Object new Y Position (Relative to the NEW vehicle)
        int ObstcaleLocInMapX;
        int ObstcaleLocInMapY;
        int i,j,Theta,k;
        double phi,xTag,yTag,x0,y0;
        int x,y,Last;                   // projection of range 'r' on X and Y axis


                if (g_BB.iCycle%2==0)
                        Last=4;
                else
                        Last=1;


        for (i=0; i<g_CamGridSizeX ; i++)
        {
                for (j=0; j<g_CamGridSizeY; j++)
                        CAM.iaVision_LocalGridMap[i][j][Alg_Code]=0;
        }

        for (i=1; i<=g_usNumOfCamPos; i++)
        {//2
                if(CAM.iaVision_NumberOfObstacle[i][Alg_Code]!=0)
                {//3
                        for (j=1; j<=CAM.iaVision_NumberOfObstacle[i][Alg_Code]; j++)
                        { // 4
                                Xm1=CAM.iaVision_XY_CAM_Position[i][j][Alg_Code*2];
                                Ym1=CAM.iaVision_XY_CAM_Position[i][j][Alg_Code*2+1];
                                Theta1=CAM.iaVision_Theta[i]*g_pi/180;
                                Theta2=CAM.iaVision_Theta[Last]*g_pi/180;
                                DeltaX=CAM.iaVision_X[Last]-CAM.iaVision_X[i];
                                DeltaY=CAM.iaVision_Y[Last]-CAM.iaVision_Y[i];
                                if (fabs(Theta2-Theta1)<g_pi)
                                        DeltaTheta=Theta2-Theta1;
                                else
                                {
                                        if(Theta2>Theta1)
                                                DeltaTheta=(Theta2-Theta1)-2*g_pi;
                                        else
```

```
                    DeltaTheta=2*g_pi-(Theta2-Theta1);
                }
                X1_obj=Xm1*cos(DeltaTheta)+Ym1*sin(DeltaTheta);
                Y1_obj=-Xm1*sin(DeltaTheta)+Ym1*cos(DeltaTheta);
                X12=DeltaX*cos(DeltaTheta)+DeltaY*sin(DeltaTheta);
                Y12=-DeltaX*sin(DeltaTheta)+DeltaY*cos(DeltaTheta);
                Xm2=X1_obj-X12;
                Ym2=Y1_obj-Y12;
                CAM.iaVision_XY_CAM_Position[i][j][Alg_Code*2]=(int)Xm2;
                CAM.iaVision_XY_CAM_Position[i][j][Alg_Code*2+1]=(int)Ym2;
//    building the map using the array iaVision_XY_CAM_Position
ObstcaleLocInMapX=
(int)(CAM.iaVision_XY_CAM_Position[i][j][Alg_Code*2]);
ObstcaleLocInMapY=
((int)(CAM.iaVision_XY_CAM_Position[i][j][Alg_Code*2+1]+
LBM_cm_SizeY/2));

                for (k=0 ;k<15;k=k+2)//k->>from 0 to obstacle radios
                {
                    for (Theta=0;Theta<360;Theta=Theta+30)
                    {
                        phi=g_pi/180*Theta;
                        xTag=(double)k*cos(phi);
                        yTag=(double)k*sin(phi);
                        x0=xTag+(double)ObstcaleLocInMapX;
                        y0=yTag+(double)ObstcaleLocInMapY;
                        x=(int)(x0/(double)g_CamCellSize);
                        y=(int)(y0/(double)g_CamCellSize);

                if ((x>=0)&&(x<g_CamGridSizeX)&&(y>0)&&(y<=g_CamGridSizeY))
                        {
                    /*if (Alg_Code==2)
                        CAM.iaVision_LocalGridMap[x][g_CamGridSizeY-
y][Alg_Code]=0; //CAM3=Empty
                    else
                        CAM.iaVision_LocalGridMap[x][g_CamGridSizeY-
y][Alg_Code]=1;
                        */
                    //finding how many times each cell is sampled
                    CAM.iaVision_LocalGridMap[x][g_CamGridSizeY-y][Alg_Code]++;
                    //CAM.iaVision_LocalGridMap[x][g_CamGridSizeY-y][Alg_Code]=1;


                        }
                    }//end for (theta)
                }//end for (k)
            }//4 j - iaVision_NumberOfObstacle
        }//3 if iaVision_NumberOfObstacle>0
    }//2 i - g_usNumOfCamPos

    CAM.iaVision_X[0]=CAM.iaVision_X[Last];
    CAM.iaVision_Y[0]=CAM.iaVision_Y[Last];
    CAM.iaVision_Theta[0]=CAM.iaVision_Theta[Last];

}//1
```

```
/*********************************************************************************
* Name: Vision_Class::Vision_GridMapCellConversion          *
* Description: This function convert the maps into a one grid cell size    *
*********************************************************************************/
void Vision_Class::Vision_GridMapCellConversion()
{
	int i, j, k, m, n;
	int iResParam;

	g_BB.iLBM_X_Old=g_BB.iLBM_X_New;
	g_BB.iLBM_Y_Old=g_BB.iLBM_Y_New;

	g_BB.iLBM_X_New=CAM.iaVision_X[0];
	g_BB.iLBM_Y_New=CAM.iaVision_Y[0];

	g_BB.iPPGM_X=CAM.iaVision_X[0]; // Saving the X coardinate for the PPGM [cm]
	g_BB.iPPGM_Y=CAM.iaVision_Y[0]; // Saving the Y coardinate for the PPGM
	g_BB.iPPGM_Theta=CAM.iaVision_Theta [0]; // Saving the Theta coardinate for teh PPGM

	iResParam= (int)((float)g_LBMCellSize/(float)g_CamCellSize);
	for (k=0; k<g_iTotalNumOfCamLS; k++)
	{
		g_BB.bLGM_NewDataFlag[k+1+g_iTotalNumOfUsLS+g_iTotalNumOfSiLS]=1; // turn the flag
on
		for (i=0; i<g_CamGridSizeX; i++)
		{
			for (j=0; j<g_CamGridSizeY; j++)
			{
				for (m=0; m<iResParam; m++)
				{
					for (n=0; n<iResParam; n++)
	g_BB.iaLBM[(i*iResParam+m)][(j*iResParam+n)][k+1+g_iTotalNumOfUsLS+g_iTotalNumOfSiLS]=CA
M.iaVision_LocalGridMap[i][j][k];
				}
			}
		}
	}//end for (k - g_iTotalNumOfCamLS)

	for (i=0;i<=g_usNumOfCamPos;i++)
	{
		if (i>0){
			CAM.iaVision_X[i]=0; // Robot X Location
			CAM.iaVision_Y[i]=0; // Robot Y location
			CAM.iaVision_Theta[i]=0; // Robot Theta angle [Deg or Rad]
		}
		CAM.iaVision_Phi[i]=0; // Camera angle [Deg] (Cell number 0 is not in use)
		for (k=0;k<g_iTotalNumOfCamLS;k++){
			CAM.iaVision_NumberOfObstacle[i][k]=0; // (Cell number 0 is not in use)
			for (j=0;j<=g_iMaxNumOfObstacle;j++)
				CAM.iaVision_XY_CAM_Position[i][j][k]=0;
		}
	}

	for(i = 0; i <g_CamGridSizeX ; i++ ){
		for(j = 0; j < g_CamGridSizeY; j++ ){
			CAM.iaVision_LocalGridMap[i][j][0]=0;
			CAM.iaVision_LocalGridMap[i][j][1]=0;
		}
	}
}
```

```
/**
** UltraSonic_Class.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**/

#ifndef __UltraSonic_Class_h__
#define __UltraSonic_Class_h__

#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"

extern BlackBoard g_BB;

class UltraSonic_Class
{
public:

	int iUS_X; // Robot X Location
	int iUS_Y; // Robot Y location
	int iUS_Theta; // Robot Theta angle [Deg or Rad]
	int sonarNum[6];//
	//This array represents Six local physical maps and 4 fused LGMs.
	//Level 0: Sensor Num 2
	//Level 1: Sensor Num 3
	//Level 2: Sensor Num 4
	//Level 3: Sensor Num 5
	//Level 4: Sensor Num 6
	//Level 5: Sensor Num 7
	//Level 6: Fusion Algorithm AND
	//Level 7: Miguel Ribo and Axel Pinz, 2001, A comparison of three uncertainty
	//   calculi for building sonar based occupancy grids,
	//   Robotics and Automation systems 35: 201-209
	//Level 8: Fifith LS all zeros or all ones.

	unsigned short usaUS_PhysicalSensor[g_USGridSizeX][g_USGridSizeY][6+g_iTotalNumOfUsLS+1];
	unsigned short NewusaUS_PhysicalSensor[g_USGridSizeX][g_USGridSizeY][6+g_iTotalNumOfUsLS+1];
	int iaUS_Range[6]; //sensor data

	float faUS_SonarLoc[6][3]; //Locataion of each sonar from the center of the camera
				//Row 0 : X;
				//Row 1 : Y;
				//Row 2 : Theta;

	int iaUS_NumCellCccupy[6];
	int iaUS_NumCellEmpty[6];
	void US_ReadDataFromUS();
	void US_SFA_LogicalOR();
	void US_SFA_ProbabilisticApproach();
	void US_GridMapCellConversion();
	UltraSonic_Class();  // Default constructor
	~UltraSonic_Class(); // Default distructor
};
#endif
```

```
/**
** UltraSonic_Class.cpp
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**/

#include "Aria.h"
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"
#include "UltraSonic_Class.h"

extern UltraSonic_Class US;
extern ArRobot robot ;


/*******************************************************************************
* Name: US_ReadDataFromUS      *
* Description: This function reads the data form the sonar        *
*******************************************************************************/
void UltraSonic_Class::US_ReadDataFromUS(){
	int i,j,Theta,k;
	double phi,xTag,yTag,x0,y0;
	int x,y; // projection of range 'r' on X and Y axis
	static iCycle;
	iCycle++;
	FILE *f;
	char fname[60],cLocNum[10];

	for (k=0; k<8; k++)//Initialzaing the maps
	{
		for(i=0; i<g_USGridSizeX; i++)
		{
			for (j=0; j<g_USGridSizeY; j++)
				usaUS_PhysicalSensor[i][j][k]=500;
			//	usaUS_PhysicalSensor[i][j][k]=2;
		}
	}

	iUS_X=(int)(robot.getX()*0.1);
	iUS_Y=(int)(robot.getY()*0.231); // relative transformation based on the center of mass point

	g_BB.iPPGM_X=(int)(robot.getX()*0.1); // cm
	g_BB.iPPGM_Y=(int)(robot.getY()*0.231);
	g_BB.iPPGM_Theta=(int)robot.getTh();

	printf("Start reading sonar\n");
	for (i=0;i<6; i++)
	{
		iaUS_Range[i]=(int)(0.1*robot.getSonarRange(sonarNum[i]));// Converting to [cm]

	   printf("Sonar %d Range = %d \n",i, iaUS_Range[i]);
		x=0;
		y=0;
		phi=0;
		int flag=0;
		// Define the 'steps' fo range chacking
```

```
		for (k=1 ;k<=(iaUS_Range[i])+10;k++)
		{
			for (Theta=-5 ;Theta<=5;Theta++)
			{
				phi=(g_pi/180*(Theta+faUS_SonarLoc[i][2]));

				xTag=k*cos(phi)/*-k*sin(phi)*/;
				yTag=/*k*cos(phi)+*/k*sin(phi);

				x0=xTag+faUS_SonarLoc[i][0];
				y0=yTag+faUS_SonarLoc[i][1];

				x=(int)(x0/g_USCellSize);

				if((Theta+faUS_SonarLoc[i][2])>0)
					y=(int)(ceil((y0/g_USCellSize)+(0.5*g_USGridSizeY)))-1;
				else
					y=(int)(floor((y0/g_USCellSize)+(0.5*g_USGridSizeY)));

				if ((x>=0) && (x<g_USGridSizeX) &&
					(y>=0) && (y<g_USGridSizeY))
				{

					if (k<=iaUS_Range[i])
					{
						if (usaUS_PhysicalSensor[x][y][i]!=500)
							if (usaUS_PhysicalSensor[x][y][i]>=1)
									usaUS_PhysicalSensor[x][y][i]--;
							else
								usaUS_PhysicalSensor[x][y][i]=0;
						else
							usaUS_PhysicalSensor[x][y][i]=0;

					}//if smaller then range
					else
					{
						//usaUS_PhysicalSensor[x][y][i]=1;

						if (usaUS_PhysicalSensor[x][y][i]==500)
							usaUS_PhysicalSensor[x][y][i]=1;
						else
						{
							usaUS_PhysicalSensor[x][y][i]++;
							//printf("%d\n",(int)usaUS_PhysicalSensor[x][y][i]);
						}

					}//else

				}//if
		}//for k
	}//for i
	}
}
```

```
/*********************************************************************************
* Name: UltraSonic_Class::US_SFA_LogicalOR       *
* Description: This function fuse the data between the physical US sensors         *
* based on the OR method *
*********************************************************************************/
void UltraSonic_Class::US_SFA_LogicalOR()
{
	//level 6

	int i, j, k;// temp;
	unsigned short max;

	for(i=0; i<g_USGridSizeX; i++)
	{
		for (j=0; j<g_USGridSizeY; j++)
		{
			max=0;
			for (k=0; k<6; k++)
			{
				//printf("%d",(int)usaUS_PhysicalSensor[i][j][k]);
				if (usaUS_PhysicalSensor[i][j][k]>max &&
usaUS_PhysicalSensor[i][j][k]!=500)
					max=usaUS_PhysicalSensor[i][j][k];

			}//for k
			if (max!=0)
				usaUS_PhysicalSensor[i][j][6]=max;
			else
				usaUS_PhysicalSensor[i][j][6]=0;

		}//j
	}//i
}

/*********************************************************************************
* Name: UltraSonic_Class::US_SFA_ProbabilisticApproach    *
* Description: This function fuse the data between the physical US sensors         *
* based on the algorithm which is based on the paper of Miguel Ribo and            *
* Axel Pinz, 2001,            *
* A comparison of three uncertainty calculi for building sonar based      *
* occupancy grids algorithms, Robotics and Automation systems 35: 201-209     *
*********************************************************************************/
void UltraSonic_Class::US_SFA_ProbabilisticApproach()
{
	//level 7
	//Cell values: Unknown=500; Occupied=1; Empty=0;

//| 0   | 500 | 1 |
//   ===================
//    0  || 0 | 0   | 0 |
//   -----------------
//    500|| 0 | 500 | 1 |
//   -----------------
//    1  || 0 | 1   | 1 |
//   -----------------

	int i, j, k;
	unsigned short Temp1, Temp2;

	for(i=0; i<g_USGridSizeX; i++)
```

```
	{
		for (j=0; j<g_USGridSizeY; j++)
			usaUS_PhysicalSensor[i][j][7]=usaUS_PhysicalSensor[i][j][0];
	}

	//New Probablistic Approach

	for(i=0; i<g_USGridSizeX; i++)
	{
		for (j=0; j<g_USGridSizeY; j++)
			usaUS_PhysicalSensor[i][j][7]=usaUS_PhysicalSensor[i][j][0];
	}

	for (k=1; k<6; k++)
	{
		for (i=0; i<g_USGridSizeX; i++)
		{
			for (j=0; j<g_USGridSizeY; j++)
			{
				if ( (usaUS_PhysicalSensor[i][j][7]==0)||(usaUS_PhysicalSensor[i][j][k]==0))
					usaUS_PhysicalSensor[i][j][7]=0;
				else if

	((usaUS_PhysicalSensor[i][j][7]==500)&&(usaUS_PhysicalSensor[i][j][k]==2))
					usaUS_PhysicalSensor[i][j][7]=3;
				else
				{
					Temp1=usaUS_PhysicalSensor[i][j][7];
					Temp2=usaUS_PhysicalSensor[i][j][k];
					if (Temp1==500 && Temp2!=500)
						usaUS_PhysicalSensor[i][j][7]=Temp2;
					if (Temp1!=500 && Temp2==500)
						usaUS_PhysicalSensor[i][j][7]=Temp1;
					if (Temp1!=500 && Temp2!=500)
						if (Temp1>Temp2)
							usaUS_PhysicalSensor[i][j][7]=Temp1;
						else
							usaUS_PhysicalSensor[i][j][7]=Temp2;

				}
			}//for j
		}//for i
	}//for k

}


/*********************************************************************************
* Name: UltraSonic_Class::US_GridMapCellConversion		*
* Description: This function convert cell size from US to LBM		*
*********************************************************************************/
void UltraSonic_Class::US_GridMapCellConversion()
{
	int i, j, k, m, n;
	int iResParam;

	g_BB.iLBM_X_Old=g_BB.iLBM_X_New;
	g_BB.iLBM_Y_Old=g_BB.iLBM_Y_New;
	g_BB.iLBM_Theta_Old=g_BB.iLBM_Theta_New;
```

```
	g_BB.iLBM_X_New=US.iUS_X;
	g_BB.iLBM_Y_New=US.iUS_Y;
	g_BB.iLBM_Theta_New=US.iUS_Theta;


// replacing cells marked as unknown (2) to empty (0),
// and cells maeked as conflit (3) to occupy (1)

	for (k=0; k<=(g_iTotalNumOfUsLS-1); k++)
	{
		for (i=0; i<g_USGridSizeX; i++)
		{
			for (j=0; j<g_USGridSizeY; j++)
			{
				if (usaUS_PhysicalSensor[i][j][k+6]==500)
					usaUS_PhysicalSensor[i][j][k+6]=0;
				//else if (usaUS_PhysicalSensor[i][j][k+7]==3)
				//	usaUS_PhysicalSensor[i][j][k+7]=1;
			}
		}
	}

// cell conversion procedure

	iResParam= g_USCellSize/g_LBMCellSize;

	for (k=0; k<g_iTotalNumOfUsLS; k++)
	{
		g_BB.bLGM_NewDataFlag[k+1]=1; // turn the flag on
		for (i=0; i<g_USGridSizeX; i++)
		{
			for (j=0; j<g_USGridSizeY; j++)
			{
				for (m=0; m<iResParam; m++)
				{
					for (n=0; n<iResParam; n++)

	g_BB.iaLBM[(i*iResParam+m)][(j*iResParam+n)][k+1]=usaUS_PhysicalSensor[i][j][k+6];
				}
			}
		}
	}

}


/******************************************************************************
* Name: UltraSonic_Class::UltraSonic_Class		*
* Description: Default Constructor.The location of the US physical sensors		*
* relative to the center of mass is defined	*
******************************************************************************/
UltraSonic_Class::UltraSonic_Class()
{
	sonarNum[0] = 1;
	sonarNum[1] = 2;
   sonarNum[2] = 3;
	sonarNum[3] = 4;
	sonarNum[4] = 5;
	sonarNum[5] = 6;
	faUS_SonarLoc[0][0]=-54.5;// X Location
```

```
	faUS_SonarLoc[1][0]=-51;// X Location
	faUS_SonarLoc[2][0]=-49;// X Location
	faUS_SonarLoc[3][0]=-49;// X Location
	faUS_SonarLoc[4][0]=-51;// X Location
	faUS_SonarLoc[5][0]=-54.5;// X Location

	faUS_SonarLoc[0][1]=11.5;// Y Location
	faUS_SonarLoc[1][1]=8.0;// Y Location
	faUS_SonarLoc[2][1]=2.5;// Y Location
	faUS_SonarLoc[3][1]=-2.5;// Y Location
	faUS_SonarLoc[4][1]=-8.0;// Y Location
	faUS_SonarLoc[5][1]=-11.5;// Y Location

	faUS_SonarLoc[0][2]=50;// Theta Location
	faUS_SonarLoc[1][2]=30;// Theta Location
	faUS_SonarLoc[2][2]=10;// Theta Location
	faUS_SonarLoc[3][2]=-10;// Theta Location
	faUS_SonarLoc[4][2]=-30;// Theta Location
	faUS_SonarLoc[5][2]=-50;// Theta Location
}

/*******************************************************************************
* Name: UltraSonic_Class::~UltraSonic_Class		*
* Description: Default Destructor	*
*******************************************************************************/
UltraSonic_Class::~UltraSonic_Class()
{
	;
}
```

```
/**
** FuzzyLogic_Algorithm.h
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**/

#ifndef __FuzzyLogic_Algorithm_h__
#define __FuzzyLogic_Algorithm_h__

#include <windows.h>
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"

// Constant parameters
const double	cf_TT[]		={-0.0001,0,0.3,0.45,  0.4,0.45,0.55,0.6,  0.55,0.7,1,1.0001};
const double	cf_FF[]		={-0.0001,0,0.3,0.45,  0.4,0.45,0.55,0.6,  0.55,0.7,1,1.0001};
const double	cf_TF[]		={-0.0001,0,0.3,0.45,  0.4,0.45,0.55,0.6,  0.55,0.7,1,1.0001};
const double	cf_FT[]		={-0.0001,0,0.3,0.45,  0.4,0.45,0.55,0.6,  0.55,0.7,1,1.0001};
const double	cf_TRUE[]	={-0.0001,0,0.3,0.45,  0.4,0.45,0.55,0.6,  0.55,0.7,1,1.0001};
const double	cf_FALSE[]	={-0.0001,0,0.3,0.45,  0.4,0.45,0.55,0.6,  0.55,0.7,1,1.0001};

class FuzzyLogic
{
private:
	const double *Data;  // Const Data which contain 12 parametrs for each one of the three
			     // Trapezoids "Low","Avarage","High"
	char* FuzzyName; // The name of the traoezoid we want to reafer to, can be
			 // one of :"Low","Avarage","High"
	float CrispValue; //The crisp value we get from the programe
	float FuzzyValue; //The fuzzy value we calculate by the 'FL_Crisp2Fuzzy' function
	float CenterOfMassCrisp; //The COM of the Crisp value which is calculated by the
				 // operator '>>'
	float CenterOfMassFuzzy;
	float Area;

public:
	FuzzyLogic ();
	FuzzyLogic &FLInsCrispVal(float);

	friend FuzzyLogic operator>>(/*const*/ FuzzyLogic&, FuzzyLogic&);
	friend FuzzyLogic operator+(const FuzzyLogic&,const FuzzyLogic&);
	friend FuzzyLogic operator*(const FuzzyLogic&,const FuzzyLogic&);
	FuzzyLogic FL_Crisp2Fuzzy(char*);
	FuzzyLogic &FLInsFuzzyName(char*);
	FuzzyLogic (const double *);
	float FuzzyLogicGetCrispValue();
	float FuzzyLogicGetFuzzyValue();
	float FuzzyLogicGetCenterOfMassCrisp();
	float FuzzyLogicGetAraeValue();

	~FuzzyLogic ();
};
#endif
```

```
/**
** FuzzyLogic_Algorithm.cpp
**
** Copyright 2001 by Ofir Cohen
**
** E-mail:  oprc@bgumail.bgu.ac.il
**/

#include <windows.h>
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"
#include "InitiationFile.h"
#include "FuzzyLogic_Algorithm.h"



/******************************************************************************
* Name: FuzzyLogic::FuzzyLogic ()            *
* Description: Default Constructor with no data         *
******************************************************************************/
FuzzyLogic::FuzzyLogic ()
{
	CrispValue=0;   // The Crisp Value default function value
	Area=0;         // The Area default function value
	FuzzyValue=0;   //The Fuzzy default function value
	CenterOfMassCrisp=0; // The Center Of Mass default Crisp value
	CenterOfMassFuzzy=0; // The Center Of Mass default Fuzzy value
}

/******************************************************************************
* Name: FuzzyLogic::FuzzyLogic (const double *Data1)         *
* Description: Default Constructor for with constant DATA (12 parameters which represent the     *
* 3 trapezoids "Low","Avarage","High", )    *
******************************************************************************/
FuzzyLogic::FuzzyLogic (const double *Data1)
{
	Data=Data1;
	CrispValue=0;   // The Crisp Value default function value
	Area=0;         // The Area default function value
	FuzzyValue=0;   //The Fuzzy default function value
	CenterOfMassCrisp=0; // The Center Of Mass default Crisp value
	CenterOfMassFuzzy=0; // The Center Of Mass default Fuzzy value
}


/******************************************************************************
* Name: FuzzyLogic FuzzyLogic::FL_Crisp2Fuzzy (char *FuzzyName)            *
* Description: This function: FL_Crisp2Fuzzy calculate the FUZZY value for each crispy value     *
******************************************************************************/
FuzzyLogic FuzzyLogic::FL_Crisp2Fuzzy (char *FuzzyName)
{

	this->FuzzyName=FuzzyName;
	this->FuzzyValue=0;

	int result,i;
	float a,b,DegreeOfMembership=0.;
	int FlagChack=0;
```

```
            result = strspn(FuzzyName,"Low");
   if (result==3)
              i=0;
            result = strspn(FuzzyName,"Avarage");
   if (result==7)
              i=4;
            result = strspn(FuzzyName,"High");
   if (result==4)
              i=8;
                        if ((this->CrispValue>=this->Data[i]) && (this->CrispValue<=this->Data[i+3]))
                        {
                                    if ((this->CrispValue>=this->Data[i]) && (this->CrispValue<=this->Data[i+1]))
                                    {
                                                a=((float)1./(this->Data[i+1]-this->Data[i]));
                                                b=((float)(-1.)*a*this->Data[i]);
                                                DegreeOfMembership=((float)a*this->CrispValue+b);
                                                if ((this->FuzzyValue<DegreeOfMembership))
                                                            this->FuzzyValue=DegreeOfMembership;
                                                FlagChack=1;
                                    }

                                    if((this->CrispValue>=this->Data[i+2]) && (this->CrispValue<=this->Data[i+3]))
                                    {
                                                a=((float)(-1.)/(this->Data[i+3]-this->Data[i+2]));
                                                b=((float)(-1.)*a*this->Data[i+3]);
                                                DegreeOfMembership=((float)a*this->CrispValue+b);
                                                if (this->FuzzyValue<(float)DegreeOfMembership)
                                                            this->FuzzyValue=(float)DegreeOfMembership;
                                                FlagChack=1;
                                    }
                                    if (FlagChack==0)
                                    {
                                                DegreeOfMembership=1.;
                                                if (this->FuzzyValue<(float)DegreeOfMembership)
                                                            this->FuzzyValue=(float)DegreeOfMembership;
                                    }
                        }
            return FuzzyLogic(*this);
}

/*****************************************************************************
* Name: FuzzyLogic operator>>(FuzzyLogic &FL_Source1, FuzzyLogic &FL_Target1)  *
* Description: This Operator: >> Means 'Then' at the IF....THEN fuzzy rules       *
*****************************************************************************/
FuzzyLogic operator>>(/*const*/ FuzzyLogic &FL_Source1,/*const*/ FuzzyLogic &FL_Target1)
{
            //Beacuse the Data parameters for each object is CONST we need to 'copy' the
            // object and then work on the new objects
            FuzzyLogic FL_Source;
            FuzzyLogic FL_Target;
            FL_Source=FL_Source1;
            FL_Target=FL_Target1;
            FL_Target.FuzzyValue=FL_Source1.FuzzyValue; //we need to get the new fuzzy value
                                                                                   //after making OR or AND operations
            int result,i;
            float a,b ; //the parametrs of the linear equation
            float StamArray[4];
            result = strspn(FL_Target.FuzzyName,"Low");
   if (result==3)
```

```
             i=0;
           result = strspn(FL_Target.FuzzyName,"Avarage");
  if (result==7)
             i=4;
           result = strspn(FL_Target.FuzzyName,"High");
  if (result==4)
             i=8;

if(FL_Target.FuzzyValue>0)
  {
  a=1/(FL_Target.Data[i+1]-FL_Target.Data[i]);
  b=(-1)*a*FL_Target.Data[i];
  StamArray[1]=(FL_Target.FuzzyValue-b)/a;

  a=(-1)/(FL_Target.Data[i+3]-FL_Target.Data[i+2]);
  b=(-1)*a*FL_Target.Data[i+3];
  StamArray[2]=(FL_Target.FuzzyValue-b)/a;

  StamArray[0]=FL_Target.Data[i];
  StamArray[3]=FL_Target.Data[i+3];
  FL_Target.Area=0.5*(FL_Target.FuzzyValue)*
             (StamArray[3]+StamArray[2]-StamArray[1]-StamArray[0]);
 FL_Target.CenterOfMassCrisp=
             (0.5*(StamArray[2]+StamArray[1])*(StamArray[2]-StamArray[1])*FL_Target.FuzzyValue+
             0.5*((2./3.)*StamArray[2]+(1./3.)*StamArray[3])*(StamArray[3]-
StamArray[2])*FL_Target.FuzzyValue+
             0.5*((2./3.)*StamArray[1]+(1./3.)*StamArray[0])*(StamArray[1]-
StamArray[0])*FL_Target.FuzzyValue)/
             FL_Target.Area;
  }
  else
  {
             FL_Target.CenterOfMassCrisp=0;
             FL_Target.Area=0;
  }
           FL_Target1.FuzzyValue=FL_Target.FuzzyValue;
           FL_Target1.CenterOfMassCrisp=FL_Target.CenterOfMassCrisp;
           FL_Target1.Area=FL_Target.Area;
  //FL_Target1=FL_Target;
           return FuzzyLogic(FL_Target1);
}

/*********************************************************************************
* Name: FuzzyLogic operator+(const FuzzyLogic &FL1,const FuzzyLogic &FL2)          *
* Description: This Operator: + Means 'OR' at the IF....THEN fuzzy rules          *
*********************************************************************************/
FuzzyLogic operator+(const FuzzyLogic &FL1,const FuzzyLogic &FL2){
           FuzzyLogic FL_Stam;
           if (FL1.FuzzyValue > FL2.FuzzyValue)
           {
                      //FL_Stam.FuzzyValue=FL1.FuzzyValue;
                      FL_Stam=FL1;
           }
           else
           {
                      //FL_Stam.FuzzyValue=FL2.FuzzyValue;
                      FL_Stam=FL2;
           }
           return FuzzyLogic(FL_Stam);}
```

```
/*******************************************************************************
* Name: FuzzyLogic operator*(const FuzzyLogic &FL1,const FuzzyLogic &FL2)          *
* Description: This Operator: * Means 'AND' at the IF....THEN fuzzy rules        *
*******************************************************************************/
FuzzyLogic operator*(const FuzzyLogic &FL1,const FuzzyLogic &FL2){
	FuzzyLogic FL_Stam;
	if (FL1.FuzzyValue < FL2.FuzzyValue)
	{
		FL_Stam=FL1;
	}
	else
	{
		FL_Stam=FL2;
	}
	return FuzzyLogic(FL_Stam);}

/*******************************************************************************
* Name: FuzzyLogic::FuzzyLogicGetCenterOfMassCrisp()     *
* Description: This function: FuzzyLogicGetCenterOfMassCrisp returns the crisp value of the       *
* COM after running the rules        *
*******************************************************************************/
float FuzzyLogic::FuzzyLogicGetCenterOfMassCrisp(){
	return (this->CenterOfMassCrisp);}

/*******************************************************************************
* Name: FuzzyLogic::FuzzyLogicGetFuzzyValue()    *
* Description: This function: FuzzyLogicGetFuzzyValue prints out the fuzzy value of the object    *
*******************************************************************************/
float FuzzyLogic::FuzzyLogicGetFuzzyValue(){
	return (this->FuzzyValue);}

/*******************************************************************************
* Name: FuzzyLogic::FuzzyLogicGetAraeValue()      *
* Description: This function: FuzzyLogicGetAraeValue return the area of the object        *
*******************************************************************************/
float FuzzyLogic::FuzzyLogicGetAraeValue(){
	return (this->Area);}


/*******************************************************************************
* Name: FuzzyLogic::FuzzyLogicGetCrispValue()     *
* Description: This function: FuzzyLogicGetCrispValue prints out the crisp value of the object      *
*******************************************************************************/
float FuzzyLogic::FuzzyLogicGetCrispValue(){
	return (this->CrispValue);}

/*******************************************************************************
* Name: &FuzzyLogic::FLInsFuzzyName(char* FuzzyName) *
* Description: This function: FLInsFuzzyName enters new Fuzzy name for the object      *
*******************************************************************************/
FuzzyLogic &FuzzyLogic::FLInsFuzzyName(char* FuzzyName){
	this->FuzzyName=FuzzyName;
	return (*this);}
```

```
/*********************************************************************************
* Name: &FuzzyLogic::FLInsCrispVal(float CValue)*
* Description: This function: FLInsFuzzyName enters new Crisp value for the object        *
*********************************************************************************/
FuzzyLogic &FuzzyLogic::FLInsCrispVal(float CValue){
        this->CrispValue=CValue;
        return (*this);
}

/*********************************************************************************
* Name: FuzzyLogic::~FuzzyLogic()           *
* Description: Default Destructor with no data           *
*********************************************************************************/
FuzzyLogic::~FuzzyLogic() {
        PostQuitMessage(0);}
```

```
/**
**Sick_Class.h
**
** Copyright 2007 by Keren Kapach
**
** E-mail:  kapach@bgu.ac.il
**/
#include <time.h>
#include <conio.h>
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdio.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"

#include "Aria.h"
extern BlackBoard g_BB;

class Sick_Class
{
		public:
				int iSick_X; //robot's Y location
				int iSick_Y; //robot's X location
				int iSick_Theta;
				int Sick_PhysicalMap[g_iX_LBM_MapSize][g_iY_LBM_MapSize][g_iTotalNumOfSiLS];
				int range[360];

				 Sick_Class(); //default constructor
				 void ReadFromSick();
	void Si_GridMapCellConversion();
};
```

```
/**
**Sick_Class.cpp
**
** Copyright 2007 by Keren Kapach
**
** E-mail:  kapach@bgu.ac.il
**/
#include "Sick_Class.h"
#include <math.h>
#include "ConstantParameters.h"
#include "GlobalParameters.h"

extern  ArRobot robot ;
extern  ArSick *sick;
extern Sick_Class mySick;

/*******************************************************************************
* Name: Sick_Class::Sick_Class     *
* Description: Default constructor   *
*******************************************************************************/
Sick_Class::Sick_Class()
{
	int i,j,k;
	for (k=0;k<=g_iTotalNumOfSiLS;k++)
	{
		for (i=0; i<g_SickGridSizeX; i++)
		{
			for (j=0; j<g_SickGridSizeY ;j++)
				Sick_PhysicalMap[i][j][k]=0;
		}//for i
	}//for k


}
/*******************************************************************************
* Name: Sick_Class::ReadFromSick            *
* Description: This function reads the data from the laser sensor and generates two logical sensors from * this data .
*******************************************************************************/
void Sick_Class::ReadFromSick()
{

	double phi,tempPhi,xTag,yTag,x0,y0;
	int x,y,flag=0;
	int i=0;
	int ObsLocInMapX, ObsLocInMapY,Theta,k;

	static int sum=0;

	iSick_X=(int)(robot.getX()*0.1);
	iSick_Y=(int)(robot.getY()*0.231);
	iSick_Theta=(int)(robot.getTh());

	g_BB.iPPGM_X=iSick_X;
	g_BB.iPPGM_Y=iSick_Y;
	g_BB.iPPGM_Theta=iSick_Theta;

	const std::list<ArSensorReading *> *readings;
   std::list<ArSensorReading *>::const_iterator it;
```

```
        sick->lockDevice();


//Map building for the first laser LS
        readings = sick->getRawReadings();
        if (readings != NULL)
   {
          for (it = readings->begin() , i=0; it != readings->end(); it++,i++)
          {
         char tmp[100];
                range[i] = (*it)->getRange();

                range[i]=range[i]*0.1; //converting to cm
                sprintf(tmp,"Angle %d  reading %d \n ",i, range[i]);
                output<<tmp;

                phi=(g_pi/180*(i-90)); //transferring to the Sonar's angle
                x0=range[i]*cos(phi)-60;
                y0=range[i]*sin(phi);
                ObsLocInMapX=(int)(x0);
                ObsLocInMapY=(int)(y0+LBM_cm_SizeY/2);

                for (k=0;k<5;k=k+2)
                {
                        for(Theta=0; Theta<360; Theta=Theta+30)
                        {
                                tempPhi=g_pi/180*Theta;
                                xTag=(double)k*cos(tempPhi);
                                yTag=(double)k*sin(tempPhi);
                                x0=xTag+(double)ObsLocInMapX;
                                y0=yTag+(double)ObsLocInMapY;
                                x=(int)(x0/(double)g_CamCellSize);
                                y=(int)(y0/(double)g_CamCellSize);

                if (x>=0 && x<g_SickGridSizeX && y>=0 && y<g_SickGridSizeY)
                        Sick_PhysicalMap[x][y][0]++; //finding how many times cell is samples
                         //Sick_PhysicalMap[x][y][0]=1;
               //       Sick_PhysicalMap[x][y][0]=0;  //LASER1=Empty
                         }//for theta
                }//for k
          }//for it
        }//if


//Map building for the second laser LS
        for(i=0; i<181; i=i+3)
        {
               phi=(g_pi/180*(i-90));
               x0=range[i]*cos(phi)-60;
               y0=range[i]*sin(phi);
               ObsLocInMapX=(int)(x0);
               ObsLocInMapY=(int)(y0+LBM_cm_SizeY/2);

               for (k=0;k<5;k=k+2)
               {
                       for(Theta=0; Theta<360; Theta=Theta+30)
                        {
                                tempPhi=g_pi/180*Theta;
                                xTag=(double)k*cos(tempPhi);
                                yTag=(double)k*sin(tempPhi);
```

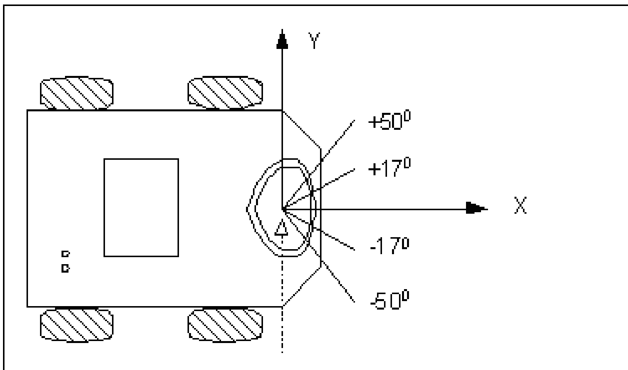
```
                        x0=xTag+(double)ObsLocInMapX;
                        y0=yTag+(double)ObsLocInMapY;
                        x=(int)(x0/(double)g_CamCellSize);
                        y=(int)(y0/(double)g_CamCellSize);

                if (x>=0 && x<g_SickGridSizeX && y>=0 && y<g_SickGridSizeY)
                    Sick_PhysicalMap[x][y][1]++;
                     }//for theta
                }//for k
        }//for i


        sum++;
        output.close();
        sick->unlockDevice();
}

/**********************************************************************************
* Name: Sick_Class:: Si_GridMapCellConversion     *
* Description: This function converts cell size from laser to LBM.
**********************************************************************************/
void Sick_Class::Si_GridMapCellConversion()
{

        int i, j, k,m,n, iResParam;

        g_BB.iLBM_X_Old=g_BB.iLBM_X_New;
        g_BB.iLBM_Y_Old=g_BB.iLBM_Y_New;

        g_BB.iLBM_X_New=mySick.iSick_X;
        g_BB.iLBM_Y_New=mySick.iSick_Y;

        iResParam=g_SickCellSize/g_LBMCellSize;

        for(k=0; k<g_iTotalNumOfSiLS; k++)
        {
                g_BB.bLGM_NewDataFlag[k+1+g_iTotalNumOfUsLS]=1;
                for(i=0; i<g_SickGridSizeX; i++)
                {
                        for (j=0; j<g_SickGridSizeY; j++)
                        {
                                for (m=0; m<iResParam; m++)
                                        for (n=0; n<iResParam; n++)

        g_BB.iaLBM[(i*iResParam+m)][(j*iResParam+n)][k+1+g_iTotalNumOfUsLS]=Sick_PhysicalMap[i][j][k]
;
                        }//j
                }//i
        }//k

//initialazing the physical maps
        for(k=0; k<g_iTotalNumOfSiLS; k++)
        {
                for(i=0; i<g_SickGridSizeX; i++)
                        for (j=0; j<g_SickGridSizeY; j++)
                                Sick_PhysicalMap[i][j][k]=0;
        }//for
}
```

Appendix V Camera calibration

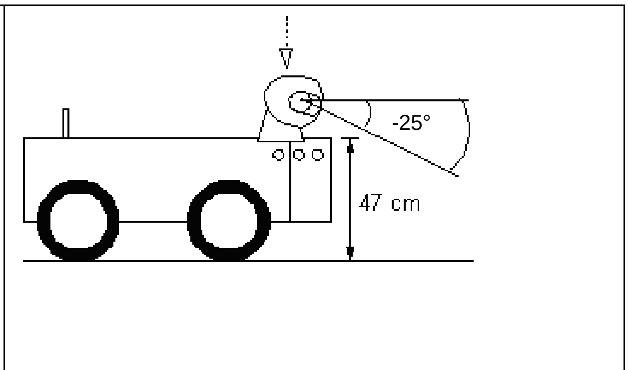
General

The main goal of this work is to map the robot's surrounding using different physical sensors. One of these sensors is a PTZ CCD camera, mounted on top of the robot, as detailed in chapter 6. The robot’s surrounding is represented by the grid map paradigm. The obstacles within the grid map are placed in the real world X-Y coordinates. To create grid maps from photos taken from the camera, we need to find the mathematical relation between the obstacle’s pixels coordinates and the real world X-Y obstacle’s coordinates. This is done through a calibration process.

In the mobile robot experiment, the camera is set to a specific tilt angle (-25°) and takes photos from four different pan angles (-50°, -17°, 17° and 50°) as shown in Figure 17. To avoid a different calibration for each pan angle, the distance was measured only from the rotation axis, as detailed later.



Pan angles



Tilt angle

Figure 17 Camera angles: pan and tilt

(Adapted from Cohen, 2005)

The basic assumption is that the calibration was employed for a specific camera tilt angle (-25°), zoom and height from the floor (47 cm), as shown in Figure 17. To determine the obstacle’s location relative to the robot, two functions were developed, one along the robot’s X- axis of movement, the second on the Y-axis. The obstacle's location relative to the zero point was derived from the geometrical relation.

In the calibration process, small obstacles were placed on the floor at known distances, photos were taken, and equations that describe the relation between the pixel coordinates and the real X-Y coordinates were determined.

An experiment was conducted to check the calibration parameters. In the experiment pointed obstacles were pointed at a known location relative to the zero point. The obstacles’ position was found according to the parameters that were derived, and the mean error between the real location and the calculated location was found.

Methodology

In the calibration process the following steps were taken:

1. Determining the rotation axis
2. Deriving the mathematical equation for X axis relative to the robot
3. Deriving the mathematical equation for Y axis relative to the robot
4. Finding the obstacle's location relative to the zero point

1. *Determining the rotation axis*

To avoid a different calibration for each pan angle, we assume that the points on the rotation axis do not move. All the distances were measured relatively to the rotation axis. The rotation axis was found from the camera in an experiment. The point on this axis remains static and does not move while the camera turns to the different pan angles.

The rotation axis was found in the following experimental procedure:

1. Set the camera tilt angle to -25°.
2. Estimate the rotation axis and gently touch this point, using a pencil. Since on the rotation axis the pencil remains static and does not move while the camera turns to the different pan angles, converge to this point by 'trial – and – error'. When at the rotation axis, the pencil creates a point; when away from the rotation axis, the pencil will create an arc.

2. *Deriving the mathematical equation for X axis relative to the robot*

A set of pointed obstacles was placed at known distances from the rotation axis, in different X and Y distances, as shown in Figure 18.

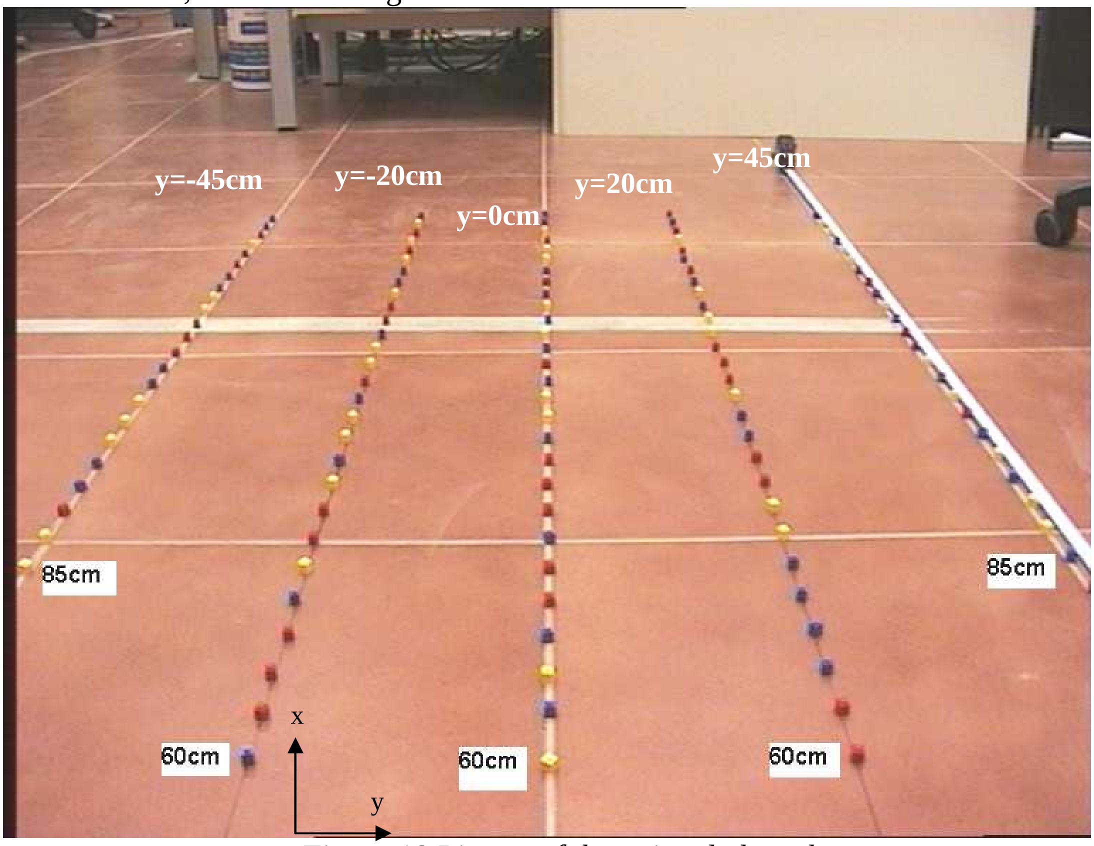


Figure 18 Picture of the pointed obstacles

Along the X axis, the obstacles are placed 5cm apart. The Y coordinate of each line is shown in the figure. All the lines were measured from the rotation axis, where y=0 is the rotation axis itself. Along the rotation axis (y=0), the real distances in centimeters of the pointed obstacles vs. their pixel location (as taken from the image using a PaintBrush application), shown in Table 39.

Table 39 Raw data to derive the polynomial equation

X measured [cm]	X [pixels]	X measured [cm]	X [pixels]
60	521	95	347
65	484	100	329
70	460	105	312
75	435	110	297
80	409	115	282
85	387	120	269
90	368	125	257

The polynomial equation was derived in MATLAB using *polyfit* function, which finds the coefficients of a polynomial of degree n that fits the data. The function receives a vector containing values and the desired polynomial degree, and returns a row vector of length n+1 containing the polynomial coefficients in descending powers

The equation is presented in [30]. Where x is the distance in pixels, and $X_{Distance}$ is the distance relative to the robot.

$$X_{Distance} = -6.71E-12x^5 + 1.59E-08x^4 - 1.54E-05x^3 + 0.007846x^2 - 2.3348x + 406.14 \quad [30]$$

3. *Deriving the mathematical equation for Y axis relative to the robot*

Calculation of the distance of the obstacle in the Y axis is based on the concept that the vertical and horizontal lines connect at point P, as shown in Figure 19. The bold line is the rotation axis, and each line in the figure represents the same vertical distance from the rotation axis.

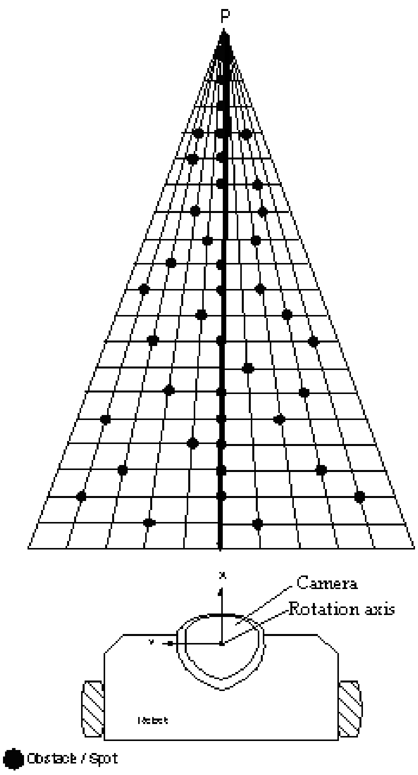


Figure 19 Camera horizontal and vertical lines

The first step is to find the point P coordinates (X_p , Y_p). This was achieved by an intersection of two lines of data taken from the pointed obstacles shown in Figure 18.
X_P coordinate is the same as the rotation axis and was measured as X_p*=385*. This was measured from the X coordinate of the line at the robot's rotation axis (y=0 in Figure 18). The coordinate was found using PaintBrush software.
Y_p coordinate was calculated as follows:
The Pixels location of the pointed obstacles in lines *y=-45cm*, *y=-20cm, y=20cm* and *y=45cm* were taken manually using PaintBrush software, as presented in Table 40.
A linear equation for each line was derived using linear regression. The equations are presented in Table 40. Y_p is the value of the linear line at X_p. Since slightly different values were derived, Y_p was taken as the average value.
Table 2 represents the (x, y) pixel values of the pointed obstacles, the linear equation and the calculated Y value for each line in Figure 18, as derived from the experiment.

Table 40 Obstacle's pixels location

y=-20cm		**y=-45cm**		**y=20cm**		**y=45cm**	
X[pix]	*Y[pix]*	*X[pix]*	*Y[pix]*	*X[pix]*	*Y[pix]*	*X[pix]*	*Y[pix]*
172	518			602	516		
183	486			592	485		
188	460			580	457		
201	432			572	431		
205	408			563	408		
213	386	19	389	557	385	752	378
219	367	29	365	550	363	736	360
226	347	42	348	543	345	725	341
233	328	54	332	537	328	713	325
237	313	67	321	531	311	703	309
241	296	76	299	526	297	691	294
246	283	85	286	521	283	680	280
251	270	95	272	516	268	671	269
256	258	104	261	512	257	662	256
259	246	112	249	508	246	651	245
262	235	121	239	502	235	643	235
267	226	129	228	499	225	637	225
270	216	136	219	497	214	630	215
273	206	143	208	494	206	622	207
276	197	148	200	490	196	616	199
279	190	152	194	487	190	610	190
282	183	159	185	485	182	603	182
284	175	164	178	480	174	597	173
286	168	170	170	479	168	594	169
293	164	175	164	476	160	589	163
290	156	182	157	475	155	583	155
292	149	185	151	472	149	579	150
294	144	189	146	470	143	574	144
y = -3.0127x + 1029.6		*y = -1.3985x + 408.14*		*y = 2.8253x - 1187*		*y = 1.310x - 609.34*	
y(x=385)=-130		*y(x=385)=-130.28*		*y(x=385)=-99.3*		*y(x=385)=-104.72*	
$Y_{Pavg.}$*=-117*							

P coordinates are: **(385,-117)**

For each pointed obstacle, we can calculate *a*, *b* values as shown in Figure 20. ‘a’ represents the vertical distance to point P, and ‘b’ is the horizontal distance.

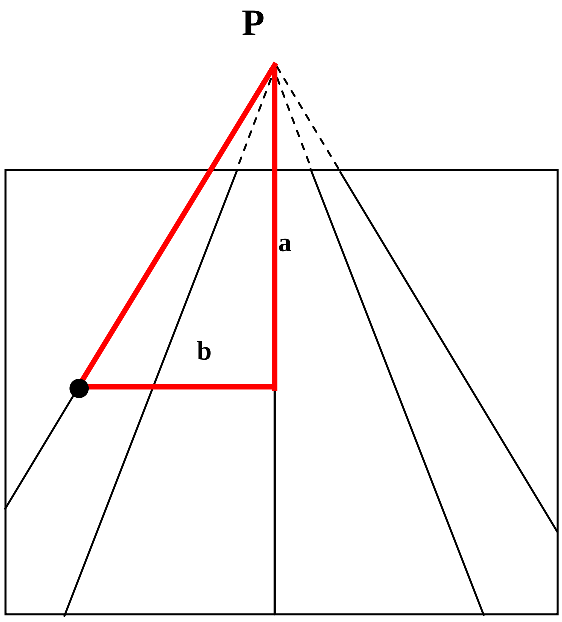


Figure 20 a and b values for each obstacle

Note that the value b/a is the tangent value of the head angle.

Calculation of the *a*, *b* values for each obstacle is presented in equations [31] and [32].

$$a = Y_{pixel} - X_p = Y_{pixel} + 117 \qquad [31]$$

$$b = X_{pixel} - Y_p = X_{pixel} - 385 \qquad [32]$$

The next step is to derive the mathematical relationship between each obstacle's b/a value, and the real Y distance. This was done in the following way:

For each real Y distance (*y=-45cm, y=-20cm, y=20cm and y=-45cm*) the value b/a was calculated. Table 41 represents values for each line as drawn from Figure 18. Since b/a is the tangent of the head angle, values of all the obstacles within the same lines were the same, as expected.

Table 41 Values for each line's obstacle

	y=-20cm					**y=-45cm**				
Y[cm]	*X[pix]*	*Y[pix]*	*a*	*b*	***b/a***	*X[pix]*	*Y[pix]*	*a*	*b*	***b/a***
105	237	313	430	-148	-0.34	67	321	438	-318	-0.73
110	241	296	413	-144	-0.35	76	299	416	-309	-0.74
115	246	283	400	-139	-0.35	85	286	403	-300	-0.74
120	251	270	387	-134	-0.35	95	272	389	-290	-0.75
125	256	258	375	-129	-0.34	104	261	378	-281	-0.74
130	259	246	363	-126	-0.35	112	249	366	-273	-0.75
135	262	235	352	-123	-0.35	121	239	356	-264	-0.74
140	267	226	343	-118	-0.34	129	228	345	-256	-0.74
145	270	216	333	-115	-0.35	136	219	336	-249	-0.74
150	273	206	323	-112	-0.35	143	208	325	-242	-0.74
155	276	197	314	-109	-0.35	148	200	317	-237	-0.75
160	279	190	307	-106	-0.35	152	194	311	-233	-0.75
165	282	183	300	-103	-0.34	159	185	302	-226	-0.75
170	284	175	292	-101	-0.35	164	178	295	-221	-0.75
175	286	168	285	-99	-0.35	170	170	287	-215	-0.75
180	293	164	281	-92	-0.33	175	164	281	-210	-0.75
185	290	156	273	-95	-0.35	182	157	274	-203	-0.74
190	292	149	266	-93	-0.35	185	151	268	-200	-0.75
195	294	144	261	-91	-0.35	189	146	263	-196	-0.75

Table 38 (continued)

	y=20cm	**y=45cm**								
Y[cm]	*X[pix]*	*1. Y[pix]*	*a*	*b*	***b/a***	*X[pix]*	*Y[pix]*	*a*	*b*	***b/a***
105	531	311	428	146	0.34	703	309	426	318	0.75
110	526	297	414	141	0.34	691	294	411	306	0.74
115	521	283	400	136	0.34	680	280	397	295	0.74
120	516	268	385	131	0.34	671	269	386	286	0.74
125	512	257	374	127	0.34	662	256	373	277	0.74
130	508	246	363	123	0.34	651	245	362	266	0.73
135	502	235	352	117	0.33	643	235	352	258	0.73
140	499	225	342	114	0.33	637	225	342	252	0.74
145	497	214	331	112	0.34	630	215	332	245	0.74
150	494	206	323	109	0.34	622	207	324	237	0.73
155	490	196	313	105	0.34	616	199	316	231	0.73
160	487	190	307	102	0.33	610	190	307	225	0.73
165	485	182	299	100	0.33	603	182	299	218	0.73
170	480	174	291	95	0.33	597	173	290	212	0.73
175	479	168	285	94	0.33	594	169	286	209	0.73
180	476	160	277	91	0.33	589	163	280	204	0.73
185	475	155	272	90	0.33	583	155	272	198	0.73
190	472	149	266	87	0.33	579	150	267	194	0.73
195	470	143	260	85	0.33	574	144	261	189	0.72

From Table 41 we can find the mathematical relationship between each obstacle b/a value and the real Y distance in centimeters. The raw data is presented in Table 42.

Table 42 Raw data to derive the mathematical relationship between Y[cm] and b/a

Y[cm]	***b/a***
0	0
-20	-0.34
-45	-0.75
20	0.34
45	0.73

Using linear regression, the equation is presented in [33].

$$\mathrm{Y}_{\mathrm{Distance}} = 60.459 \cdot b/a + 0.2418 \qquad [33]$$

The distance between the center of the camera and the obstacle is presented in [34].

$$R = \sqrt{X_{\mathrm{Distance}}^{2} + Y_{\mathrm{Distance}}^{2}} \qquad [34]$$

The angle between the center of the camera and the obstacle is presented in [35].

$$\alpha = a \cdot \tan\left(\frac{X_{\mathrm{Distance}}}{Y_{\mathrm{Distance}}}\right) \qquad [35]$$

4. *Finding the obstacle's location relative to the zero point*

Once we know the mathematical relationship between the pixel coordinates and the X-Y coordinates relative to the robot, the robot's location and the camera's pan angle, the X-Y coordinates relative to the zero point can be derived.

Figure 21 shows the movement axis, starting at point O. OC is the robot's distance from the starting point (taken from the robot's encoders). OB and AB is the vertical distance from point O and the horizontal distance from the movement axis, in correspondence, and θ is the given camera's tilt angle.

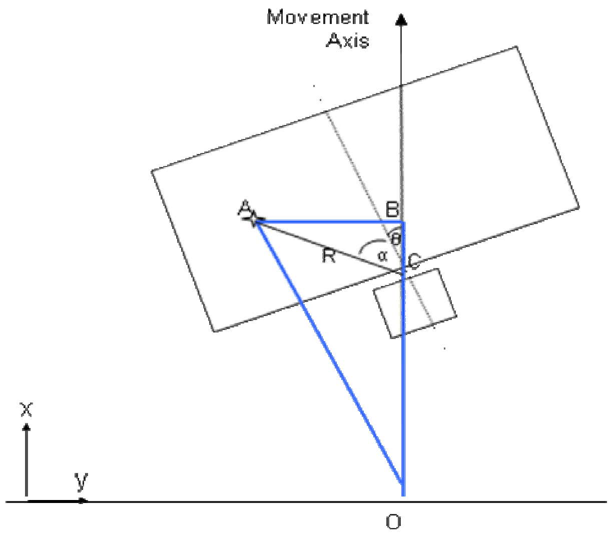


Figure 21 The robot’s distance relative to the starting point

From the geometrical relationships in Figure 21, where point A represents the obstacle's location, we can see that the Real X distance relative to the starting point is BO, and the real Y distance is AB. BO is the sum of BC and CO, where CO is the robot's distance from the zero point, and can be found from the robot's encoders. θ is given, α and R are calculated from [34] and [35].
The real X and Y distances are derived from [36] and [37] as follows:

$$Real\ X = BO = BC + CO = R\cos\left(\alpha+\theta\right) \quad [36]$$

$$Real\ Y = AB = R\sin\left(\alpha+\theta\right) \quad [37]$$

Experiment
An experiment was performed to test the calibration result.
An array of seven pointed obstacles (with different colors) was pointed in known X and Y distances, as shown in Table 43.

Table 43 Obstacle's array location for the expirement

#Obs. Num.	Color	Vertical distance from the driving path [cm]	Horizontal distance [cm]
1	Orange	45	-45
2	White	90	-45
3	Black	135	-90
4	Red	45	45
5	Black	135	45
6	Purple	90	90
7	Yellow	180	90

The robot moves a distance of 1.5m in a straight line, at a constant velocity (0.5 m/s) and takes images at 5 different angles: -40°, -20°, 0°, 20° and 40°, in a continuous loop.
Overall, 40 images were taken. For every picture, the pan angle, the robots X location (OC) and Y location (Zero to all the images, since the robot moves in a straight line) are known. The images were analyzed to find the obstacle's location.
Examples of the obstacle's images in the different pan angles are presented in Figure 22.

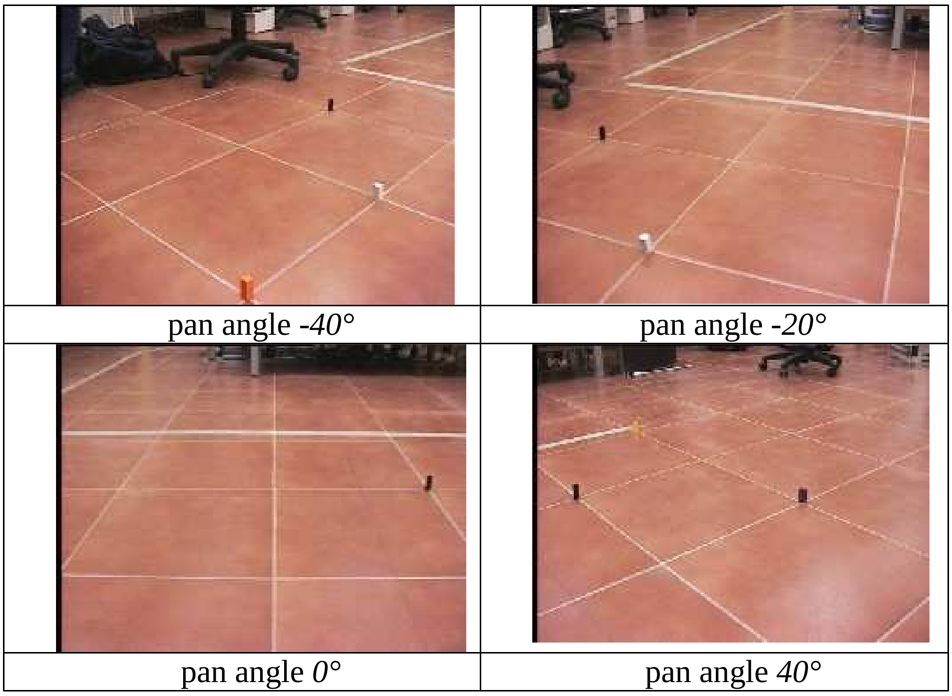


Figure 22 Examples of the obstacle's photos in the different pan angles

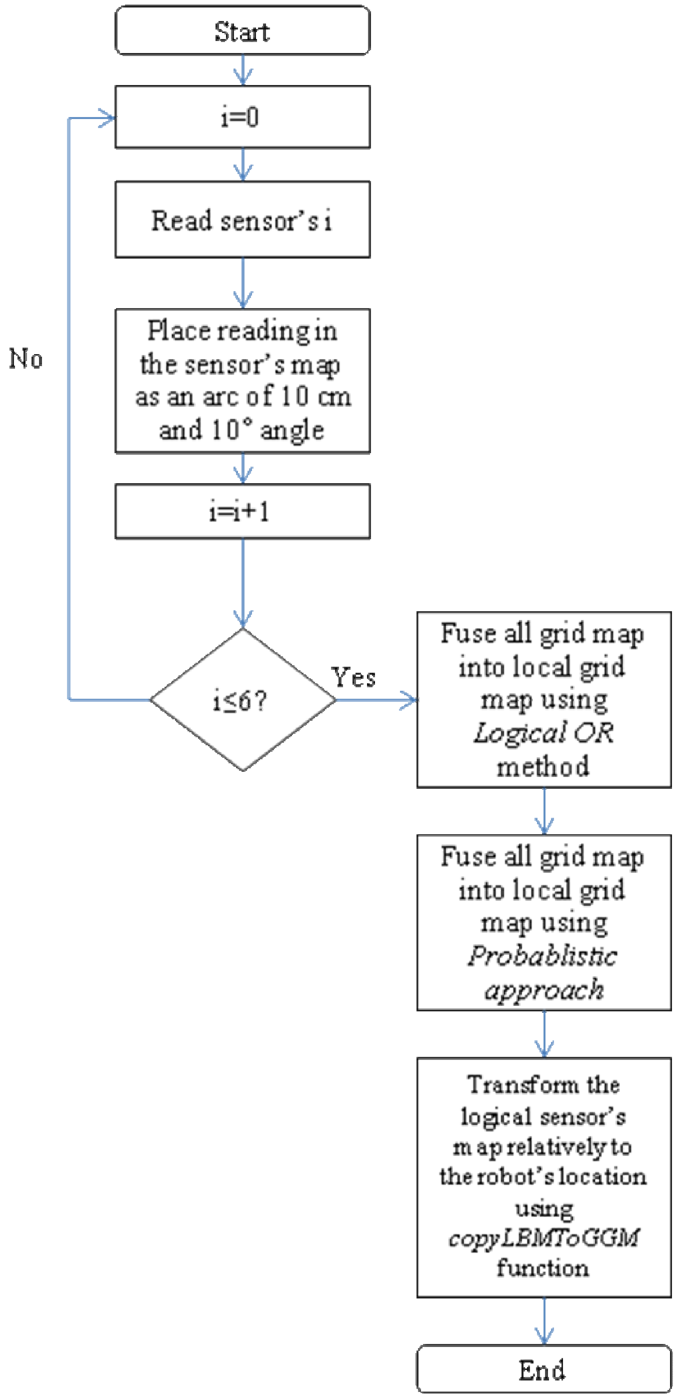
The real X-Y of the obstacle was derived according to the method described above.
Table 44 presents an example of the obstacle's location analysis.
The error is set to the absolute distance of the obstacle real location (taken from Table 43) and the one that was found from the data according to the method (denoted as X calculated and Y calculated in Table 44).
The mean error in X axis is 6.5cm, and in Y axis is 6.62cm. The error is caused by deviation in the robot's location relative to the rotation axis. Since the robot is massive, it is a hard task placing it exactly in the rotation axis, and its location varies a bit from on experiment to the other. This error is acceptable, since the maps resolution is 5cm and each obstacle is denoted as a group of cells (explained in section 6.3).

Table 44 Obstacle's location analysis

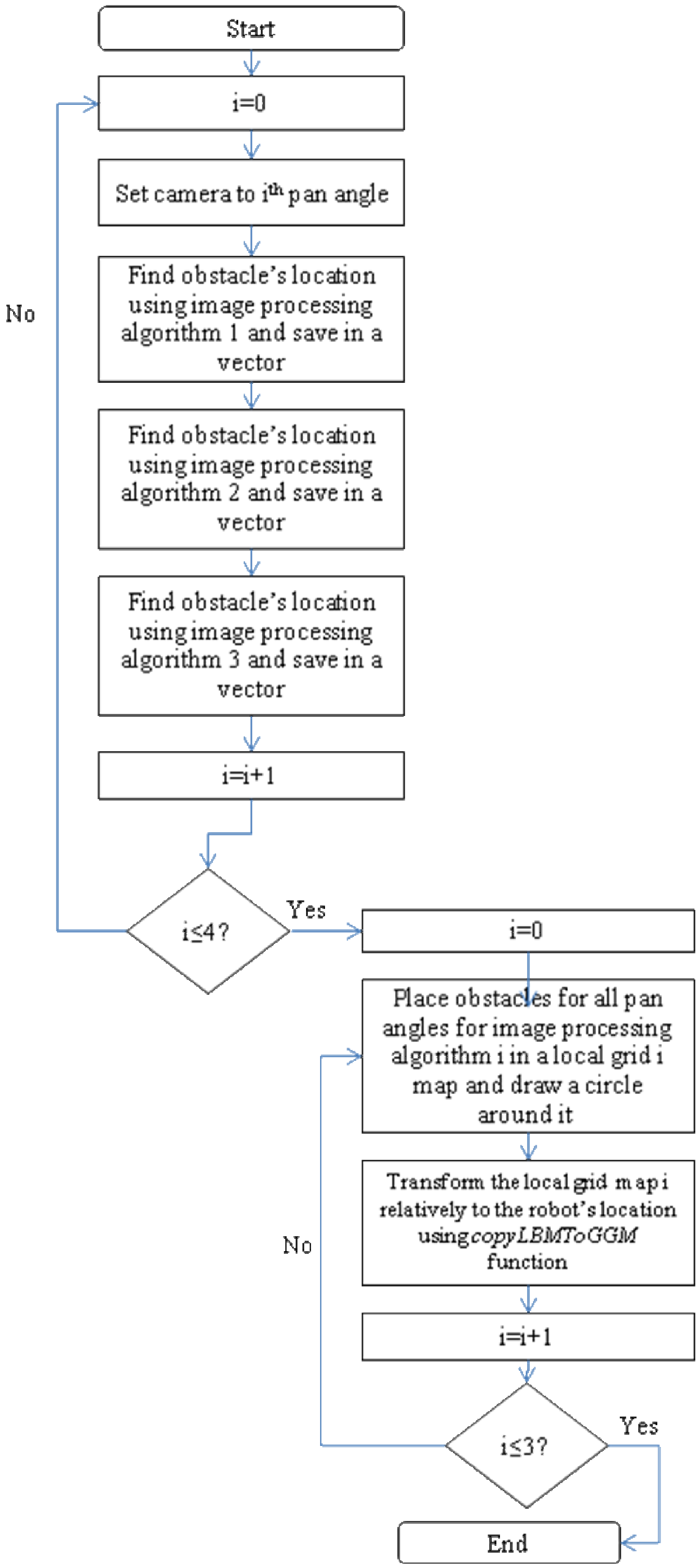
Pan angle	X Pic [cm]	Y Pic [cm]	Object	Obs. number	Cols [pixels]	Rows [pixels]	X calculated [cm]	Y calculated [cm]	Real X[cm]	Real Y[cm]
-40	6	0	1	1	402	531	56.93	1.83	50.78	-35.19
-40	6	0	2	2	635	353	91.15	32.40	96.65	-33.77
-40	6	0	3	3	541	191	154.77	30.86	144.40	-75.84
-20	10	0	1	1	104	575	50.05	-24.31	48.72	-39.96
-20	10	0	2	2	327	351	91.68	-7.25	93.67	-38.17
-20	10	0	3	3	217	199	150.06	-31.90	140.10	-81.30
0	13	0	2	2	15	405	78.82	-42.61	91.82	-42.61
0	13	0	5	5	687	258	121.81	48.93	134.81	48.93
20	17	0	5	5	378	250	125.09	-0.91	134.86	41.93
20	17	0	7	7	503	153	181.14	26.66	178.10	87.01
40	21	0	5	5	74	284	112.05	-46.65	136.82	36.29
40	21	0	6	6	496	292	109.29	16.65	94.02	83.00
40	21	0	7	7	188	164	172.75	-42.14	180.43	78.76

Appendix VI Mapping algorithms flowcharts

<u>Ultrasonic algorithms</u>

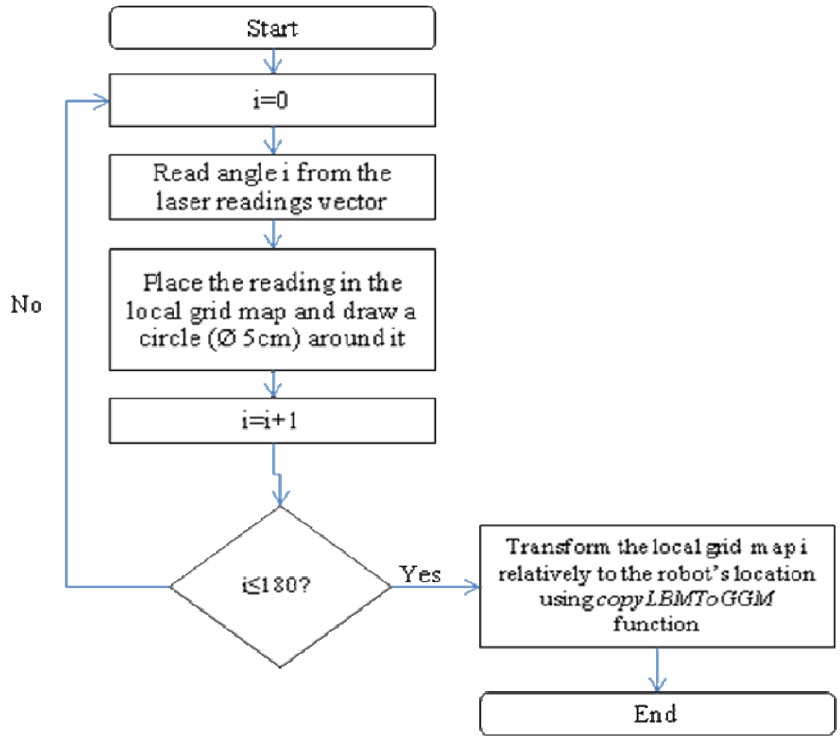


Camera algorithms

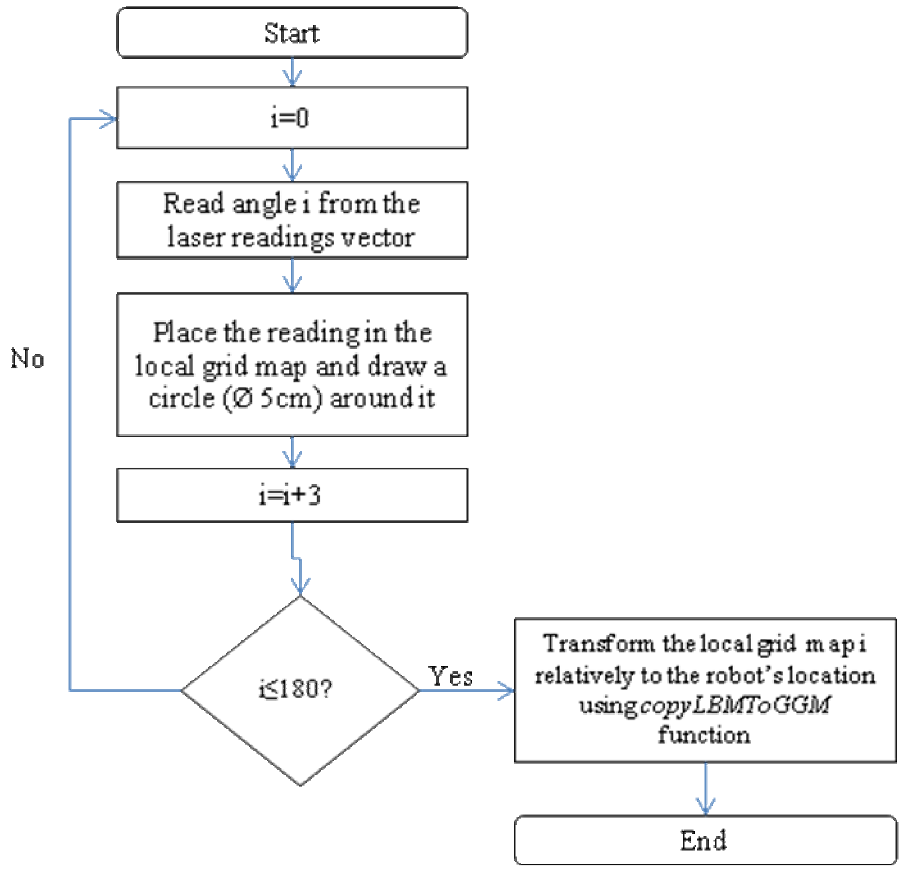


Laser algorithms

Laser1



Laser2



Appendix VII Code for analysis procedure

All experiments results were analyzed using MATLAB 7.1.
Since we can't tell in advance if the experiments are different from each other (as part of the statistical analysis requirements), several experiments were performed in each experiments set and the experiments that fulfill the volume of overlap region criteria were chosen (see section 2.5.9). The analysis procedure consists of the following steps. First, all logical sensors local maps and algorithms maps from all experiments and repetitions were read and saved. Next, the difference between experiments maps and repetitions was checked and the number of signed cells for each comparison was saved. The volume of overlap region (VOLR) was calculated for every experiments combination, and only the experiments that hold the criteria of negative VOLR was chosen. Next, algorithms maps were compared to the real world map (that was created according to the real obstacle's location in the experiment) and type I performance measures were calculated and saved. The performance measures were used to perform the statistical analysis procedure: calculating the number of repetitions required, friedman's test, multiple comparison procedure and sign test. The statistical analysis procedure was done manually using tables as shown in Appendix IX-Appendix XV. Table 45 presents a list of MATLAB functions that were used in the analysis and a brief explanation of their purpose. The table is followed by the functions MATLAB code.

Table 45 List of MATLAB functions and explanations

Function name	**Explanation**
Load_LS_maps;	This function loads all LS maps from the different exp. and rep. and returns the LS_Maps 5D array. The dimensions of the LS maps array are: LS_Maps[NumOfLS,NumOfExp,NumOfRep,107,48]
Load_SFA_maps;	This function loads all SFA maps from the different exp. and rep. and returns the SFA_Maps 5D array. The dimensions of the SFA maps array are: SFA_Maps[NumOfSFA,NumOfExp,NumOfRep,107,48]
CheckMaps(LS_Maps);	This function calculates the max. number of signed cells for every comparison between different experiments and repetitions in order to determine if the experiments are different enough.
one_count(GridMap);	%this function counts the num. of cells that are NOT zero
ChoosingExp(MaxExp,MaxRep,NumOfChosenExp);	This function takes all the experiments combinations and calculates the VOLR for each one of the combinations.
VOLR_Calc(MaxExp,MaxRep);	This function calculated the volume of over lap region for each experiments set using the MaxExp and MaxRep vector.
PM_Calc(SFA_Maps);	This function receives the sensor fusion algorithms maps and calculates the Sensor's fusion algorithms performance measures. The four PM are calculated according to type I performance measures equations.
truth_map;	This function creates the truth world map of the experiment

```
function LS_Maps=Load_LS_maps
%****************************************************************
%This function loads all LS maps from the different exp. and rep. and
%returns the LS_Maps 5D array.
%The dimensions of the LS maps array are:
%LS_Maps[NumOfLS,NumOfExp,NumOfRep,107,48]
%****************************************************************

LS_Maps=zeros(7,13,7,107,48);
Exp=[1:1:13];
[m,NumOfExp]=size(Exp);
NumOfRep=7;
NumOfLS=7;


%Reading data into Maps matrix
for i=1:NumOfExp
   for j=1:NumOfRep
      for k=1:NumOfLS
         filename=('H:\kapach\Thesis\Experiments\New Algorithm 16042007\Experiments\Exp.');
         filename=strcat(filename,int2str(Exp(i)),'\');
         filename=strcat(filename,'Rep.',int2str(j),'\');
         filename=strcat(filename,'LS_PPGM',int2str(k),'.data');
         fid=fopen(filename,'r');
         for rows=1:107
            for cols=1:48
               LS_Maps(k,i,j,rows,cols)=fscanf(fid,'%d',1);
               if LS_Maps(k,i,j,rows,cols)~=0
                  LS_Maps(k,i,j,rows,cols)=1;
               end %if
            end %for cols
         end
         fclose(fid);
      end
   end
end

function SFA_Maps=Load_SFA_maps()
%****************************************************************
%This function loads all SFA maps from the different exp. and rep. and
%returns the SFA_Maps 5D array.
%The dimensions of the SFA maps array are:
%SFA_Maps[NumOfSFA,NumOfExp,NumOfRep,107,48]

%The codes for the number of sensor fusion algorithms (SFA) are:
%
% TOTAL - 5 Algorithms.
%****************************************************************

NumOfPM=4;
Exp=[1:13];
[m,NumOfExp]=size(Exp);
NumOfRep=7;
NumOfSFA=5;
SFA_Code=[1:5];
SFA_Maps=zeros(NumOfSFA,9,NumOfRep,107,48);


%reading data into Maps matrix
for i=1:NumOfExp
```

```
   for j=1:NumOfRep
      for k=1:NumOfSFA
         filename=('H:\kapach\Thesis\Experiments\New Algorithm 16042007\Experiments\Exp.');
         filename=strcat(filename,int2str(Exp(i)),'\');
         filename=strcat(filename,'Rep.',int2str(j),'\');
         filename=strcat(filename,'PPGM',int2str(SFA_Code(k)),'.data');
         fid=fopen(filename,'r');
         for rows=1:107
            for cols=1:48
               SFA_Maps(k,i,j,rows,cols)=fscanf(fid,'%d',1);
               if SFA_Maps(k,i,j,rows,cols)~=0
                  SFA_Maps(k,i,j,rows,cols)=1;
               end %if
            end %for cols
         end
         fclose(fid);
      end
   end
end
```

function PM=PM_Calc(SFA_Maps);

```
%****************************************************************
%This function calculates the Sensor's fusion algorithms performance measures.
%The four PM are calculated according to the eq. in page 34.
%****************************************************************

%Constructing the TM
TM=truth_map;
[m,NumOfExp]=size(Exp);
NumOfRep=7;
NumOfSFA=5;
NumOfPM=4;
SFA_Code=[1:5];

PM=zeros(NumOfSFA,NumOfExp,NumOfRep,NumOfPM);

%Explanation for the PM matrix:
%The Matrix has 4 dimentions:
% The 1st dimention is the num. of SFA-  1- OR, 2- AND, 3- MOST, 4- AFL
% The second dimention is the number of exp., the third is the num of rep..
% The fifth dimention is 4, one cell for each PM  in the following order: OO,EE,OE,EO


GSize=107*48; %Global grid map's dimensions.
global O_tm E_tm
One_Count=0;
Zero_Count=0;
Occ_Coeff=0;
Empty_Coeff=0;

% GGM=zeros(4,160,48);
%
% GGM(1,:,:)=OR_map;
% GGM(2,:,:)=AND_map;
% GGM(3,:,:)=MOST_map;
% GGM(4,:,:)=AFL_map;

%figure(2);

% subplot(1,4,1); imshow(~OR_map); title('OR map');
```

```
% subplot(1,4,2); imshow(~AND_map); title('AND map');
% subplot(1,4,3); imshow(~MOST_map); title('MOST map');
% subplot(1,4,4); imshow(~AFL_map); title('AFL map');


%Counting the numbers of '1' and '0' in TM
O_tm=0;
[m,n]=size(TM);

for i=1:m
   for j=1:n
      if(TM(i,j)~=0)
         TM(i,j)=1;
         O_tm=O_tm+1;
      end
   end
end
E_tm=GSize-O_tm;

for i=1:NumOfSFA
   for j=1:NumOfExp
      for k=1:NumOfRep
         One_Count=0;
         for rows=1:m
            for cols=1:n
               if SFA_Maps(i,j,k,rows,cols)~=0
                  One_Count=One_Count+1; %counting the num. of signed cells for each SFA map
               end %if
            end %cols
         end %rows
         Zero_Count=GSize-One_Count;
         if (O_tm==One_Count)& (O_tm==0) %Calculating Occupy_Coeff for the current map
            Occ_Coeff=0;
         elseif (One_Count/O_tm<=1) & (One_Count/O_tm>=0)
            Occ_Coeff=One_Count/O_tm;
         else
            Occ_Coeff=O_tm/One_Count;
         end
         if (O_tm==Zero_Count) & (O_tm==GSize) %Calculating Empty_Coeff for the current map
            Empty_Corff=Occ_Coeff;
         elseif (Zero_Count/E_tm <=1) & (Zero_Count/E_tm>=0)
            Empty_Coeff=Zero_Count/E_tm;
         else
            Empty_Coeff=E_tm/Zero_Count;
         end
         %Calculating the four PM for the current map
         % OO
         if (O_tm>0)
            OO=Calc_OO(squeeze(SFA_Maps(i,j,k,:,:)),TM);
         else %OO=EE
            OO=Calc_EE(squeeze(SFA_Maps(i,j,k,:,:)),TM);
         end
         PM(i,j,k,1)=Occ_Coeff*OO;

         %Calculating EE
         if (E_tm>0)
            EE=Calc_EE(squeeze(SFA_Maps(i,j,k,:,:)),TM);
         else
            EE=Calc_OO(squeeze(SFA_Maps(i,j,k,:,:)),TM);
         end
```

```
            PM(i,j,k,2)=Empty_Coeff*EE;

            %Calculating OE
            if (E_tm>0)
               OE=Calc_OE(squeeze(SFA_Maps(i,j,k,:,:)),TM);
            else
               OE=1-Calc_OO(squeeze(SFA_Maps(i,j,k,:,:)),TM);
            end
            PM(i,j,k,3)=(1-Empty_Coeff)*OE;

            %Calculating EO
            if (O_tm>0)
               EO=Calc_EO(squeeze(SFA_Maps(i,j,k,:,:)),TM);
            else
               EO=1-Calc_EE(squeeze(SFA_Maps(i,j,k,:,:)),TM);
            end
            PM(i,j,k,4)=(1-Occ_Coeff)*EO;

        end %k
    end %j
end %i

%*********************************
% Writing the PM into files
%*********************************
for i=1:NumOfExp
    filename=('H:\kapach\Thesis\Experiments\New Algorithm 16042007\Experiments\PM\');
    filename=strcat(filename,int2str(Exp(i)),'.txt');
    fid=fopen(filename,'wt');
    for j=1:NumOfRep
        text='Repetition ';
        text=strcat(text,int2str(j), '  \n');
        fprintf(fid,text,'\n');
%        text='  OO    EE    OE    EO   ';
%        fprintf(fid,text');
%        fprintf(fid, '\n');
        for k=1:NumOfSFA
%            if (k==1) fprintf(fid, 'OR ');
%            elseif (k==2) fprintf(fid, ' AND ');
%            elseif (k==3) fprintf(fid, ' MOST ' );
%            elseif (k==4) fprintf(fid, ' AFL ');
%            end
            for m=1:NumOfPM
              fprintf(fid,'%4.3f',PM(k,i,j,m));
              fprintf(fid, ' ');
            end
            fprintf(fid,'\n');
        end
        fprintf(fid,'\n\n');
    end
    fclose(fid);
end


%*************************************
%              SUB-FUNCTIONS
%*************************************
function OO1= Calc_OO(A,TM)
%Calculates the number of '1' in GGM and TM devided by the num. of '1' in TM
global O_tm
```

```
[row,col]=size(A);
temp=0;
for m=1:row
   for n=1:col
    temp=temp+A(m,n)*TM(m,n);
   end
end

OO1=temp/O_tm;

%**********************************************************************************
function EE1=Calc_EE(A,TM)
%Calculates the number of '0' in GGM and TM devided by the num. of '0' in TM
global E_tm
[row,col]=size(A);
temp=0;
for m=1:row
   for n=1:col
      temp=temp+(1-A(m,n))*(1-TM(m,n));
   end
end

EE1=temp/E_tm;

%**********************************************************************************
function OE1=Calc_OE(A,TM)
%Calculates the num. of '1' in GGM and '0' in TM devided by the num. of '0' in TM
global E_tm
[row,col]=size(A);
temp=0;
for m=1:row
   for n=1:col
      temp=temp+A(m,n)*(1-TM(m,n));
   end
end
OE1=temp/E_tm;

%**********************************************************************************
function EO1=Calc_EO(A,TM)
%Calculates the num. of '0' in GGM and '1' in TM devided by the num. of '1' in TM
global O_tm
[row,col]=size(A);
temp=0;
for m=1:row
   for n=1:col
      temp=temp+((1-A(m,n))*TM(m,n));
   end
end
EO1=temp/O_tm;
%**********************************************************************************
```

```
function [MaxExp,MaxRep]=CheckMaps(LS_Maps)
%***************************************************************************
%This function calculates the max. number of signed cells
% for every comparison between different experiments and repetitions
% in order to determine if the experiments are different enough
%***************************************************************************
Exp=[1:1:14];
[m,NumOfExp]=size(Exp);
NumOfRep=10;
NumOfLS=7;

%Copying the LS_Maps to Maps matrix
Maps=zeros(NumOfLS,NumOfExp,NumOfRep,107,48);

for i=1:NumOfExp
   Maps(:,i,:,:,:)=LS_Maps(:,Exp(i),:,:,:);
end


%Structure of the maps array:
%maps[NumOfLS][NumOfExp][NumOfRep][160][48]


%SubMaps=zeros(NumOfLS,NumOfExp,160,48);
SignedCells=zeros(NumOfLS,NumOfExp,NumOfRep);

%********************************
% Different experiments
%********************************
%Calculation of the number of sighned cells for every comparison between
% the different experiments.
%for each experiment between every two rep., each logical sensor's map from
%one repetition is compared to all other LS maps from the other repetition.
%i.e, for LS1 - Exp1. Rep1. is compared with Exp2.
%Rep1.,Exp2.Rep2.,Exp2.Rep3 and so on.
ExpSignedCells=zeros(NumOfLS,NumOfExp-1,NumOfRep,NumOfExp,NumOfRep);
NumOfCompExp=0;
for i=1:NumOfLS
   for j=1:(NumOfExp-1)
      for k=1:NumOfRep
         for l=(j+1):NumOfExp
            for m=1:NumOfRep
               SubMaps=abs(Maps(i,j,k,:,:)-Maps(i,l,m,:,:));
               counter=one_count(squeeze(SubMaps));
               ExpSignedCells(i,j,k,l,m)=counter;
               NumOfCompExp=NumOfCompExp+1;
               end
            end
         end
      end
   end

NumOfCompExp


%Finding the max value for each comparison for each LS
%for each comparison, the maximum difference of all LS is saved.
%i.e for the comp. between Exp1.Rep1. and Exp2.Rep.5  the max. difference
%from all LS is saved.
MaxExp=zeros(NumOfExp-1,NumOfRep,NumOfExp,NumOfRep);
```

```
IndexExp=zeros(NumOfExp-1,NumOfRep,NumOfExp,NumOfRep);

for j=1:(NumOfExp-1)
   for k=1:NumOfRep
      for l=(j+1):NumOfExp
         for m=1:NumOfRep
            for i=1:NumOfLS
               if ExpSignedCells(i,j,k,l,m)>MaxExp(j,k,l,m)
                  MaxExp(j,k,l,m)=ExpSignedCells(i,j,k,l,m);
                  IndexExp(j,k,l,m)=i;
               end
            end
         end
      end
   end
end

% MaxExp=reshape(MaxExp, 1, (NumOfExp-1)*NumOfRep*NumOfExp*NumOfRep);
%********************************
% Different repetitions
%********************************
%Calculation of the number of sighned cells for every comparison between
% the repetitions
%for each exp., each LS map is compared to all other rep. in pairs.e.g -
%LS1 exp1. rep.1. with exp1.rep2., exp1.rep3.,exp1.rep4 and so on.
RepSignedCells=zeros(NumOfLS,NumOfExp,NumOfRep-1,NumOfRep);
NumOfCompRep=0;
for i=1:NumOfLS
   for j=1:NumOfExp
      for k=1:(NumOfRep-1)
         for m=(k+1):NumOfRep
            SubMaps=Maps(i,j,k,:,:)-Maps(i,j,m,:,:);
            NumOfCompRep=NumOfCompRep+1;
            counter=one_count(squeeze(SubMaps));
            RepSignedCells(i,j,k,m)=counter;
         end
      end
   end
end
NumOfCompRep

%For each exp. and each comp., the maximum value for all LS is saved.
MaxRep=zeros(NumOfExp,NumOfRep-1,NumOfRep);
IndexRep=zeros(NumOfExp,NumOfRep-1,NumOfRep);

for i=1:NumOfExp
   for j=1:NumOfRep-1
      for k=(j+1):NumOfRep
         for m=1:NumOfLS
            if RepSignedCells(m,i,j,k)>MaxRep(i,j,k)
               MaxRep(i,j,k)=RepSignedCells(m,i,j,k);
               IndexRep(i,j,k)=m;
            end
         end
      end
   end
end
```

```
function one_count=one_count(GridMap)
%********************************
%this function counts the num. of cells that are NOT zero
% and turns the maps into binary maps.
%********************************

one_count=0;
[n,m]=size(GridMap);


for i=1:n
   for j=1:m
      if (GridMap(i,j)~=0)
         one_count=one_count+1;
      end
   end
end

function VOLR=ChoosingExp(MaxExp,MaxRep,NumOfChosenExp)
%*********************************************************
%This function takes all the experiments combinations and calculates the
%VOLR for each one of the combinations.
%*********************************************************

NumOfRep=10;
% NumOfChosenExp=5;
%Building the Exp vector
A=[1:14];
Exp_Comb=nchoosek(A,NumOfChosenExp);
[rows,cols]=size(Exp_Comb);
count=0;
VOLR=zeros(rows,1+NumOfChosenExp+4);

for comb_num=1:rows
   %building the MaxExp1 array that contains the num. of signed cells from the
   % comparisons between the different experiments.
   %MaxExp1 is a 2-D array with dimentions ((NumOfChosenExp-
1)*NumOfRep,NumOfExpChosen*NumOfRep)

   %Choosing the wanted experiments from the MaxExp array
   Exp=Exp_Comb(comb_num,:);
   Exp=[1,6,10,11,12,13,14];
   [m,n]=size(Exp);

   %Creating the first row
   MaxExp1=squeeze(MaxExp(Exp(1),:,Exp(1),:));
   for i=2:n
      MaxExp1=[MaxExp1,squeeze(MaxExp(Exp(1),:,Exp(i),:))];
   end

   %Creating the rest of the rows
   for i=2:(n-1)
      temp=squeeze(MaxExp(Exp(i),:,Exp(1),:));
      for j=2:n
         temp=[temp,squeeze(MaxExp(Exp(i),:,Exp(j),:))];
      end %j
      MaxExp1=[MaxExp1;temp];
   end %i
   MaxExp1=reshape(MaxExp1,1,(NumOfChosenExp-1)*NumOfRep*NumOfChosenExp*NumOfRep);
   % MaxExp1=reshape(MaxExp1,1,(n-1)*10*n*10);
```

```
    %building the MaxRep1 array that contains the num. of signed cells from the
    %comparisons between the different repetitions.
    %MaxRep1 is a 2-D array with dimentions ((NumOfRep-1)*10,NumOfRep)
    MaxRep1=squeeze(MaxRep(Exp(1),:,:));

    for i=2:n
       temp=squeeze(MaxRep(Exp(i),:,:));
       MaxRep1=[MaxRep1;temp];
    end
    MaxRep1=reshape(MaxRep1,1,(NumOfRep-1)*NumOfChosenExp*NumOfRep);
    temp=zeros(1,4);
    [f,temp(1),temp(2),temp(3),temp(4)]=VOLR_Calc(MaxExp1,MaxRep1);
        VOLR(comb_num,1)=f;
        VOLR(comb_num,2:8)=Exp;
        VOLR(comb_num,9)=temp(1);
        VOLR(comb_num,10)=temp(2);
         VOLR(comb_num,11)=temp(3);
          VOLR(comb_num,12)=temp(4);

VOLR=sortrows(VOLR,1);
```

function [f,Min_Exp,Max_Exp,Min_Rep,Max_Rep]=VOLR_Calc(MaxExp,MaxRep)

```
%****************************************************
% This function calculated the volume of over lap region for each
% experiments set using the MaxExp and MaxRep vector.
%***************************************************

MaxExp=sort(MaxExp);
MaxRep=sort(MaxRep);
[m,n]=size(MaxExp);

i=1;
while(MaxExp(i)==0)
   i=i+1;
end

Min_Exp=MaxExp(i);
Max_Exp=MaxExp(n);

[m,n]=size(MaxRep);
i=1;
while (MaxRep(i)==0)
   i=i+1;
end

Min_Rep=MaxRep(i);
Max_Rep=MaxRep(n);

f=(min(Max_Exp,Max_Rep)-max(Min_Exp,Min_Rep))/(max(Max_Exp,Max_Rep)-min(Min_Exp,Min_Rep));
```

function truth_map=truth_map;

```
%************************************************
%This function creates the truth world map of the experiment
%************************************************

TM=zeros(107,48);

%Coordinates of the Center of mass of each obstable
x=[21,45,20,43,79];
y=[40,39,11,7,22];
```

```
[n,m]=size(x);

for i=1:m
    x(i)=x(i)*5;
    y(i)=y(i)*5;
end

%drawing the center of the obstacles
% for i=1:m
%    TM(x(i),y(i))=1;
% end

%drawing a circle around the center of the obstacles
for i=1:m
    for k=1:1:15
        for theta=0:20:360
            phi=pi/180*theta;
            xTag=k*cos(phi);
            yTag=k*sin(phi);
            x0=xTag+x(i);
            y0=yTag+y(i);
            x1=round(x0/5);
            y1=round(y0/5);
            if (x1>=1 & x1<=160 & y1>=1 & y1<=48)
                TM(x1,y1)=TM(x1,y1)+1;
            end
        end
    end
end
truth_map=TM;
```

Appendix VIII Raw data for extended fusion framework experimental set

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
1	OR	1	0.038	0	0.9988	0
		2	0.038	0	0.9988	0
		3	0.038	0	1	0
		4	0.038	0	1	0
		5	0.038	0	1	0
		6	0.038	0	1	0
		7	0.038	0	1	0
	AND	1	0	0.962	0	1
		2	0	0.962	0	1
		3	0	0.962	0	1
		4	0	0.962	0	1
		5	0	0.962	0	1
		6	0	0.962	0	1
		7	0	0.962	0	1
	MOST	1	0.0121	0.9652	0.0001	0.7864
		2	0.0044	0.9632	0.0001	0.8454
		3	0.0016	0.9627	0.0001	0.9041
		4	0.0057	0.9636	0.0001	0.8364
		5	0.0109	0.9643	0.0001	0.7802
		6	0.0188	0.9651	0.0002	0.7162
		7	0.019	0.9655	0.0001	0.7267
	AFL	1	0.0121	0.9652	0.0001	0.7864
		2	0.0044	0.9632	0.0001	0.8454
		3	0.0016	0.9627	0.0001	0.9041
		4	0.0057	0.9636	0.0001	0.8364
		5	0.0109	0.9643	0.0001	0.7802
		6	0.0188	0.9651	0.0002	0.7162
		7	0.019	0.9655	0.0001	0.7267

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
2	OR	1	0.038	0	1	0
		2	0.038	0	1	0
		3	0.038	0	1	0
		4	0.038	0	1	0
		5	0.038	0	1	0
		6	0.038	0	1	0
		7	0.038	0	1	0
	AND	1	0	0.962	0	1
		2	0	0.962	0	1
		3	0	0.962	0	1
		4	0	0.962	0	1
		5	0	0.962	0	1
		6	0	0.962	0	1
		7	0	0.962	0	1
	MOST	1	0.0048	0.9634	0.0001	0.8459
		2	0.0028	0.9631	0.0001	0.8798
		3	0.0018	0.9627	0.0001	0.8941
		4	0.0046	0.9634	0.0001	0.8508
		5	0.011	0.9646	0.0001	0.7854
		6	0.005	0.9634	0.0001	0.8409
		7	0.0032	0.9631	0.0001	0.8698
	AFL	1	0.3513	0.9754	0.0001	0.0385
		2	0.3654	0.9751	0	0.0065
		3	0.3648	0.9753	0	0.016
		4	0.3511	0.9758	0.0001	0.0536
		5	0.4475	0.979	0	0.0219
		6	0.3111	0.9748	0.0001	0.0855
		7	0.323	0.975	0.0001	0.0717

			Performance measure			
Experiment	Algorithm	Repetition	OO	EE	OE	EO
3	OR	1	0.038	0	1	0
		2	0.038	0	1	0
		3	0.038	0	1	0
		4	0.0383	0.0001	0.9807	0
		5	0.0383	0.0001	0.9807	0
		6	0.038	0	1	0
		7	0.0383	0.0001	0.9807	0
	AND	1	0	0.962	0	1
		2	0	0.962	0	1
		3	0	0.962	0	1
		4	0	0.962	0	1
		5	0	0.962	0	1
		6	0	0.962	0	1
		7	0	0.962	0	1
	MOST	1	0.0792	0.9683	0.0002	0.4586
		2	0.107	0.9711	0.0002	0.43
		3	0.059	0.9682	0.0002	0.5462
		4	0.0511	0.9673	0.0002	0.5588
		5	0.054	0.9687	0.0001	0.577
		6	0.095	0.9699	0.0002	0.4437
		7	0.0525	0.9689	0.0001	0.5859
	AFL	1	0.3868	0.9703	0.0001	0.0465
		2	0.3416	0.9743	0	0.0134
		3	0.287	0.9725	0.0001	0.0358
		4	0.3718	0.976	0	0.0282
		5	0.2902	0.9732	0.0001	0.0595
		6	0.4573	0.9774	0	0.0107
		7	0.2823	0.973	0.0001	0.067

			Performance measure			
Experiment	Algorithm	Repetition	OO	EE	OE	EO
4	OR	1	0.038	0	1	0
		2	0.038	0	1	0
		3	0.038	0	1	0
		4	0.038	0	1	0
		5	0.038	0	1	0
		6	0.038	0	1	0
		7	0.038	0	1	0
	AND	1	0	0.962	0	1
		2	0	0.962	0	1
		3	0	0.962	0	1
		4	0	0.962	0	1
		5	0	0.962	0	1
		6	0	0.962	0	1
		7	0	0.962	0	1
	MOST	1	0.2172	0.907	0.0021	0.279
		2	0.2119	0.9018	0.0024	0.2875
		3	0.2825	0.9311	0.0011	0.2071
		4	0.3346	0.9484	0.0006	0.151
		5	0.2852	0.9464	0.0006	0.1681
		6	0.2652	0.9256	0.0013	0.2246
		7	0.2767	0.9297	0.0012	0.2122
	AFL	1	0.2172	0.907	0.0021	0.279
		2	0.2119	0.9018	0.0024	0.2875
		3	0.2825	0.9311	0.0011	0.2071
		4	0.3346	0.9484	0.0006	0.151
		5	0.2852	0.9464	0.0006	0.1681
		6	0.2652	0.9256	0.0013	0.2246
		7	0.2767	0.9297	0.0012	0.2122

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
5	OR	1	0.038	0	1	0
		2	0.038	0	1	0
		3	0.038	0	1	0
		4	0.038	0	1	0
		5	0.038	0	1	0
		6	0.038	0	1	0
		7	0.038	0	1	0
	AND	1	0	0.962	0	1
		2	0	0.962	0	1
		3	0	0.962	0	1
		4	0	0.962	0	1
		5	0	0.962	0	1
		6	0	0.962	0	1
		7	0	0.962	0	1
	MOST	1	0.3192	0.9671	0.0001	0.055
		2	0.3065	0.9713	0	0.0138
		3	0.3433	0.9721	0	0.0193
		4	0.3067	0.9625	0.0002	0.0862
		5	0.3592	0.9711	0.0001	0.0331
		6	0.2676	0.9713	0	0.0112
		7	0.318	0.9657	0.0001	0.0655
	AFL	1	0.2079	0.8695	0.0045	0.2897
		2	0.1953	0.8717	0.0043	0.3095
		3	0.2143	0.8765	0.004	0.2845
		4	0.2108	0.8576	0.0054	0.2747
		5	0.2298	0.8708	0.0044	0.2569
		6	0.1832	0.873	0.0042	0.327
		7	0.2144	0.8593	0.0052	0.2702

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
6	OR	1	0.038	0	1	0
		2	0.038	0	1	0
		3	0.038	0	1	0
		4	0.038	0	1	0
		5	0.038	0	1	0
		6	0.038	0	1	0
		7	0.038	0	1	0
	AND	1	0	0.962	0	1
		2	0	0.962	0	1
		3	0	0.962	0	1
		4	0	0.962	0	1
		5	0	0.962	0	1
		6	0	0.962	0	1
		7	0	0.962	0	1
	MOST	1	0.2964	0.9725	0	0.0143
		2	0.3333	0.9707	0.0001	0.0287
		3	0.3754	0.9764	0.0001	0.0369
		4	0.3627	0.9719	0	0.0274
		5	0.343	0.9745	0	0.0199
		6	0.3448	0.9715	0	0.0253
		7	0.3744	0.9753	0	0
	AFL	1	0.2964	0.9725	0	0.0143
		2	0.3333	0.9707	0.0001	0.0287
		3	0.3754	0.9764	0.0001	0.0369
		4	0.3627	0.9719	0	0.0274
		5	0.343	0.9745	0	0.0199
		6	0.3448	0.9715	0	0.0253
		7	0.3744	0.9753	0	0

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
7	OR	1	0.22551	0.81768	0.009139	0.17865
		2	0.214	0.77953	0.013697	0.12564
		3	0.16511	0.74157	0.019246	0.21369
		4	0.18265	0.73724	0.019972	0.13953
		5	0.17493	0.78433	0.013028	0.25505
		6	0.15926	0.70492	0.025709	0.16786
		7	0.19831	0.83198	0.007653	0.2678
	AND	1	0.000657	0.96297	0	0.94938
		2	0.005444	0.96362	9.53E-05	0.84134
		3	0.000657	0.96297	0	0.94938
		4	0.000947	0.96316	0	0.93941
		5	0.004471	0.96386	4.93E-05	0.86601
		6	0.00355	0.96368	4.27E-05	0.88047
		7	0.004734	0.96385	5.60E-05	0.86114
	MOST	1	0.29507	0.97445	0.000118	0.1002
		2	0.37365	0.97553	1.45E-05	0.009546
		3	0.2784	0.97429	0.000137	0.12455
		4	0.25247	0.97287	0.000142	0.12426
		5	0.17988	0.9699	0.000187	0.16963
		6	0.22974	0.97122	0.000128	0.10154
		7	0.19282	0.97028	0.000173	0.15179
	AFL	1	0.41584	0.97474	3.38E-05	0.019726
		2	0.39474	0.96558	0.000187	0.077935
		3	0.40097	0.97653	9.34E-06	0.006101
		4	0.36092	0.97534	3.29E-05	0.022459
		5	0.21964	0.97037	0.000108	0.081183
		6	0.3048	0.97291	2.64E-05	0.01762
		7	0.23406	0.97075	8.88E-05	0.064826

Appendix IX Statistical evauation - Friedman's ranking

		Experiment 1			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OO	1	4	1	2.5	2.5
	2	4	1	2.5	2.5
	3	4	1	2.5	2.5
	4	4	1	2.5	2.5
	5	4	1	2.5	2.5
	6	4	1	2.5	2.5
	7	4	1	2.5	2.5
	Sum of ranks	**28**	**7**	**17.5**	**17.5**

		Experiment 1			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EE	1	1	2	3.5	3.5
	2	1	2	3.5	3.5
	3	1	2	3.5	3.5
	4	1	2	3.5	3.5
	5	1	2	3.5	3.5
	6	1	2	3.5	3.5
	7	1	2	3.5	3.5
	Sum of ranks	**7**	**14**	**24.5**	**24.5**

		Experiment 1			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OE	1	1	4	2.5	2.5
	2	1	4	2.5	2.5
	3	1	4	2.5	2.5
	4	1	4	2.5	2.5
	5	1	4	2.5	2.5
	6	1	4	2.5	2.5
	7	1	4	2.5	2.5
	Sum of ranks	**7**	**28**	**17.5**	**17.5**

		Experiment 1			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EO	1	1	4	2.5	2.5
	2	1	4	2.5	2.5
	3	1	4	2.5	2.5
	4	1	4	2.5	2.5
	5	1	4	2.5	2.5
	6	1	4	2.5	2.5
	7	1	4	2.5	2.5
	Sum of ranks	**7**	**28**	**17.5**	**17.5**

		Experiment 2			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OO	1	3	1	2	4
	2	3	1	2	4
	3	3	1	2	4
	4	3	1	2	4
	5	3	1	2	4
	6	3	1	2	4
	7	3	1	2	4
	Sum of ranks	**21**	**7**	**14**	**28**

		Experiment 2			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EE	1	1	2	3	4
	2	1	2	3	4
	3	1	2	3	4
	4	1	2	3	4
	5	1	2	3	4
	6	1	2	3	4
	7	1	2	3	4
	Sum of ranks	**7**	**14**	**21**	**28**

		Experiment 2			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OE	1	1	4	2.5	2.5
	2	1	3.5	2	3.5
	3	1	3.5	2	3.5
	4	1	4	2.5	2.5
	5	1	3.5	2	3.5
	6	1	4	2.5	2.5
	7	1	4	2.5	2.5
	Sum of ranks	**7**	**26.5**	**16**	**20.5**

		Experiment 2			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EO	1	4	1	2	3
	2	4	1	2	3
	3	4	1	2	3
	4	4	1	2	3
	5	4	1	2	3
	6	4	1	2	3
	7	4	1	2	3
	Sum of ranks	**28**	**7**	**14**	**21**

		Experiment 3			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OO	1	2	1	3	4
	2	2	1	3	4
	3	2	1	3	4
	4	2	1	3	4
	5	2	1	3	4
	6	2	1	3	4
	7	2	1	3	4
	Sum of ranks	**14**	**7**	**21**	**28**

		Experiment 3			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EE	1	1	2	3	4
	2	1	2	3	4
	3	1	2	3	4
	4	1	2	3	4
	5	1	2	3	4
	6	1	2	3	4
	7	1	2	3	4
	Sum of ranks	**7**	**14**	**21**	**28**

		Experiment 3			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OE	1	1	4	2	3
	2	1	4	2	3
	3	1	4	2	3
	4	1	4	2	3
	5	1	4	2	3
	6	1	4	2	3
	7	1	4	2	3
	Sum of ranks	**7**	**28**	**14**	**21**

		Experiment 3			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EO	1	4	1	2	3
	2	4	1	2	3
	3	4	1	2	3
	4	4	1	2	3
	5	4	1	2	3
	6	4	1	2	3
	7	4	1	2	3
	Sum of ranks	**28**	**7**	**14**	**21**

		Experiment 4 Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
OO	1	2	1	3.5	3.5
	2	2	1	3.5	3.5
	3	2	1	3.5	3.5
	4	2	1	3.5	3.5
	5	2	1	3.5	3.5
	6	2	1	3.5	3.5
	7	2	1	3.5	3.5
	Sum of ranks	**14**	**7**	**24.5**	**24.5**

		Experiment 4 Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
EE	1	1	4	2.5	2.5
	2	1	4	2.5	2.5
	3	1	4	2.5	2.5
	4	1	4	2.5	2.5
	5	1	4	2.5	2.5
	6	1	4	2.5	2.5
	7	1	4	2.5	2.5
	Sum of ranks	**7**	**28**	**17.5**	**17.5**

		Experiment 4 Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
OE	1	1	4	2.5	2.5
	2	1	4	2.5	2.5
	3	1	4	2.5	2.5
	4	1	4	2.5	2.5
	5	1	4	2.5	2.5
	6	1	4	2.5	2.5
	7	1	4	2.5	2.5
	Sum of ranks	**7**	**28**	**17.5**	**17.5**

		Experiment 4 Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
EO	1	4	1	2.5	2.5
	2	4	1	2.5	2.5
	3	4	1	2.5	2.5
	4	4	1	2.5	2.5
	5	4	1	2.5	2.5
	6	4	1	2.5	2.5
	7	4	1	2.5	2.5
	Sum of ranks	**28**	**7**	**17.5**	**17.5**

		Experiment 5			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OO	1	2	1	4	3
	2	2	1	4	3
	3	2	1	4	3
	4	2	1	4	3
	5	2	1	4	3
	6	2	1	4	3
	7	2	1	4	3
	Sum of ranks	**14**	**7**	**28**	**21**

		Experiment 5			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EE	1	1	3	4	2
	2	1	3	4	2
	3	1	3	4	2
	4	1	3	4	2
	5	1	3	4	2
	6	1	3	4	2
	7	1	3	4	2
	Sum of ranks	**7**	**21**	**28**	**14**

		Experiment 5			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OE	1	1	4	3	2
	2	1	4	3	2
	3	1	4	3	2
	4	1	4	3	2
	5	1	4	3	2
	6	1	4	3	2
	7	1	4	3	2
	Sum of ranks	**7**	**28**	**21**	**14**

		Experiment 5			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EO	1	4	1	3	2
	2	4	1	3	2
	3	4	1	3	2
	4	4	1	3	2
	5	4	1	3	2
	6	4	1	3	2
	7	4	1	3	2
	Sum of ranks	**28**	**7**	**21**	**14**

		Experiment 6			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OO	1	2	1	3.5	3.5
	2	2	1	3.5	3.5
	3	2	1	3.5	3.5
	4	2	1	3.5	3.5
	5	2	1	3.5	3.5
	6	2	1	3.5	3.5
	7	2	1	3.5	3.5
	Sum of ranks	**14**	**7**	**24.5**	**24.5**

		Experiment 6			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EE	1	1	2	3.5	3.5
	2	1	2	3.5	3.5
	3	1	2	3.5	3.5
	4	1	2	3.5	3.5
	5	1	2	3.5	3.5
	6	1	2	3.5	3.5
	7	1	2	3.5	3.5
	Sum of ranks	**7**	**14**	**24.5**	**24.5**

		Experiment 6			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
OE	1	1	4	2.5	2.5
	2	1	4	2.5	2.5
	3	1	4	2.5	2.5
	4	1	4	2.5	2.5
	5	1	4	2.5	2.5
	6	1	4	2.5	2.5
	7	1	3	3	3
	Sum of ranks	**7**	**27**	**18**	**18**

		Experiment 6			
		Sensor fusion algorithm			
Performance measure	**Repetition**	**OR**	**AND**	**MOST**	**AFL**
EO	1	4	1	2.5	2.5
	2	4	1	2.5	2.5
	3	4	1	2.5	2.5
	4	4	1	2.5	2.5
	5	4	1	2.5	2.5
	6	4	1	2.5	2.5
	7	3	1	3	3
	Sum of ranks	**27**	**7**	**18**	**18**

		Experiment 7			
		Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
OO	1	2	1	3	4
	2	2	1	3	4
	3	2	1	3	4
	4	2	1	3	4
	5	2	1	3	4
	6	2	1	3	4
	7	3	1	2	4
	Sum of ranks	**15**	**7**	**20**	**28**

		Experiment 7			
		Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
EE	1	1	2	3	4
	2	1	2	4	3
	3	1	2	3	4
	4	1	2	3	4
	5	1	2	3	4
	6	1	2	3	4
	7	1	2	3	4
	Sum of ranks	**7**	**14**	**22**	**27**

		Experiment 7			
		Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
OE	1	1	4	2	3
	2	1	3	4	2
	3	1	4	2	3
	4	1	4	2	3
	5	1	4	2	3
	6	1	4	2	3
	7	1	4	2	3
	Sum of ranks	**7**	**27**	**16**	**20**

		Experiment 7			
		Sensor fusion algorithm			
Performance measure	Repetition	OR	AND	MOST	AFL
EO	1	2	1	3	4
	2	2	1	3	4
	3	2	1	3	4
	4	2	1	3	4
	5	2	1	3	4
	6	2	1	3	4
	7	2	1	3	4
	Sum of ranks	**14**	**7**	**21**	**28**

Appendix X Statistial evaluation - Multiple comparison procedure

Experiment 1							
OO measure				**EE measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
OR	28	A		MOST	24.5	A	
MOST	17.5	A	B	AFL	24.5	A	
AFL	17.5	A	B	AND	14	A	B
AND	7		B	OR	7		B
OE measure				**EO measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
AND	28	A		AND	28	A	
MOST	17.5	A	B	MOST	17.5	A	B
AFL	17.5	A	B	AFL	17.5	A	B
OR	7		B	OR	7		B

Experiment 2							
OO measure				**EE measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
AFL	28	A		AFL	28	A	
OR	21	A	B	MOST	21	A	B
MOST	14	A	B	AND	14	A	B
AND	7		B	OR	7		B
OE measure				**EO measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
AND	26.5	A		OR	28	A	
AFL	20.5	A	B	AFL	21	A	B
MOST	16	A	B	MOST	14	A	B
OR	7		B	AND	7		B

<table>
<tr><th colspan="8">Experiment 3</th></tr>
<tr><th colspan="4">OO measure</th><th colspan="4">EE measure</th></tr>
<tr><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td></tr>
<tr><td>AFL</td><td>28</td><td>A</td><td></td><td>AFL</td><td>28</td><td>A</td><td></td></tr>
<tr><td>MOST</td><td>21</td><td>A</td><td>B</td><td>MOST</td><td>21</td><td>A</td><td>B</td></tr>
<tr><td>OR</td><td>14</td><td>A</td><td>B</td><td>AND</td><td>14</td><td>A</td><td>B</td></tr>
<tr><td>AND</td><td>7</td><td></td><td>B</td><td>OR</td><td>7</td><td></td><td>B</td></tr>
<tr><th colspan="4">OE measure</th><th colspan="4">EO measure</th></tr>
<tr><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td></tr>
<tr><td>AND</td><td>28</td><td>A</td><td></td><td>OR</td><td>28</td><td>A</td><td></td></tr>
<tr><td>AFL</td><td>21</td><td>A</td><td>B</td><td>AFL</td><td>21</td><td>A</td><td>B</td></tr>
<tr><td>MOST</td><td>14</td><td>A</td><td>B</td><td>MOST</td><td>14</td><td>A</td><td>B</td></tr>
<tr><td>OR</td><td>7</td><td></td><td>B</td><td>AND</td><td>7</td><td></td><td>B</td></tr>
</table>

<table>
<tr><th colspan="8">Experiment 4</th></tr>
<tr><th colspan="4">OO measure</th><th colspan="4">EE measure</th></tr>
<tr><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td></tr>
<tr><td>MOST</td><td>24.5</td><td>A</td><td></td><td>AND</td><td>28</td><td>A</td><td></td></tr>
<tr><td>AFL</td><td>24.5</td><td>A</td><td>B</td><td>MOST</td><td>17.5</td><td>A</td><td>B</td></tr>
<tr><td>OR</td><td>14</td><td>A</td><td>B</td><td>AFL</td><td>17.5</td><td>A</td><td>B</td></tr>
<tr><td>AND</td><td>7</td><td></td><td>B</td><td>OR</td><td>7</td><td></td><td>B</td></tr>
<tr><th colspan="4">OE measure</th><th colspan="4">EO measure</th></tr>
<tr><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td><td>Sensor fusion algorithm</td><td>Sum of ranks</td><td colspan="2">Sub groups</td></tr>
<tr><td>AND</td><td>28</td><td>A</td><td></td><td>OR</td><td>28</td><td>A</td><td></td></tr>
<tr><td>MOST</td><td>17.5</td><td>A</td><td>B</td><td>MOST</td><td>17.5</td><td>A</td><td>B</td></tr>
<tr><td>AFL</td><td>17.5</td><td>A</td><td>B</td><td>AFL</td><td>17.5</td><td>A</td><td>B</td></tr>
<tr><td>OR</td><td>7</td><td></td><td>B</td><td>AND</td><td>7</td><td></td><td>B</td></tr>
</table>

Experiment 5							
OO measure				**EE measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
MOST	28	A		MOST	28	A	
AFL	21	A	B	AND	21	A	B
OR	14	A	B	AFL	14	A	B
AND	7		B	OR	7		B
OE measure				**EO measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
AND	28	A		OR	28	A	
MOST	21	A	B	MOST	21	A	B
AFL	14	A	B	AFL	14	A	B
OR	7		B	AND	7		B

Experiment 6							
OO measure				**EE measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
MOST	24.5	A		MOST	24.5	A	
AFL	24.5	A	B	AFL	24.5	A	B
OR	14	A	B	AND	14	A	B
AND	7		B	OR	7		B
OE measure				**EO measure**			
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups	
AND	27	A		OR	27	A	
MOST	18	A	B	MOST	18	A	B
AFL	18	A	B	AFL	18	A	B
OR	7		B	AND	7		B

Experiment 7								
OO measure				**EE measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups		
AFL	28	A		AFL	27	A		
MOST	20	A	B	MOST	22	A	B	
OR	15	A	B	AND	14	A	B	C
AND	7		B	OR	7			C
OE measure				**EO measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups		Sensor fusion algorithm	Sum of ranks	Sub groups		
AND	27	A		AFL	28	A		
AFL	20	A	B	MOST	21	A	B	
MOST	16	A	B	OR	14	A	B	
OR	7		B	AND	7		B	

Appendix XI Statistical evaluation - Sign test results

	Experiment 1											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.0121	0.0121	0.9652	0.9652	0.0001	0.0001	0.7864	0.7864	Ties	Ties	Ties	Ties
2	0.0044	0.0044	0.9632	0.9632	0.0001	0.0001	0.8454	0.8454	Ties	Ties	Ties	Ties
3	0.0016	0.0016	0.9627	0.9627	0.0001	0.0001	0.9041	0.9041	Ties	Ties	Ties	Ties
4	0.0057	0.0057	0.9636	0.9636	0.0001	0.0001	0.8364	0.8364	Ties	Ties	Ties	Ties
5	0.0109	0.0109	0.9643	0.9643	0.0001	0.0001	0.7802	0.7802	Ties	Ties	Ties	Ties
6	0.0188	0.0188	0.9651	0.9651	0.0002	0.0002	0.7162	0.7162	Ties	Ties	Ties	Ties
7	0.019	0.019	0.9655	0.9655	0.0001	0.0001	0.7267	0.7267	Ties	Ties	Ties	Ties
									Ties	**Ties**	**Ties**	**Ties**

	Experiment 2											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.0048	0.3513	0.9634	0.9754	0.0001	0.0001	0.8459	0.0385	AFL	AFL	TIES	AFL
2	0.0028	0.3654	0.9631	0.9751	0.0001	0	0.8798	0.0065	AFL	AFL	AFL	AFL
3	0.0018	0.3648	0.9627	0.9753	0.0001	0	0.8941	0.016	AFL	AFL	AFL	AFL
4	0.0046	0.3511	0.9634	0.9758	0.0001	0.0001	0.8508	0.0536	AFL	AFL	TIES	AFL
5	0.011	0.4475	0.9646	0.979	0.0001	0	0.7854	0.0219	AFL	AFL	AFL	AFL
6	0.005	0.3111	0.9634	0.9748	0.0001	0.0001	0.8409	0.0855	AFL	AFL	TIES	AFL
7	0.0032	0.323	0.9631	0.975	0.0001	0.0001	0.8698	0.0717	AFL	AFL	TIES	AFL
									AFL	**AFL**	**AFL**	**AFL**

	Experiment 3											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.0792	0.3868	0.9683	0.9703	0.0002	0.0001	0.4586	0.0465	AFL	AFL	AFL	AFL
2	0.107	0.3416	0.9711	0.9743	0.0002	0	0.43	0.0134	AFL	AFL	AFL	AFL
3	0.059	0.287	0.9682	0.9725	0.0002	0.0001	0.5462	0.0358	AFL	AFL	AFL	AFL
4	0.0511	0.3718	0.9673	0.976	0.0002	0	0.5588	0.0282	AFL	AFL	AFL	AFL
5	0.054	0.2902	0.9687	0.9732	0.0001	0.0001	0.577	0.0595	AFL	AFL	TIES	AFL
6	0.095	0.4573	0.9699	0.9774	0.0002	0	0.4437	0.0107	AFL	AFL	AFL	AFL
7	0.0525	0.2823	0.9689	0.973	0.0001	0.0001	0.5859	0.067	AFL	AFL	TIES	AFL
									AFL	**AFL**	**AFL**	**AFL**

	Experiment 4											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.2172	0.2172	0.907	0.907	0.0021	0.0021	0.279	0.279	Ties	Ties	Ties	Ties
2	0.2119	0.2119	0.9018	0.9018	0.0024	0.0024	0.2875	0.2875	Ties	Ties	Ties	Ties
3	0.2825	0.2825	0.9311	0.9311	0.0011	0.0011	0.2071	0.2071	Ties	Ties	Ties	Ties
4	0.3346	0.3346	0.9484	0.9484	0.0006	0.0006	0.151	0.151	Ties	Ties	Ties	Ties
5	0.2852	0.2852	0.9464	0.9464	0.0006	0.0006	0.1681	0.1681	Ties	Ties	Ties	Ties
6	0.2652	0.2652	0.9256	0.9256	0.0013	0.0013	0.2246	0.2246	Ties	Ties	Ties	Ties
7	0.2767	0.2767	0.9297	0.9297	0.0012	0.0012	0.2122	0.2122	Ties	Ties	Ties	Ties
									Ties	**Ties**	**Ties**	**Ties**

	Experiment 5											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.3192	0.2079	0.9671	0.8695	0.0001	0.0045	0.055	0.2897	MOST	MOST	MOST	MOST
2	0.3065	0.1953	0.9713	0.8717	0	0.0043	0.0138	0.3095	MOST	MOST	MOST	MOST
3	0.3433	0.2143	0.9721	0.8765	0	0.004	0.0193	0.2845	MOST	MOST	MOST	MOST
4	0.3067	0.2108	0.9625	0.8576	0.0002	0.0054	0.0862	0.2747	MOST	MOST	MOST	MOST
5	0.3592	0.2298	0.9711	0.8708	0.0001	0.0044	0.0331	0.2569	MOST	MOST	MOST	MOST
6	0.2676	0.1832	0.9713	0.873	0	0.0042	0.0112	0.327	MOST	MOST	MOST	MOST
7	0.318	0.2144	0.9657	0.8593	0.0001	0.0052	0.0655	0.2702	MOST	MOST	MOST	MOST
									MOST	**MOST**	**MOST**	**MOST**

	Experiment 6											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.2964	0.2964	0.9725	0.9725	0	0	0.0143	0.0143	Ties	Ties	Ties	Ties
2	0.3333	0.3333	0.9707	0.9707	0.0001	0.0001	0.0287	0.0287	Ties	Ties	Ties	Ties
3	0.3754	0.3754	0.9764	0.9764	0.0001	0.0001	0.0369	0.0369	Ties	Ties	Ties	Ties
4	0.3627	0.3627	0.9719	0.9719	0	0	0.0274	0.0274	Ties	Ties	Ties	Ties
5	0.343	0.343	0.9745	0.9745	0	0	0.0199	0.0199	Ties	Ties	Ties	Ties
6	0.3448	0.3448	0.9715	0.9715	0	0	0.0253	0.0253	Ties	Ties	Ties	Ties
7	0.3744	0.3744	0.9753	0.9753	0	0	0	0	Ties	Ties	Ties	Ties
									Ties	**Ties**	**Ties**	**Ties**

	Experiment 7											
	Performance measure											
	OO		**EE**		**OE**		**EO**		**OO**	**EE**	**OE**	**EO**
Repetition	MOST	AFL	MOST	AFL	MOST	AFL	MOST	AFL				
1	0.29507	0.41584	0.97445	0.97474	0.000118	3.38E-05	0.1002	0.019726	AFL	AFL	AFL	AFL
2	0.37365	0.39474	0.97553	0.96558	1.45E-05	0.000187	0.009546	0.077935	AFL	MOST	MOST	MOST
3	0.2784	0.40097	0.97429	0.97653	0.000137	9.34E-06	0.12455	0.006101	AFL	AFL	AFL	AFL
4	0.25247	0.36092	0.97287	0.97534	0.000142	3.29E-05	0.12426	0.022459	AFL	AFL	AFL	AFL
5	0.17988	0.21964	0.9699	0.97037	0.000187	0.000108	0.16963	0.081183	AFL	AFL	AFL	AFL
6	0.22974	0.3048	0.97122	0.97291	0.000128	2.64E-05	0.10154	0.01762	AFL	AFL	AFL	AFL
7	0.19282	0.23406	0.97028	0.97075	0.000173	8.88E-05	0.15179	0.064826	AFL	AFL	AFL	AFL
									AFL	**AFL**	**AFL**	**AFL**

Appendix XII Raw data for adaptive weighted average experiment set

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
1	AdpWA1	1	0.0347	0.9659	0.0002	0.6142
		2	0.0193	0.9659	0.0001	0.7322
		3	0.0376	0.9667	0.0002	0.6222
		4	0.0417	0.9663	0.0002	0.5801
		5	0.0370	0.9659	0.0002	0.5960
		6	0.0312	0.9660	0.0002	0.6415
	AdpWA2	1	0.0383	0.0001	0.9807	0.0000
		2	0.0383	0.0001	0.9807	0.0000
		3	0.0383	0.0001	0.9807	0.0000
		4	0.0383	0.0001	0.9807	0.0000
		5	0.0383	0.0001	0.9807	0.0000
		6	0.0383	0.0001	0.9807	0.0000
	AdpWA3	1	0.0033	0.9637	0.0000	0.8854
		2	0.0038	0.9643	0.0000	0.8807
		3	0.0055	0.9647	0.0000	0.8568
		4	0.0021	0.9635	0.0000	0.9098
		5	0.0060	0.9642	0.0000	0.8470
		6	0.0024	0.9637	0.0000	0.9049
	AdpWA4	1	0.0395	0.0017	0.9203	0.0000
		2	0.0395	0.0017	0.9203	0.0000
		3	0.0395	0.0017	0.9203	0.0000
		4	0.0395	0.0017	0.9203	0.0000
		5	0.0395	0.0017	0.9203	0.0000
		6	0.0395	0.0017	0.9203	0.0000
	AFL	1	0.0058	0.9629	0.0002	0.7853
		2	0.0064	0.9631	0.0002	0.7911
		3	0.0072	0.9631	0.0002	0.7713
		4	0.0097	0.9633	0.0002	0.7328
		5	0.0062	0.9629	0.0002	0.7754
		6	0.0079	0.9631	0.0002	0.7515

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
2	AdpWA1	1	0.3255	0.9744	0.0001	0.0434
		2	0.2336	0.9701	0.0000	0.0234
		3	0.2883	0.9723	0.0000	0.0216
		4	0.3058	0.9568	0.0003	0.1205
		5	0.2454	0.9629	0.0001	0.0708
		6	0.3016	0.9729	0.0000	0.0247
	AdpWA2	1	0.0383	0.0001	0.9807	0.0000
		2	0.0383	0.0001	0.9807	0.0000
		3	0.0383	0.0001	0.9807	0.0000
		4	0.0383	0.0001	0.9807	0.0000
		5	0.0383	0.0001	0.9807	0.0000
		6	0.0383	0.0001	0.9807	0.0000
	AdpWA3	1	0.1627	0.9715	0.0002	0.2858
		2	0.1249	0.9687	0.0002	0.2890
		3	0.1516	0.9698	0.0002	0.2542
		4	0.1768	0.9704	0.0002	0.2024
		5	0.1094	0.9677	0.0003	0.2940
		6	0.1323	0.9697	0.0002	0.3118
	AdpWA4	1	0.0395	0.0017	0.9203	0.0000
		2	0.0395	0.0017	0.9203	0.0000
		3	0.0395	0.0017	0.9203	0.0000
		4	0.0395	0.0017	0.9203	0.0000
		5	0.0395	0.0017	0.9203	0.0000
		6	0.0395	0.0017	0.9203	0.0000
	AFL	1	0.2650	0.9739	0.0001	0.1368
		2	0.2036	0.9707	0.0002	0.1421
		3	0.2808	0.9726	0.0001	0.0501
		4	0.2535	0.9731	0.0001	0.1304
		5	0.2045	0.9705	0.0002	0.1276
		6	0.2540	0.9729	0.0001	0.1207

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
3	AdpWA1	1	0.2390	0.9727	0.0002	0.1466
		2	0.2421	0.9712	0.0001	0.0677
		3	0.3119	0.9742	0.0001	0.0606
		4	0.2183	0.9708	0.0001	0.1106
		5	0.2663	0.9730	0.0001	0.1022
		6	0.2663	0.9730	0.0001	0.1022
	AdpWA2	1	0.2267	0.8921	0.0030	0.2742
		2	0.1965	0.8780	0.0038	0.3096
		3	0.2829	0.9348	0.0010	0.2004
		4	0.1905	0.8888	0.0031	0.3159
		5	0.2356	0.8881	0.0032	0.2621
		6	0.2274	0.9030	0.0024	0.2722
	AdpWA3	1	0.1282	0.9704	0.0002	0.3487
		2	0.1148	0.9685	0.0003	0.3200
		3	0.1539	0.9711	0.0002	0.2974
		4	0.0884	0.9683	0.0002	0.4166
		5	0.1175	0.9694	0.0002	0.3483
		6	0.1325	0.9701	0.0002	0.3274
	AdpWA4	1	0.2378	0.9311	0.0011	0.2283
		2	0.2305	0.9348	0.0009	0.2208
		3	0.3020	0.9523	0.0004	0.1424
		4	0.2180	0.9361	0.0009	0.2202
		5	0.2727	0.9365	0.0009	0.2008
		6	0.2429	0.9406	0.0007	0.1977
	AFL	1	0.0000	0.9620	0.0000	1.0000
		2	0.0000	0.9620	0.0000	1.0000
		3	0.0000	0.9620	0.0000	1.0000
		4	0.0000	0.9620	0.0000	1.0000
		5	0.0000	0.9620	0.0000	1.0000
		6	0.0000	0.9620	0.0000	1.0000

			Performance measure			
Experiment	**Algorithm**	**Repetition**	**OO**	**EE**	**OE**	**EO**
4	AdpWA1	1	0.2514	0.9716	0.0001	0.0667
		2	0.2700	0.9724	0.0001	0.0649
		3	0.2459	0.9718	0.0001	0.0920
		4	0.2570	0.9715	0.0001	0.0519
		5	0.2244	0.9715	0.0001	0.1270
		6	0.2538	0.9707	0.0000	0.0076
	AdpWA2	1	0.1951	0.8802	0.0037	0.3117
		2	0.2023	0.8846	0.0034	0.3027
		3	0.2000	0.8861	0.0033	0.3055
		4	0.1991	0.8785	0.0038	0.3062
		5	0.2051	0.8869	0.0033	0.2993
		6	0.1949	0.8854	0.0034	0.3117
	AdpWA3	1	0.1294	0.9693	0.0002	0.3038
		2	0.1251	0.9693	0.0002	0.3200
		3	0.1531	0.9692	0.0002	0.2147
		4	0.1436	0.9694	0.0002	0.2615
		5	0.1041	0.9686	0.0002	0.3657
		6	0.1215	0.9676	0.0002	0.2343
	AdpWA4	1	0.2114	0.9260	0.0012	0.2526
		2	0.2468	0.9313	0.0011	0.2239
		3	0.2405	0.9467	0.0005	0.1731
		4	0.2132	0.9254	0.0013	0.2532
		5	0.2244	0.9285	0.0012	0.2404
		6	0.1951	0.9211	0.0014	0.2724
	AFL	1	0.0000	0.9620	0.0000	1.0000
		2	0.0000	0.9620	0.0000	1.0000
		3	0.0006	0.9628	0.0000	0.9494
		4	0.0012	0.9627	0.0001	0.9242
		5	0.0006	0.9628	0.0000	0.9494
		6	0.0000	0.9620	0.0001	0.9590

Appendix XIII Statistical evaluation - Friedman's ranking

		Experiment 1				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OO	1	3	4	1	5	2
	2	3	4	1	5	2
	3	3	4	1	5	2
	4	5	3	1	4	2
	5	3	4	1	5	2
	6	3	4	1	5	2
	Sum of ranks	**20**	**23**	**6**	**29**	**12**

		Experiment 1				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EE	1	5	1	4	2	3
	2	5	1	4	2	3
	3	5	1	4	2	3
	4	5	1	4	2	3
	5	5	1	4	2	3
	6	5	1	4	2	3
	Sum of ranks	**30**	**6**	**24**	**12**	**18**

		Experiment 1				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OE	1	3	1	5	2	4
	2	4	1	5	2	3
	3	4	1	5	2	3
	4	4	1	5	2	3
	5	3	1	5	2	4
	6	4	1	5	2	3
	Sum of ranks	**22**	**6**	**30**	**12**	**20**

		Experiment 1				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EO	1	3	5	1	5	2
	2	3	5	1	5	2
	3	3	5	1	5	2
	4	3	5	1	5	2
	5	3	5	1	5	2
	6	3	5	1	5	2
	Sum of ranks	**18**	**27**	**6**	**27**	**12**

		Experiment 2				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OO	1	5	1	3	2	4
	2	5	1	3	2	4
	3	5	1	3	2	4
	4	5	1	3	2	4
	5	5	1	3	2	4
	6	5	1	3	2	4
	Sum of ranks	**30**	**6**	**18**	**12**	**24**

		Experiment 2				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EE	1	5	1	3	2	4
	2	4	1	3	2	5
	3	4	1	3	2	5
	4	3	1	4	2	5
	5	3	1	4	2	5
	6	5	1	3	2	4
	Sum of ranks	**24**	**6**	**20**	**12**	**28**

		Experiment 2				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OE	1	5	1	3	2	4
	2	5	1	3	2	4
	3	5	1	3	2	4
	4	3	1	4	2	5
	5	5	1	3	2	4
	6	5	1	3	2	4
	Sum of ranks	**28**	**6**	**19**	**12**	**25**

		Experiment 2				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EO	1	3	5	1	5	2
	2	3	5	1	5	2
	3	3	5	1	5	2
	4	3	5	1	5	2
	5	3	5	1	5	2
	6	3	5	1	5	2
	Sum of ranks	**18**	**27**	**6**	**27**	**12**

		Experiment 3				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OO	1	5	3	2	4	1
	2	5	3	2	4	1
	3	5	3	2	4	1
	4	5	3	2	4	1
	5	5	3	2	4	1
	6	5	3	2	4	1
	Sum of ranks	**30**	**18**	**12**	**24**	**6**

		Experiment 3				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EE	1	5	1	4	2	3
	2	5	1	4	2	3
	3	5	1	4	2	3
	4	5	1	4	2	3
	5	5	1	4	2	3
	6	5	1	4	2	3
	Sum of ranks	**30**	**6**	**24**	**12**	**18**

		Experiment 3				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OE	1	3	1	2	4	5
	2	4	1	3	2	5
	3	4	1	3	2	5
	4	4	1	3	2	5
	5	4	1	3	2	5
	6	4	1	3	2	5
	Sum of ranks	**23**	**6**	**17**	**14**	**30**

		Experiment 3				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EO	1	5	4	2	3	1
	2	5	3	2	4	1
	3	5	4	3	2	1
	4	5	3	2	4	1
	5	5	3	2	4	1
	6	5	3	2	4	1
	Sum of ranks	**30**	**20**	**13**	**21**	**6**

		Experiment 4				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OO	1	5	3	2	4	1
	2	5	3	2	4	1
	3	5	3	2	4	1
	4	5	3	2	4	1
	5	5	3	2	4	1
	6	5	3	2	4	1
	Sum of ranks	**30**	**18**	**12**	**24**	**6**

		Experiment 4				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EE	1	5	1	4	2	3
	2	5	1	4	2	3
	3	5	1	4	2	3
	4	5	1	4	2	3
	5	5	1	4	2	3
	6	5	1	4	2	3
	Sum of ranks	**30**	**6**	**24**	**12**	**18**

		Experiment 4				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
OE	1	4	1	3	2	5
	2	4	1	3	2	5
	3	4	1	3	2	5
	4	4	1	3	2	5
	5	4	1	3	2	5
	6	5	1	3	2	4
	Sum of ranks	**25**	**6**	**18**	**12**	**29**

		Experiment 4				
		Sensor fusion algorithm				
Performance measure	**Repetition**	**AdpWA1**	**AdpWA2**	**AdpWA3**	**AdpWA4**	**AFL**
EO	1	5	2	3	4	1
	2	5	3	2	4	1
	3	5	2	3	4	1
	4	5	2	3	4	1
	5	5	3	2	4	1
	6	5	2	4	3	1
	Sum of ranks	**30**	**14**	**17**	**23**	**6**

Appendix XIV Statisticl evaluation - Multiple comparison results

Experiment 1									
OO measure					**EE measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups			Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA4	29	A			AdpWA1	30	A		
AdpWA2	23	A	B		AdpWA3	24	A	B	
AdpWA1	20	A	B	C	AFL	18	A	B	C
AFL	12		B	C	AdpWA4	12		B	C
AdpWA3	6			**C**	AdpWA2	6			**C**
OE measure					**EO measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups			Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA3	30	A			AdpWA2	27	A		
AdpWA1	22	A	B		AdpWA4	27	A		
AFL	20	A	B	C	AdpWA1	18	A	B	
AdpWA4	12		B	C	AFL	12	A	B	
AdpWA2	6			C	AdpWA3	6		B	

Experiment 2									
OO measure					**EE measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups			Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	30	A			AFL	28	A		
AFL	24	A	B		AdpWA1	24	A	B	
AdpWA3	18	A	B	C	AdpWA3	20	A	B	C
AdpWA4	12		B	C	AdpWA4	12		B	C
AdpWA2	6			**C**	AdpWA2	6			**C**
OE measure					**EO measure**				
Sensor fusion algorithm	Sum of ranks	Sub groups			Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	28	A			AdpWA2	27	A		
AFL	25	A	B		AdpWA4	27	A		
AdpWA3	19	A	B	C	AdpWA1	18	A	B	
AdpWA4	12		B	C	AFL	12		B	
AdpWA2	6			C	AdpWA3	6		B	

Experiment 3

OO measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	30	A		
AdpWA4	24	A	B	
AdpWA2	18	A	B	C
AdpWA3	12		B	C
AFL	6			**C**

EE measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	30	A		
AdpWA3	24	A	B	
AFL	18	A	B	C
AdpWA4	12		B	C
AdpWA2	6			**C**

OE measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AFL	30	A		
AdpWA1	23	A	B	
AdpWA3	17	A	B	C
AdpWA4	14		B	C
AdpWA2	6			C

EO measure

Sensor fusion algorithm	Sum of ranks	Sub groups	
AdpWA1	30	A	
AdpWA4	21	A	B
AdpWA2	20	A	B
AdpWA3	13		B
AFL	6		B

Experiment 4

OO measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	30	A		
AdpWA4	24	A	B	
AdpWA2	18	A	B	C
AdpWA3	12		B	C
AFL	6			**C**

EE measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	30	A		
AdpWA3	24	A	B	
AFL	18	A	B	C
AdpWA4	12		B	C
AdpWA2	6			**C**

OE measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AFL	29	A		
AdpWA1	25	A	B	
AdpWA3	18	A	B	C
AdpWA4	12		B	C
AdpWA2	6			C

EO measure

Sensor fusion algorithm	Sum of ranks	Sub groups		
AdpWA1	30	A		
AdpWA4	23	A	B	
AdpWA3	17	A	B	C
AdpWA2	14		B	C
AFL	6			C

Appendix XV Statistical evaluation - Sign test results

	Experiment 1											
	Performance measure											
	OO		**EE**		**OE**		**EO**		**OO**	**EE**	**OE**	**EO**
Repetition	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL				
1	0.0347	0.0058	0.9659	0.9629	0.0002	0.0002	0.6142	0.7853	AdpWA1	AdpWA1	TIES	AdpWA1
2	0.0193	0.0064	0.9659	0.9631	0.0001	0.0002	0.7322	0.7911	AdpWA1	AdpWA1	AdpWA1	AdpWA1
3	0.0376	0.0072	0.9667	0.9631	0.0002	0.0002	0.6222	0.7713	AdpWA1	AdpWA1	TIES	AdpWA1
4	0.0417	0.0097	0.9663	0.9633	0.0002	0.0002	0.5801	0.7328	AdpWA1	AdpWA1	TIES	AdpWA1
5	0.037	0.0062	0.9659	0.9629	0.0002	0.0002	0.596	0.7754	AdpWA1	AdpWA1	TIES	AdpWA1
6	0.0312	0.0079	0.966	0.9631	0.0002	0.0002	0.6415	0.7515	AdpWA1	AdpWA1	TIES	AdpWA1
									AdpWA1	**AdpWA1**	**AdpWA1**	**AdpWA1**

	Experiment 2											
	Performance measure											
	OO		**EE**		**OE**		**EO**		**OO**	**EE**	**OE**	**EO**
Repetition	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL				
1	0.3255	0.265	0.9744	0.9739	0.0001	0.0001	0.0434	0.1368	AdpWA1	AdpWA1	TIES	AdpWA1
2	0.2336	0.2036	0.9701	0.9707	0	0.0002	0.0234	0.1421	AdpWA1	AFL	AdpWA1	AdpWA1
3	0.2883	0.2808	0.9723	0.9726	0	0.0001	0.0216	0.0501	AdpWA1	AFL	AdpWA1	AdpWA1
4	0.3058	0.2535	0.9568	0.9731	0.0003	0.0001	0.1205	0.1304	AdpWA1	AFL	AFL	AdpWA1
5	0.2454	0.2045	0.9629	0.9705	0.0001	0.0002	0.0708	0.1276	AdpWA1	AFL	AdpWA1	AdpWA1
6	0.3016	0.254	0.9729	0.9729	0	0.0001	0.0247	0.1207	AdpWA1	TIES	AdpWA1	AdpWA1
									AdpWA1	AFL	AdpWA1	AdpWA1

	Experiment 3											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL				
1	0.239	0	0.9727	0.962	0.0002	0	0.1466	1	AdpWA1	AdpWA1	AFL	AdpWA1
2	0.2421	0	0.9712	0.962	0.0001	0	0.0677	1	AdpWA1	AdpWA1	AFL	AdpWA1
3	0.3119	0	0.9742	0.962	0.0001	0	0.0606	1	AdpWA1	AdpWA1	AFL	AdpWA1
4	0.2183	0	0.9708	0.962	0.0001	0	0.1106	1	AdpWA1	AdpWA1	AFL	AdpWA1
5	0.2663	0	0.973	0.962	0.0001	0	0.1022	1	AdpWA1	AdpWA1	AFL	AdpWA1
6	0.2663	0	0.973	0.962	0.0001	0	0.1022	1	AdpWA1	AdpWA1	AFL	AdpWA1
									AdpWA1	**AdpWA1**	**AFL**	**AdpWA1**

	Experiment 4											
	Performance measure											
	OO		EE		OE		EO		OO	EE	OE	EO
Repetition	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL	AdpWA1	AFL				
1	0.2514	0	0.9716	0.962	0.0001	0	0.0667	1	AdpWA1	AdpWA1	AFL	AdpWA1
2	0.27	0	0.9724	0.962	0.0001	0	0.0649	1	AdpWA1	AdpWA1	AFL	AdpWA1
3	0.2459	0.0006	0.9718	0.9628	0.0001	0	0.092	0.9494	AdpWA1	AdpWA1	AFL	AdpWA1
4	0.257	0.0012	0.9715	0.9627	0.0001	0.0001	0.0519	0.9242	AdpWA1	AdpWA1	TIES	AdpWA1
5	0.2244	0.0006	0.9715	0.9628	0.0001	0	0.127	0.9494	AdpWA1	AdpWA1	AFL	AdpWA1
6	0.2538	0	0.9707	0.962	0	0.0001	0.0076	0.959	AdpWA1	AdpWA1	AdpWA1	AdpWA1
									AdpWA1	**AdpWA1**	**AFL**	**AdpWA1**

תקציר

עבודה זו עוסקת באלגוריתמים להיתוך מידע מחיישנים לצורך מיפוי סביבת רובוט נייד. על מנת לתפקד בסביבות לא ידועות ולא מובנות רובוט נייד חייב להיות מצויד במספר סוגי חיישנים, על מנת להבין טוב יותר את סביבתו, ולהתגבר על מידע לא מדויק או שגוי המתקבל כאשר החיישנים מתפקדים בצורה פגומה או כושלים. היתוך מידע מחיישנים עוסק בשילוב סינרגטי של מידע המגיע מהחיישנים השונים, במטרה לספק לתת תמונה יותר שלמה ומדויקת על התופעה הנלמדת.
במחקר זה, מערכת קודמת להיתוך מידע מחיישנים הורחבה ושופרה. חיישן פיזי נוסף צורף למערכת, והמערכת הורחבה לצורך הכללת חיישן זה בהתכת המידע. בנוסף,פותח אלגוריתם חדש להיתוך מידע.
מיפוי הסביבה חשוב למספר משימות רובוטיות הכוללות משימות חקר ותכנון מסלול. מודל מפת הרשת הבינארית נפוץ בין שיטות המיפוי ויושם במערכת הקודמת להיתוך המידע. בתזה זו, נעשה שימוש במודל מפת רשת לא-בינארית המציין את מידת הוודאות של כל תא.
לאורך השנים, אלגוריתמים רבים להיתוך מידע לצורך מיפוי סביבת רובוט נייד פותחו ויושמו. רובם דורשים מידע קודם על ביצועי החיישנים או על תנאי הסביבה, מידע שקשה ולפעמים אף בלתי אפשרי למצוא בסביבה לא מובנית. במחקר זה, אלגוריתם אדפטיבי חדש להיתוך מידע פותח ויושם. אלגוריתם זה אינו דורש כל מידע מוקדם, אלא מעריך בצורה מקוונת את ביצועי החיישנים ונותן יותר משקל בתהליך ההיתוך לחיישן שמתפקד יותר טוב. אלגוריתם זה משתמש בהליך שיפור שמטרתו לשפר את המפות שנוצרו על ידי החיישנים השונים. הליך השיפור בודק את שכניו של כל תא ומחליט איזה תא אכן מכיל מכשול ולאיזה תא צריך להתייחס כרעש.
האלגוריתם הוערך בעזרת שיטה סטטיסטית שפותחה בעבר להערכת ביצועי אלגוריתמים שונים לאיחוד מידע ובחירת הטוב מביניהם. השיטה מגדירה את צורת ואופן הניסויים שיש לבצע על מנת לבחון את ביצועי האלגוריתמים בתנאי סביבי שונים. שתי הערכות בוצעו. מטרת ההערכה הראשונה היא לבחון את ביצועי מערכת היתוך המידע המורחבת, ואילו מטרת ההערכה השנייה היא לבחון את ביצועי האלגוריתם החדש להיתוך מידע בהשוואה לאלגוריתמים קודמים שפותחו.

תוצאות ההערכה הראשונה מצביעות על כך שהאלגוריתם בעל הביצועים הטובים ביותר במערכת הקודמת, אלגוריתם לוגיקה עמומה אדפטיבי, הוא גם האלגוריתם בעל הביצועים הטובים ביותר במערכת המורחבת. תוצאות ההערכה השנייה מראות כי להליך השיפור אין כל השפעה על הביצועים וכי האלגוריתם החדש שפותח מספק את הביצועים הטובים ביותר בהשוואה לאלגוריתמים קודמים.

מילות מפתח: היתוך מידע מחיישנים, רובוטים ניידים, אלגוריתמים למיפוי, מפות רשת, אלגוריתמים אדפטיביים, מדדי ביצועי, הערכה סטטיסטית.

העבודה נעשתה בהדרכתה של פרופ׳ יעל אידן

המחלקה להנדסת תעשיה וניהול

הפקולטה למדעי ההנדסה

הערכת אלגוריתמים להיתוך מידע מחיישנים
לצורך מיפוי סביבת רובוט נייד

חיבור זה מהווה חלק מהדרישות לקבלת תואר מגיסטר בהנדסה

קרן קאפח

מנחה: פרופ' יעל אידן

אישור המנחה________________ תאריך____________

אישור יו"ר ועדת תואר שני מחלקתית_____________ תאריך____________

תשס"ז 2007

באר-שבע

הערכת אלגוריתמים להיתוך מידע מחיישנים לצורך מיפוי סביבת רובוט נייד

חיבור זה מהווה חלק מהדרישות לקבלת תואר מגיסטר בהנדסה

קרן קאפח

תשס"ז 2007

באר-שבע