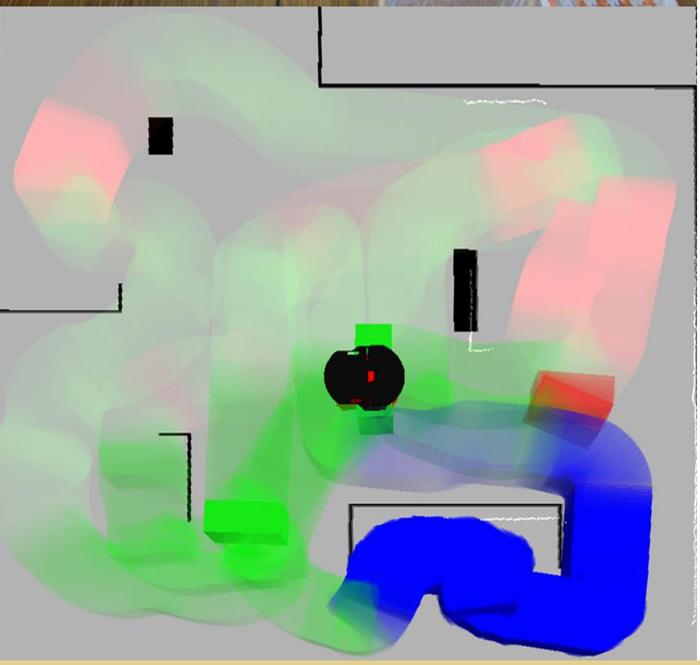

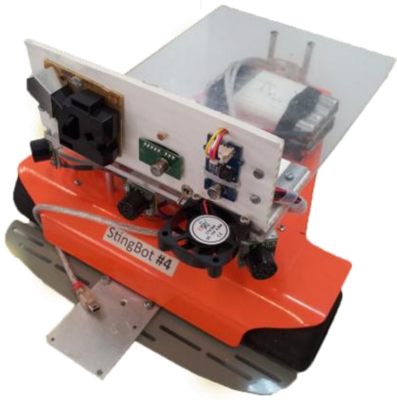

Nuno Filipe Loureiro Ferreira

MRsensing - Environmental Monitoring and Context Recognition with Cooperative Mobile Robots in Catastrophic Incidents

Dissertação

4 de Setembro/2013

· U C ·

Universidade de Coimbra

FCTUC

MRsensing - Environmental Monitoring and Context Recognition with Cooperative Mobile Robots in Catastrophic Incidents

Supervisor:

Prof. Doutor Rui P. Rocha

Co-Supervisor:

Eng. Micael Couceiro

Jury:

Prof. Doutor Rui Pedro Duarte Cortesão

Prof. Doutor Pedro Manuel Gens Azevedo de Matos Faia

Prof. Doutor Rui Paulo Pinto da Rocha

Project report written for the dissertation subject, included in the Electrical and Computer Engineering Course, submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra , September 2013

Agradecimentos

Agradeço antes de tudo aos meus Pais, pelo apoio incondicional neste percurso por vezes bastante difícil, sem eles este objetivo da minha vida nunca teria sido alcançado. Obrigado pelo homem que me fizeram ser hoje. Em seguida agradeço às minhas irmãs Cristina e Daniela, pois elas são também a base da minha educação e um exemplo de força que sempre foi instruído a nós pelos nossos Pais, humildade e persistência definem o nosso nome. Ao Lourenço meu sobrinho, que tantos sorrisos me traz nos momentos difíceis e ao meu cunhado Francisco o meu obrigado também.

Demonstro também a minha gratidão ao Professor Doutor Rui Rocha, pela insistência na perfeição do meu trabalho. Por toda a força, apoio e ensinamentos tenho de agradecer também ao meu Co-Orientador Micael Couceiro. Deixo também o meu obrigado ao colega de trabalho e amigo André Araújo e ao David Portugal pela ajuda importante no trabalho desenvolvido.

À minha namorada e amiga Joana um obrigada pela simpatia, dedicação, apoio, carinho e encorajamento neste percurso.

Obrigado também especial ao Carlos Ramos, Fernando, Pedro, Bruno, André, Gonçalo Ferreira, Sérgio, Gonçalo, João, Tiago, Fertuzinhos, Carlos, Nuno Ramos, Miguel Sousa, e outros amigos pelo companheirismo, apoio e momentos de alegria que me proporcionaram.

Pela lealdade e carinho um obrigado aos meus amigos de quatro patas Boby ,Juny , Pony e Nina.

Abstract

Multi-sensor information fusion theory concerns the environmental perception activities to combine data from multiple sensory resources. Humans, as any other animals, gather information from the environment around them using different biological sensors. Combining them allows structuring the decisions and actions when interacting with the environment. Under disaster conditions, effective mult-robot information sensor fusion can yield a better situation awareness to support the collective decision-making. Mobile robots can gather information from the environment by combining data from different sensors as a way to organize decisions and augment human perception. The is especially useful to retrieve contextual environmental information in catastrophic incidents where human perception may be limited (e.g., lack of visibility). To that end, this work proposes a specific configuration of sensors assembled in a mobile robot, which can be used as a proof of concept to measure important environmental variables in an urban search and rescue (USAR) mission, such as toxic gas density, temperature gradient and smoke particles density. This data is processed through a support vector machine classifier with the purpose of detecting relevant contexts in the course of the mission. The outcome provided by the experiments conducted with TraxBot and Pioneer-3DX robots under the Robot Operating System framework opens the door for new multi-robot applications on USAR scenarios. This work was developed within the CHOPIN research project[1] which aims at exploiting the cooperation between human and robotic teams in catastrophic accidents.

Key Words: Sensor Fusion, Information Fusion, Multi-Robot System, Optimization, Classification, Support Vector Machine, Urban Search and Rescue, Embedded System.

[1]http://chopin.isr.uc.pt/

Resumo

O tema da fusão sensorial abrange a perceção ambiental para combinar dados de vários recursos naturais. Os seres humanos, como todos os outros animais, recolhem informações do seu redor, utilizando diferentes sensores biológicos. Combinando-se informação dos diferentes sensores é possível estruturar decisões e ações ao interagir com o meio ambiente. Sob condições de desastres, a fusão sensorial de informação eficaz proveniente de múltiplos robôs pode levar a um melhor reconhecimento da situação para a tomada de decisão coletiva. Os robôs móveis podem extrair informações do ambiente através da combinação de dados de diferentes sensores, como forma de organizar as decisões e aumentar a perceção humana. Isto é especialmente útil para obter informações de contexto ambientais em cenários de catástrofe, onde a perceção humana pode ser limitada (por exemplo, a falta de visibilidade). Para este fim, este trabalho propõe uma configuração específica de sensores aplicados num robô móvel, que pode ser usado como prova de conceito para medir variáveis ambientais importantes em missões de busca e salvamento urbano (USAR), tais como a densidade do gás tóxico, gradiente de temperatura e densidade de partículas de fumo. Esta informação é processada através de uma máquina de vetores de suporte com a finalidade de classificar contextos relevantes no decorrer da missão. O resultado fornecido pelas experiências realizadas com os robôs TraxBot e Pioneer 3DX usando a arquitetura Robot Operating System abre a porta para novas aplicações com múltiplos robôs em cenários USAR.

Palavras Chave: Fusão Sensorial, Fusão de Informação, Sistemas Multi-Robô, Optimização, Classificação, Máquina de Vetores de Suporte, Busca e Salvamento, Sistemas Embebidos.

Declaration

The work in this dissertation is based on research carried out at the Mobile Robotics Laboratory of ISR (Institute of Systems and Robotics) in Coimbra, Portugal. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

"Never confuse a single defeat with a final defeat."

F. Scott Fitzgerald

Contents

List of Figures

List of Tables

Notation

A_n Alcohol concentration.

C Normalising constant.

D_n Thermopile output.

$E(w_{ij}^n)$ Sum-squared error function.

$P(x, z)$ Probability distribution.

T_n Number of particles.

$X1$ Contamination.

$X2$ Smoke.

$X3$ Fire.

$X4$ Secure.

$f(x)$ Optimal function.

n Scalar.

out_j Outputs of the neural network's final layer.

w Vector

w_{ij} Network weights.

Chapter 1

Introduction

Nature has found a way to integrate information from multiple sources to a reliable and feature-rich recognition. Such biological systems are able to compensate for the lack of information by combining data obtained from different sensors. For instance, humans combine signals from the body senses, i.e, sight, sound, smell, taste, and touch, with knowledge of the environment, to create and update a dynamic model of the surrounding world. The human way of merging information can inspire the design of artificial sensor fusion systems. They both interact with the environment by perceiving new information that is interpreted based on earlier experiences.

In the traditional method, the information acquired from multiple sensors is processed separately, cutting off the possible connections and dependencies between the acquired information, thus possibly overlooking at significant characteristics from the environment [LM09]. For instance, if a single dust sensor, measuring particles density in the air, is used to detect fire by detecting the presence of smoke, ambiguity and data misinterpretation arise because both dust and smoke are comprised of particles.

As opposed to the traditional method, several computing methods, usually denoted as multi-sensor information fusion methods [HL97], allow to analyse and synthesise information from different nodes. This approach has been widely used for real-time processing, e.g., [LM09, SVP08, PNA11a]. These computational methods appeared with the aim of obtaining a better understanding of some phenomena through the development of artificial perception systems that combine data from different sensors. Such methods involve techniques such as statistical inference, signal and image processing, artificial intelligence,

and information sciences.
Multi-sensor information fusion is a process of information integration, merging data from difrent sources with differing conceptual and contextual representations [HL97]. This process fosters the decision-making and estimation to accomplish a certain mission based on perceptual information about the environment. As in many other research areas, multi-sensor information finds a wide application in robotics, namely for object recognition, localisation and mapping, and environmental monitoring. The fusion of multi-sensory information plays an important role in mobile robot perception over real-world. Robot perception requires a system architecture support that cannot be found in simpler robot systems [SST86]. In most cases, the integration of multi-sensory fusion in mobile robots is devoted to navigation, visual recognition and monitoring. This work focuses on the use of multi-sensor information fusion for environmental monitoring and context recognition with cooperative robots in urban catastrophic incidents, namely in urban search and rescue (USAR) missions. The use of different sensors may largely improve the performance of the overall system by providing consistent environmental contextual information, with the following key advantages: redundancy, complementarity, timeliness, and cost [MMN00].

1.1 Context and motivation

Mobile robots can be useful in environments that humans cannot tolerate either due to contamination or very high risk of combustion.
The CHOPIN project[1] aims at exploiting the cooperation between human and robotic teams in catastrophic accidents. Multi-sensor fusion can be used on mobile robots teams to search and inspect, so as to prevent catastrophes, and monitor environments in the aftermath [CPR13]. Adopting a multi-sensor information method allows retrieving a variety of information (e.g., temperature) to be fused, as well as achieve an accurate judgment of contextual information (e.g., fire outbreak). The main aim in this dissertation is to use mobile robots and multi-sensor fusion to monitor the environment and detect hazards in the aftermath of an urban catastrophic incident (e.g. a fire).
Mobile robot technologies in complex and unknown unstructured environments, such as

[1] http://chopin.isr.uc.pt/

catastrophic scenarios are still under study [ZLH08, VA96]. Currently, the main control mode of search and rescue (SaR) robotics is the manual operation, also known as tele-operation, in which complete autonomy is never achieved [CM02]. It has been recently agreed in the literature that autonomous mobile robots require multi-sensor fusion to perceive the environmental information [WJL+12]. By having a robotic intelligent sensor agent capable of storing data, processing data and act upon the environment according to the context may improve the experience of information gathering. By doing this, first responders can perceive the environment and focus on important parameters related with the task, while avoiding irrelevant data through an efficient use of the group of sensors.

1.2 Objectives

The objectives of this dissertation are:

1. Design and evaluation, within laboratorial experiment, of a multi-sensor embedded system to monitor gas concentration, smoke density and temperature.
2. Design, implementation and evaluation of a supervised classifier to detect relevant contexts in an urban fire.
3. Build a map of relevant variables within an urban fire incident zone, either with a single mobile robot or with multiple cooperative robots.

1.3 Organization

This dissertation is organized in seven chapters. The first chapter introduces the context and motivation, and the objectives of this work.

Chapter 2 is a revision of sensor and multi-sensor information fusion based on some of the most relevant related work in the area.

Classification methods such as Neural Network, Fuzzy logic, Bayesian methods and Support Vector Machines (SVM), are presented and compared in Chapter 3.

Chapter 4 presents the three sensors considered in this project and proposes a specific configuration of this sensors in a mobile robot.

Chapter 5 presents the SVM-Classifier and the testbed built to create the training database. Experimental results with one and multiple robots, with both ofline and online classification, are represented in Chapter 6.

Chapter 7 presents the main conclusions of this dissertation and future work directions.

Chapter 2

Multi-Sensor Information Fusion

2.1 Sensors

A sensor, also known as a transducer, is a device that measures a physical quantity and converts it into an electrical signal which can be presented to an observer (e.g., in a graph) or read by an instrument [Elm02]. Before the advent of microelectronics, sensors used to measure physical quantities, such as temperature, pressure, and flow, were usually coupled directly to a readout device, typically a meter, read by an observer. The sensor converted the physical quantity being measured to a displacement. However, the microprocessor technology introduced the requirement to have an electrical output that could be more readily interfaced to provide unattended measurement and control. Therefore, nowadays, sensors help translating the real world of analog signals and varying voltages into the digital processing realm. Sensors typically convert non-electrical physical, biological or chemical quantities into electrical or optical signals. To be useful, the signal must be measured and transformed to a digital format which can be processed and analysed by processing units (e.g., computers). The information can be either used by a person or an intelligent device, to monitor the activity and take decisions that maintain or change a course of action.

In power plants, automated vehicles, aircraft, and in other complex systems, a large number of sensors are used for monitoring and control [AAM01]. Monitoring helps the operator in performing supervisory control tasks. A monitoring system receives information about the system through sensors and makes it available to the operator. By combining information from many different sources, it is possible to decrease the uncertainty and ambiguity inherent to processing the information from a single sensor source [AAM01]. A large number of sensors measuring many different variables can collectively achieve a high level of accuracy and reliability. Nevertheless, some steps are needed to make a multi-sensor system reliable (see Fig. 2.1). For instance, *redundancy creation* generates multiple values for the variable that is being estimated, thus improving the reliability of the measuring process. *Time-Series State prediction* uses temporal information about the variable estimate for a specified time window to predict the value of the variable being measured at the next sampling point sensor. *Data validation* and *fusion* determine whether the information for the sensor can be trusted, thus associating a degree of belief in this measurement and combining the various redundant estimates to generate a "fused" value. In *Fault detection,* the statistical properties of these residues are then used to detect failed sensors [Elm02].

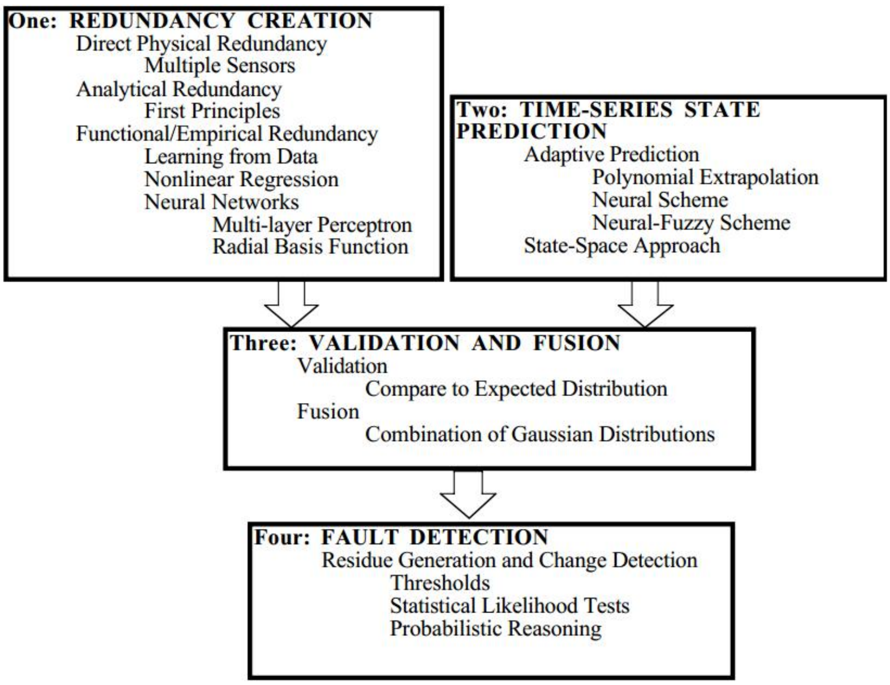

Figure 2.1: Flow chart representation of the four steps to make a multi-sensor system reliable [AAM01].

2.2 Multi-Sensor Information Fusion

Since a single sensor generally can only perceive limited or partial information about the environment, multiple similar and dissimilar sensors are required to provide sufficient local information with different focus and from different viewpoints in an integrated manner. Information from heterogeneous sensors can be combined using data fusion algorithms to obtain observable data [ZLH08]. A multi-sensor system has the vantage to broaden machine perception and enhance awareness of the state of the world compared to what could be acquired with a single sensor system [ALP+11a]. Therefore, multiple sensors are needed in response to the increasingly learning nature of the environment to be sensed. This motivates the emerging interest in research into contextual environmental information in catastrophic incidents (e.g., urban fires[1]). It is also beneficial to avoid overwhelming storage and computational requirements of a single sensor and data rich environment, by controlling the data gathering process such that only the truly necessary data is collected and stored. The simplest task of a sensor management is to choose the optimal sensor parameter values, given one or more sensors, with respect to a given task. This is called active perception, wherein sensors need to be optimally configured for a specific purpose. Multi-sensor management architectures are closely related to the form of the data fusion unit. Typically, there are three alternatives for system structure, namely:

1. Centralized. In a centralized paradigm, the data fusion unit is treated as a central mechanism. It collects information from all different platforms and sensors and decides which tasks must be accomplished by individual sensors. All the commands sent from the fusion center to the respective sensors must be accepted and followed with the proper sensor actions.

2. Decentralized. In a decentralized system, the data is fused locally with a set of local units rather than by a central unit. In this case, every sensor or multi-sensor system can be viewed as an intelligent asset having some degree of autonomy in the decision-making. Sensor coordination is achieved based on communication in the network of robots, in which sensors share locally fused information and cooperate with each other.

---

[1] http://chopin.isr.uc.pt/

3. Hierarchical. This can be regarded as a mixture of centralized and decentralized architectures. In a hierarchical system, there are usually several levels of hierarchy in which the top level functions as the global fusion center and the lowest level consists of several local fusion centers [ZLH08] .

The basic purpose of sensor management is to adapt sensor behavior to dynamic environments. By having limited sensing resources, sensors may not be able to serve all desired tasks and achieve all their associated objectives. Therefore, a reasonable process has to be made. More important tasks should be given higher priority in their competition for resources. The first step for the sensor management system should be to utilize evidences gathered to decide objects of interest and to prioritize which objects to look at in the time following. An interesting scenario requiring sensor coordination is shown in Fig. 2.2 where three autonomous robots equipped with multiple sensors cooperatively explore an area of interest. Nevertheless, to achieve some sort of decision-making, each robot needs to be capable of assessing the contextual information. To that end, for this learning process, classification techniques are needed. Next chapter describes the most well-known used classification methods in the literature.

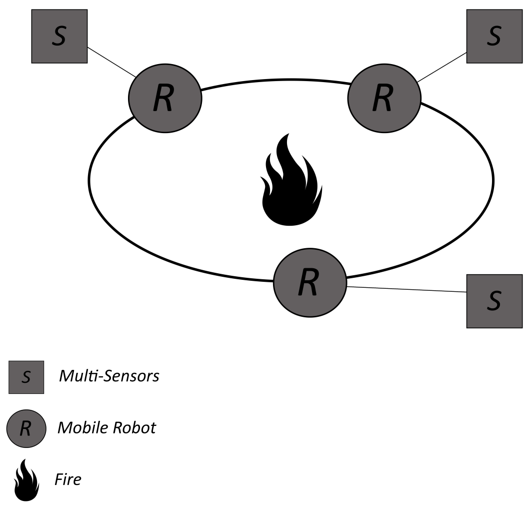

Figure 2.2: A team of cooperative mobile robots wherein each robot is equipped with multiple sensors to observe a fire from a different location.

2.3 Summary

In this chapter, the concept multi-sensor fusion was introduced as a highly important strategy to combine different sensors, so as to achieve results which would be impossible otherwise.

The next Chapter describes the most important classification methods used in multi-sensor information fusion.

Chapter 3

Classification Methods

3.1 Neural Network

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems (i.e. the brain), process information. In brief, ANN may be seen as a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use [Ste96].
ANN are composed of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Neural Networks provide the potential of an alternative information processing paradigm that involve large interconnected networks of processing units. These units, relatively simple and typically non-linear, are connected to each other by communication channels, i.e. connections that carry data.
Artificial Neural Networks have a relationship with statistics. Most neural networks that can learn to generalize effectively from noisy data are similar or identical to statistical methods. Feed forward nets with no hidden layer, including functional-link neural nets and higher-order neural nets, are basically generalized linear models. Probabilistic neural nets are identical to kernel discriminant analysis. Kohonen nets for adaptive vector quantization are very similar to k-means cluster analysis. Hebbian learning is closely related to principal component analysis [Nic03].

3.1.1 Types of Artificial Neural Networks

1. *Supervised Learning*: The network is supplied with a sequence of both input data and desired (target) output data network. It is told precisely by a "teacher" what should be emitted as output. The teacher can, during the learning phase, "tell" the network how well it should perform ("reinforcement learning") or what is the correct behavior ("fully supervised learning") [Gop98].

2. *Self-Organization or Unsupervised Learning*: This is a training scheme in which only the input is given to the network. The network finds out about some of the properties of the data set and learns to reflect these properties in its output. This type of learning presents a biologically more plausible model of learning [Gop98].

3.1.2 Networks based on Feedback and Feedforward connections

Although neural network solutions for predictive analytics, pattern recognition and classification problems can be very different, they are always the result of computations that proceed from the network inputs to the network outputs. The network inputs are referred to as patterns, and outputs are referred to as *classes*.

Frequently, the flow of these computations is in one direction, from the network input patterns to its outputs. Networks with forward-only flow are referred to as feedforward networks. Feedforward neural network is an artificial neural network where connections between the units (a.k.a. perceptrons) do not form a directed cycle. This is different from recurrent neural networks. The feedforward neural network (see Fig. 3.1), was the first and arguably simplest type of artificial neural network devised. In this network, the information flows only in the forward direction, from the input nodes, through the hidden nodes (if any), to the output nodes. There are no cycles or loops in the network. Feedforward networks never contain feedback connections between units. Feedback (recurrent) networks always do (see Fig. 3.2). The presence of feedback connections in a network typically results in a network whose behavior is far more interesting and dynamic than a network composed of feedforward connections alone. Recurrent neural networks, allow data and information to flow in both directions.

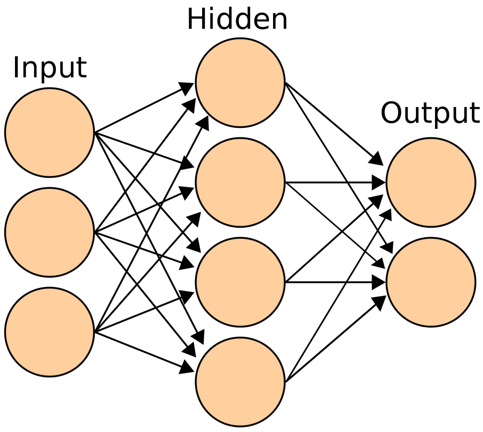

Figure 3.1: A Two-layer, Feed-Forward Network with three Inputs and Two Outputs [KRZ11].

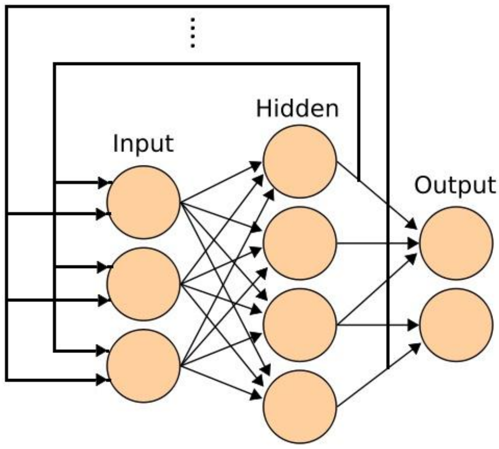

Figure 3.2: A Recurrent Neural Network with three Inputs and Two Outputs [KRZ11].

3.1.3 Methodology: Training, Testing and Validation Datasets and Classification

In the artificial neural networks methodology, the sample data is often subdivided into training, validation, and test sets.

1. *Training set:* A set of examples used for learning to fit the parameters (weights) of the classifier.

2. *Validation set:* A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network.

3. *Test set:* A set of examples used only to assess the performance (generalization) of a fully-specified classifier.

4. *Classification:* Backpropagation algorithm, Fuzzy Adaptive Resonance Theory (ARTMAP).

Multi-layer perception using backpropagation is composed of layers of processing units that are interconnected through weighted connections. The first layer consists of the input vector while the last layer consists of the output vector representing the output class. Intermediate layers, called “hidden” layers, receive the entire input pattern that is modified by the passage through the weighted connections. The hidden layer provides the internal representation of neural pathways. Learning occurs in the perception by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. The network weights $w_{ij}(n)$ are adjusted so that the sum-squared error function is minimized:

$$E(w_{ij}^{n}) = \frac{1}{2}\sum_{p}\sum_{j}(targ_j^p - out_j^{(N)} - out_j^{(N)}(in_i^p))^2,$$

and again we can do this by a series of gradient descent weight updates

$$\nabla w_{kl}^{(m)} = -\eta \frac{dE(w_{ij}^{n})}{dw_{kl}^{(m)}}.$$

Note that it is only the outputs $out_j(N)$ of the final layer that appear in the error function. However, the final layer outputs will depend on all the earlier layers of weights, and the learning algorithm will adjust them all. The learning algorithm automatically adjusts the outputs $out_j(N)$ of the earlier (hidden) layers so that they form appropriate intermediate (hidden) representations.

Fuzzy ARTMAP is a supervised neural network architecture that is based on "Adaptive Resonance Theory", proposed by Stephen Grossberg in 1976 [GRO76,GRO76a]. Adaptive Resonance Theory (ART) encompasses a wide variety of neural networks based explicitly on human information processing and neurophysiology. ART networks are defined algorithmically in terms of detailed differential equations intended as plausible models of biological neurons. In practice, ART networks are implemented using analytical solutions or approximations to these differential equations. Fuzzy ARTMAP, (Fig. 3.3), is based on ART, in which internal control mechanisms create stable recognition categories of optimal size by maximizing code compression, while minimizing predictive error during on-line learning. Fuzzy ARTMAP incorporates fuzzy logic in its ART. It has fuzzy set-theoretic operations instead of binary set-theoretic operations. It learns to classify inputs by a fuzzy set of features, or a pattern of fuzzy membership values between 0 and 1 [Sha10].

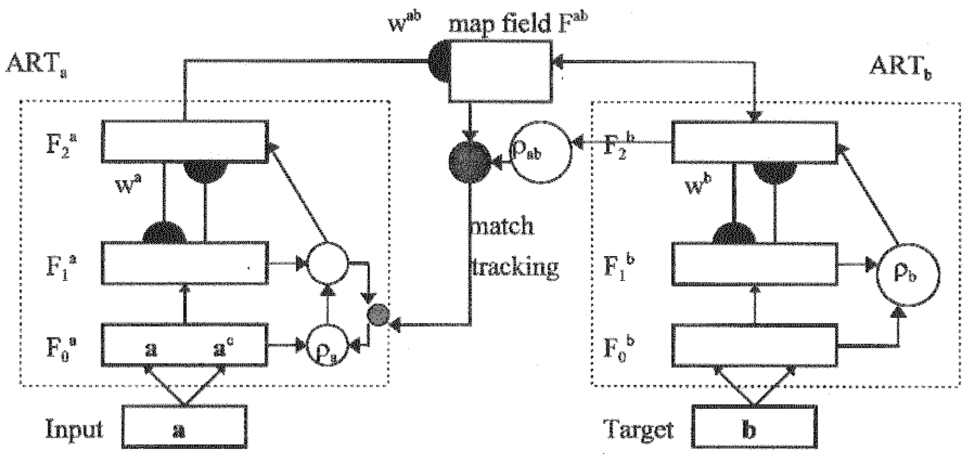

Figure 3.3: Fuzzy ARTMAP architecture [Carpenter et.al.,1992].

3.2 Fuzzy Logic

Fuzzy logic is a form of many-valued logic or probabilistic logic. It deals with reasoning that is approximate rather than fixed and exact [Zad65]. In contrast with traditional logic, they can have varying values, where binary sets have two-valued logic, true or false, fuzzy logic variables may have a truth value that ranges in degree between 0 and 1.

Fuzzy logic has found widespread popularity as a method for representing uncertainty particularly in applications such as supervisory control and high-level data fusion tasks. It provides an ideal tool for inexact reasoning, e.g. control, warning systems and adaptive behaviour [CFL+12a, CMR+12a, CFM12a]. For the combination step in the fusion process, the advantages of fuzzy sets and possibilities rely in the variety of combination operators, which may are able to deal with heterogeneous information (Dubois & Prade, 1985). An advantage of this approach is that it is able to combine heterogeneous information, which is usually the case in multi-source fusion (as in both examples given in the chapter), and to avoid to define a more or less arbitrary and questionable metric between pieces of information, since each piece of information is converted in membership functions or possibility distributions over the same decision space.

A main difference between fuzzy classification and possibilistic classification is that classes are generally considered as fuzzy sets in the first case and as crisp ones in the second case. In the following sections, these two types of modelling are illustrated.

Consider a universal set consisting of the elements x; $X = x$. Consider a proper subset$A \subseteq X$ such that

$$A = \{x \,|\, x \;\; has\, some\, specific\, property\}$$

In conventional logic systems, we can define a membership function µ$A(x)$, also called the characteristic function, which reports if a specific element $x \in Xi$

$$A \rightleftharpoons \mu_A(x) = \Big\{ 1\; if\; x \in A \quad 0\; if\; x \notin A \}$$

In the fuzzy logic literature, this is known as a crispset.

$$A \to \mu A {\to} [0, 1]$$

Composition rules for fuzzy sets follow the composition processes for normal crisp sets, for example

$$A \cap B \rightleftharpoons \mu_{A \cap B(x)} = min[\mu_A(x), \mu_B(x)]$$

$$A \cup B \rightleftharpoons \mu_{A \cup B(x)} = max[\mu_A(x), \mu_B(x)]$$

The normal properties associated with binary logic remains: commutativity, associativity, idempotence, distributivity, De Morgan's law and absorption [KT02]. The only exception is that the law of the excluded middle is no longer true $A \bigcup A = X$, $A \bigcap A = \phi$. Together, these definitions and laws provide a systematic means of reasoning about inexact values. Fuzzy logic and probabilistic logic are mathematically similar – they both have truth values ranging between 0 and 1 – but conceptually distinct, owing to different interpretations. For more information please refer to the interpretations of probability theory [1]. Fuzzy logic corresponds to "degrees of truth", while probabilistic logic corresponds to "probability, likelihood". As these differ, fuzzy logic and probabilistic logic yield different models of the same real-world situations.

A basic application might characterize subranges of a continuous variable. For instance, a temperature measurement for fire detection. Each function maps the same temperature value to a truth value in the 0 to 1 range. These truth values can then be used to determine how the fire should be controlled.

In Fig. 3.4, the meanings of the expressions harmless, warning , and danger are represented by functions mapping a temperature scale. A point on that scale has three "truth values" one for each of the three functions. The vertical line in the image represents a particular temperature that the three arrows (truth values) gauge. Since the red arrow points to zero, this temperature may be interpreted as "not hot". The orange arrow (pointing at 0.2) may describe it as "slightly warm" and the blue arrow (pointing at 0.8) "fairly cold" [BW07].

[1] http://plato.stanford.edu/entries/probability-interpret/

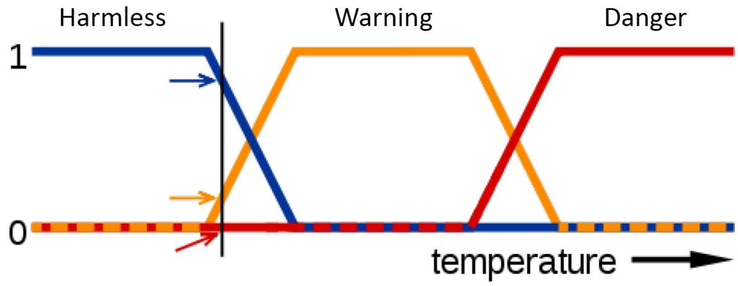

Figure 3.4: Use of fuzzy logic to model temperature.

3.3 Bayesian Models

3.3.1 Bayesian Probability

The Bayesian theory has the possibility to make predictions on future events and provides an embedded scheme for learning [RDA08]. The Bayesian interpretation of probability can be seen as an extension of logic that enables reasoning with propositions whose truth or falsity is uncertain. To evaluate the probability of a hypothesis, the Bayesian probabilist specifies some prior probability, which is then updated in the light of new, relevant data. The use of hierarchical models and marginalization over the values of nuisance parameters. In most cases, the computation is intractable, but good approximations can be obtained using estimation techniques such as Hidden Markov Models (HMMs), Kalman Filters and Particle Filters. Through the sequential use of the Bayes' formula, when more data becomes available after calculating a posterior distribution, the posterior becomes the next prior. For the frequentist, a hypothesis is a proposition which must be either true or false, so that the frequentist probability of a hypothesis is either one or zero. As in Fuzzy Logic, in Bayesian statistics, a probability can be assigned to a hypothesis that can differ from 0 or 1 if the truth value is uncertain.

3.3.2 Sensor Models and Multisensor Bayesian Inference

Bayes' rule provides a means to make inferences about an object or environment of interest described by a state, given an observation z. Bayes' rule requires that the relationship between x and z be encoded as a joint probability or joint probability distribution P(x,z) for discrete and continuous variables respectively. The chain-rule of conditional probabilities can be used to expand a joint probability in two ways:

$$P(x, z) = P(x|z)P(z) = P(z|x)P(x)$$

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)}$$

The value of this result lies in the interpretation of the probabilities P(x|z), P(z|x), and P(x). In this fusion process, the marginal probability P(z) simply serves to normalize the posterior and is not generally computed. The marginal P(z) plays an important role in model valida-tion or data association as it provides a measure of how well the observation is predicted by the prior. The value of Bayes' rule is that it provides a principled means of combining observed information with prior beliefs about the state of the world.

3.3.3 Sensor Models and Multisensor Bayesian Inference

$$P(z_1, ..., z_n|x) = P(z_1|x)...P(z_n|x) = \prod_{i=1}^{n} P(z_i|x)$$

The conditional probability $P(z|x)$ serves the role of a sensor model and can be thought of in two ways. First, in building a sensor model, the probability is constructed by fixing the value of $x = x$ and then asking what probability density $P(z|x = x)$ on z results. Conversely, when this sensor model is used and observations are made, $z = z$ is fixed and a likelihood function $P(z|x)$ on x is inferred. The likelihood function, while not strictly a probability density, models the relative likelihood that different values of x gave rise to the observed value of z. The multisensor form of Bayes' rule requires conditional independence so that

$$P(x|Z^n) = CP(x)\prod_{i=1}^{n} P(z_i|x),$$

$$P(x|Z^k) = \frac{P(z_k|x)P(x|Z^{k-1})}{P(Z_k|Z^{k-1})},$$

where C is a normalising constant. This states that the posterior probability on x given all observations Z^n , is simply proportional to the product of prior probability and individual likelihoods from each information source.

3.3.4 Naive Bayes classifier

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions [PR12a]. A more descriptive term for the underlying probability model would be "independent feature model". The probability model for a classifier is a conditional model:

$$P(C|F_1, ..., F_n).$$

Using Bayes theorem

$$p(C|F_1, ..., F_n) = \frac{p(C)p(F_1, ..., F_n|C)}{p(F_{1,...,}F_n)},$$

is equivalent to

$$posterior = \frac{prior \times likelihood}{evidence},$$

wich can be writen using the chain rule for repeated applications

$$\begin{aligned}
&p(C, F_1, ..., F_n),\\
&\alpha\, p(C)\, p(F_1, ..., F_n|C),\\
&\alpha\, p(C)\, p(F_1|C)\, p(F_2, ..., F_n|C, F_1),\\
&\alpha\, p(C)\, p(F_1|C)\, p(F_2|C, F_1)\, p(F_3, ..., F_n|C, F_1, F_2),\\
&\alpha\, p(C)\, p(F_1|C)\, p(F_2|C, F_1)\, p(F_3|C, F_1, F_2) \,...p(F_n, |C, F_1, F_2, F_3, ..., F_{n-1}),
\end{aligned}$$

so the distribution over the class variable C can be expressed like this

$$p(C|F_1, ..., F_n) = \frac{1}{Z}p(C)\prod_{i=1}^{n} p(F_i|C).$$

3.4 Support Vector Machines

3.4.1 SVM

In machine learning, support vector machines (SVMs, a.k.a. support vector networks) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis [Bur98, LM09, PNA11a, ANO08a]. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted as belong to a category based on which side of the gap they fall on. In addition to performing linear classification, SVMs can efficiently perform non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces (see next section).

SVM is a hybrid technique of statistical and deterministic approaches. This means that to find the best space for classification hypothesis, a probability distribution is determined from the input space. The technique was proposed in the work of Vapnik on the "Principle of Risk Minimization", in the area of statistical learning [HPH+08, Bur98]. The technique is applied in the following way: in the case of linear space, determine the hyperplanes of separation by an optimization problem; in the case of non-linear space, a kernel function is applied and the new space obtained is denominated the feature space. Fig. 3.5 illustrates the application of a kernel in the input space. In the feature space, an hyperplanes is obtained for separation.

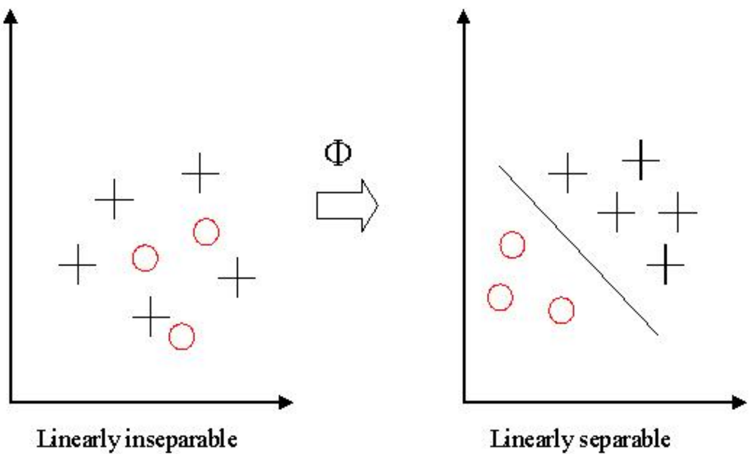

Figure 3.5: Mapping of an input space non-linearly separable for a feature space [MCC04].

In its simplest form, SVMs are linear binary classifiers that assign a given test sample a class from one of the two possible labels. An instance of a data sample to be labeled in the case of remote sensing classification is normally the individual pixel derived from the multi-spectral or hyperspectral image. Elements of the feature vector may also include other discriminative variable measurements based on pixel spatial relationships such as texture. An important generalization aspect of SVMs is that frequently not all the available training examples are used in the description and specification of the separating hyperplane. The subset of points that lie on the margin (called support vectors) are the only ones that define the hyperplane of maximum margin.

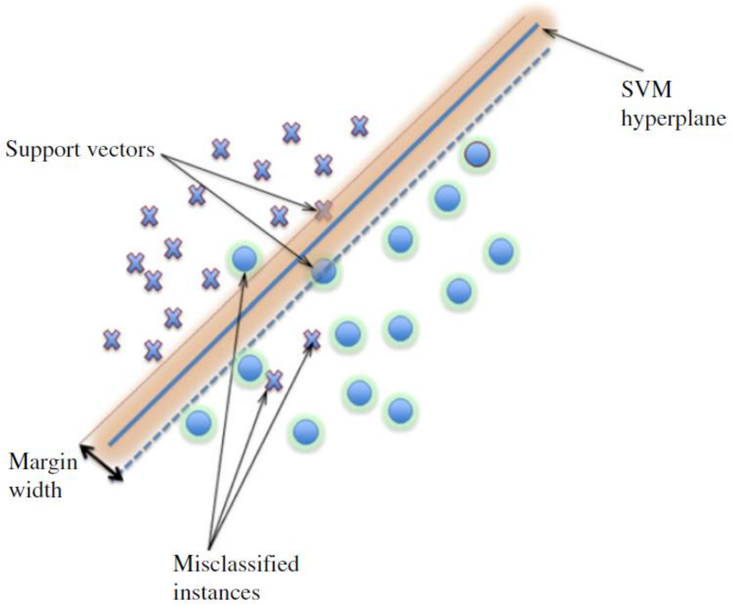

Figure 3.6: Linear SVM example [GJC10].

The implementation of a linear SVM assumes that the multi-spectral feature data are linearly separable in the input space. In practice, data points of different class memberships (clusters) overlap one another. This makes linear separability difficult as the basic linear decision boundaries are often not sufficient to classify patterns with high accuracy. Vapnik–Chervonenkis (VC) dimension and capacity of functions:

$$Test \ \leq Training\,Error + Complexity\,of\ set\,of\ Models.$$

If you take a high capacity set of functions (explain a lot) you get low training error, but you might “overfit”. If you take a very simple set of models, you have low complexity, but you get a high training error.

3.4.2 SVM kernel

The solution of SVM is mapped to the x-domain to a high-dimensional feature space with a nonlinear function φ, followed by a linear regression in high-dimensional feature space, to obtain the effect of original non-linear space regression. Its optimal function is expressed as:

$$f(x) = \omega \times \phi(x) + n,$$

wherein w is a vector and n a scalar. The dimensionality of $\phi(x)$ can be very large, making w hard to represent explicitly in memory,

$$\omega = \sum_{i=1}^{m} \alpha_i \phi(x_i).$$

So, the decision function is:

$$f(x) = \sum_i \alpha_i \phi(x_i).\phi(x) + b = \sum_i \alpha_i K(x_i, x) + b,$$

and the dual dormation

$$min\, P(w,b) = \underbrace{\frac{1}{2}||\sum_{i=1}^{m} \alpha_i \phi(x_i)||^2}_{maximize\, margin} + \underbrace{C \sum_i H_1[y_i f(x_i)]}_{minimize\, training\, error}.$$

Fusion problem of each information fusion node based on SVM theory can be expressed as: for a n-dimensional input parameter x, according to the independent distribution observation samples of k:$(x_{1,}y_1)...(x_k, y_k), X \epsilon R^n$ [LM09]. The model of multi-sensor information fusion based on support vector machine is shown in Fig. 3.7.

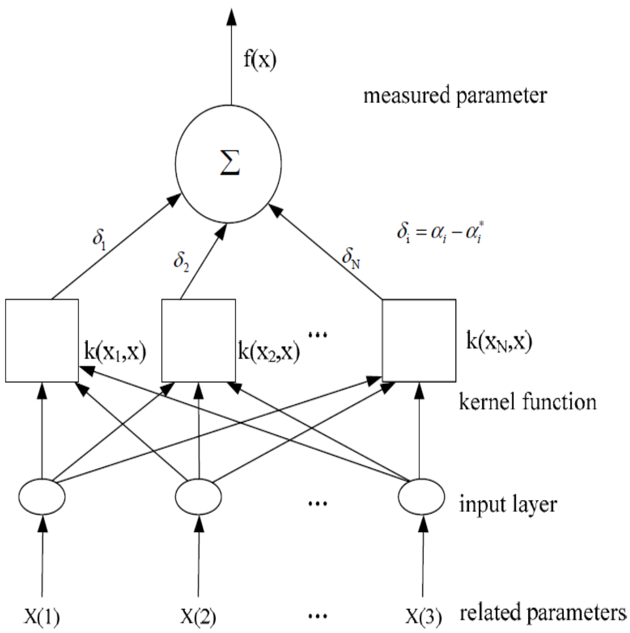

Figure 3.7: The model of multi-sensor information fusion based on SVM [LM09].

3.5 Comparison of Classification Methods

In Sharma et al. [SKV11], a comparative study between SVM, ANN and the Bayesian Classifier for mutagenicity prediction was made. The performance of the classifiers was compared to determine the best model for prediction of mutagenicity for present dataset. The sensitivity of the SVM (69.14%) was found to be better than that of the ANN (40.20%) and the Bayesian classifier (58.44%). The precision of the SVM model (74.9%) is comparatively higher than the one of the ANN (70.00%) and the Bayesian (72.38%) models. Moreover, the SVM predicts 15% and 5.5% less false negatives than ANN and Bayesian classification models respectively. The ANN based model gave the highest specificity value (approx. 81%) as compared to the other two models. However, it stays behind the other two models in terms of sensitivity, accuracy and precision values. The overall accuracy of the SVM was found to be 71.73%, whereas the accuracy of both ANN and Bayesian was 59.72% and 66.14%, respectively, this result is represented in Fig. 3.8.
These statistical studies indicate that the SVM performance is comparatively better than the other two classifiers. In Luz et al. [LCP+12a], a comparative study was carried out between Linear Discriminant Analysis (LDA),Quadratic Discriminant Analysis (QDA), Bayes with Normal (Gaussian) distribution (NV), Naiive Bayes with Kernel Smoothing Density Estimate (NVK) and Least Squares Support Vector Machines with Radial Basis Function Kernel (SVM), for golf putting performance analysis. The five classification methods were compared through the analysis of the confusion matrix and the area under the Receiver Operating Characteristic (ROC) curve . From Figure 3.9, it was possible to confirm that the SVM has the most consistent results.

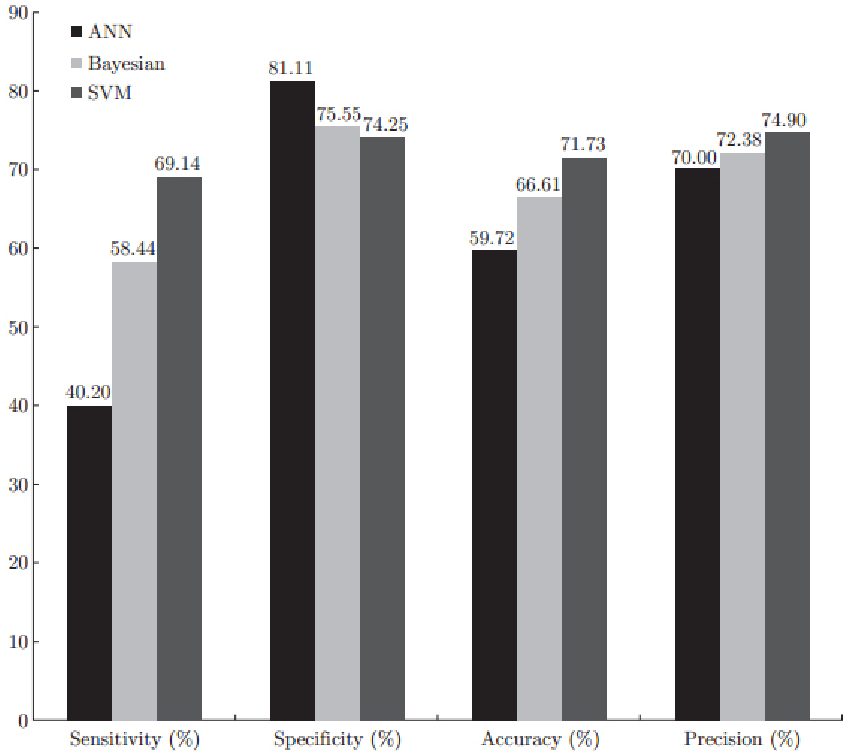

Figure 3.8: Measure of eficiency of the three classifiers [SKV11].

Class	LDA	QDA	NV	NVK	SVM
1	0.619	0.601	0.671	0.680	0.744
2	0.650	0.623	0.692	0.685	0.737
3	0.566	0.582	0.634	0.761	0.734
4	0.507	0.585	0.574	0.675	0.690
5	0.622	0.651	0.692	0.766	0.797
6	0.493	0.602	0.650	0.718	0.745

Figure 3.9: Avarage value of the AUC [LCP+12a].

3.5.1 Discussion and Decision

The most widely used data fusion methods employed in robotics originate in the fields of statistics, estimation and control. However, the application of these methods in robotics has a unique number of features and challenges. In particular, as the autonomy is often the goal, results must be presented and interpreted in a form from which autonomous decisions can be made; for recognition or navigation, for example. Classification is a computationally complex process of supervised learning where the data is separated into different classes on the basis of one or more characteristics inherent in data. In this study, it was possible to see that there are efficient alternatives to heavy probabilistic methods, such as the well-known SVM. SVM is a recent technique suitable for binary classification tasks, which is related to and contains elements of non-parametric applied statistics, neural networks and machine learning. Like classical techniques, SVM also classifies a company as solvent or insolvent according to its score value, which is a function of selected financial ratios. As we can see in works such as [SKV11] and [LCP+12a], the results of other classification models were acceptable, but the SVM was found to be more efficient. Since SVM uses a kernel, it contains a non-linear transformation and no assumptions about the functional form of the transformation are made. Thus, SVM make data linearly separable if necessary. The transformation occurs implicitly on a robust theoretical basis and, as a consequence, human expertise judgment beforehand is not needed (as opposed to Fuzzy and Bayesian models). SVM provides a good out-of-sample generalization, and by choosing an appropriate generalization grade, SVM can be robust, even when the training sample has some bias.

3.6 Summary

In this chapter a survey of the most important classification methods for multi-sensor information fusion was presented. After that, a comparison between the previous methods was carried out and based on these results, the SVM was chosen.

Chapter 4 presents the sensors used in the project, and the respective assembly on mobile robots.

Chapter 4

Multi-Sensor Embedded System

The context of this work involves urban search and rescue (USAR) emergency scenarios, focusing on fire outbreaks occurring in large basement garages. To that end, and as proof-of-concept, three low-cost sensors were chosen. These sensors are presented in the next sections.

4.1 Dust sensor

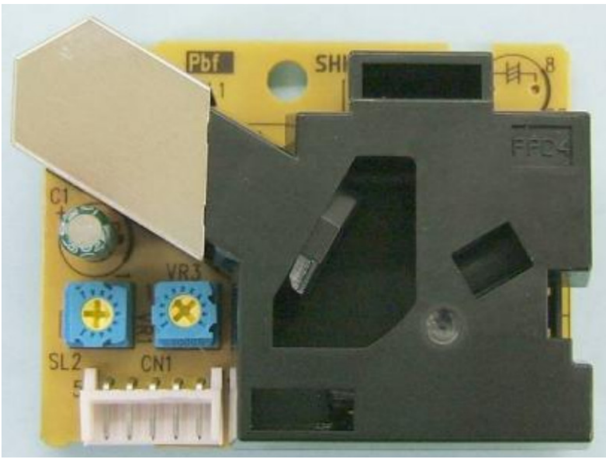

Figure 4.1: Dust sensor model PPD42NS.

The dust sensor model PPD42NS[1] manufactured by Grove is an inexpensive but very sensitive dust sensor. This device works at 5V and measures the amount of small particles

[1]http://www.sca-shinyei.com/pdf/PPD42NS.pdf

like smoke, dust, pollen, bacterias etc, being used for both indoor and outdoor applications.

This dust sensor measures the particulate matter (PM) level in air by counting the Lo Pulse Occupancy time (LPO time) in a given time unit. The LPO time corresponds to the time interval in which the output responds to PM whose size is around 1 micro meter or larger.

Parameter	Value
Detectable particle size	1μm (minimum.)
Detectable range of concentration	0~28,000 pcs/liter (0~8,000pcs/0.01 CF=283ml)
Supply Voltage	DC5V +/- 10% (CN1:Pin1=GND, Pin3=+5V) Ripple Voltage within 30mV
Operating Temperature Range	0~45°C
Operating Humidity Range	95%rh or less (without dew condensation)
Power consumption	90mA
Storage temperature	-30~60°C
Time for stabilization	1 minute after power turned on
Dimensions	59(W) × 45(H) × 22(D) [mm]
Weight	24g(approx.)
Output Method	Negative Logic, Digital output, Hi : over 4.0V(Rev.2) Lo : under 0.7V (As Input impedance : 200kΩ) OP-Amp output, Pull-up resistor : 10kΩ

Table 4.1: Specifications of dust sensor model PPD42NS.

Considering D as the number of particles with at least 1μm diameter, the output of the dust sensor is defined as:

$$0 \leq D \leq 40000$$

4.2 Thermopile array

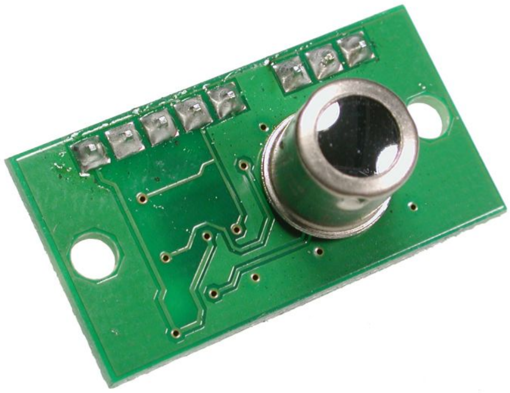

Figure 4.2: Thermopile array model TPA81.

The pyroelectric sensors that are commonly used in burglar alarms and to switch on outside lights, detect infrared in the same wavelength. However, these pyroelectric sensors can only detect a change in heat levels though, therefore they are movement detectors. Although useful in robotics, their applications are limited as they are unable to detect and measure the temperature of a static heat source. Another type of sensor is the thermopile array. These are used in non-contact infra-red thermometers. They have a very wide detection angle or field of view (FOV) of around 100°, and need either shrouding or a lens, or commonly both, to get a more useful FOV of around 12°. Some even have a built-in lens. More recently, sensors with an array of thermopiles built in electronics and a silicon lens have become available. This is the type used in the TPA81 thermopile array. The TPA81[2] (see Fig. 4.2) is a thermopile array that detects infrared light in the 2um-22um wavelength range, which is the wavelength of radiant heat. The TPA81 has an array of eight thermopiles arranged in a row, thus allowing to measure the temperature of 8 adjacent points simultaneously. The TPA81 can also control a servo to pan the module and build up a thermal image, being able to detect a candle flame at a range 2 metres (6ft) without being affected by the ambient light.

[2]http://www.robot-electronics.co.uk/htm/tpa81tech.htm

Power	5V, 5mA
Temperature Range	4° to 100°C (39.2° to 212°F)
Size	43mm x 20mm x 17mm tall (1.69" x 0.79" x 0.67 tall)
Connections	I2C
Field of View (FOV)	41° x 6° (8 pixels of approx. 5° x 6°)
Accuracy (Full FOV) 4° to 10°C (39.2° to 50°F)	+/-3°C (5.4°F)
Accuracy (Full FOV) 11° to 100°C (58.1° to 212°F)	+/-2°C (3.6°F)
Output Data	Outputs - 1 ambient + 8 pixel temperatures
Size	31mm x 18mm (1.22" x 0.71")
Servo Control Resolution	32 steps to 180° rotation

Table 4.2: Specifications of thermopile array model TPA81.

This sensor is characterized by its ability to output an array of 8 elements of 8 bits each. The analog value corresponds directly to the temperature. Hence, one may define the thermopile output as:

$$4^{\text{o}}\text{C} \leq T_i \leq 100^{\text{o}}$$

$$T_i, i = 1, \ldots, 8. \; Ti \; 8 \; Bits \; entry \; T = \; max_i \; v_i$$

4.3 Alcohol sensor

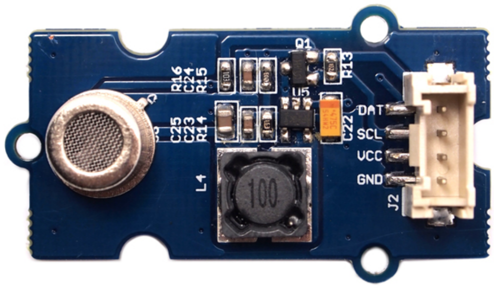

Figure 4.3: Alcohol sensor model MQ303A.

The Grove alcohol[3] sensor is a complete alcohol sensor module for Arduino or Seeeduino. It is built with a MQ303A semiconductor alcohol senso having a good sensitivity and fast response to alcohol. This sensor implements all the necessary circuitry for MQ303A, like power conditioning and heater power supply. This sensor outputs a voltage which inversely proportional to the alcohol concentration in air.

Item	Min	Typical	Max	Unit
Operating Voltage	4.75	5.0	5.25	V
Current	100	120	140	mA
Detection Gas	Alcohol			-
Detectable Concentration	20-1000			ppm

Table 4.3: Specifications of alcohol sensor model MQ303A.

This sensor has the feature to output a voltage A which is inversely proportional to the alcohol concentration in the air:

$$0 \leq A \leq 700\ mv$$

[3]http://www.seeedstudio.com/depot/images/product/MQ303A.pdf

4.4 Sensors Assembling in a Mobile Robot

The set of sensors presented in the previous sections were assembled in a Pioneer-3DX [20] and in a TraxBot [ZSS11] mobile robot.

The Pioneer-3DX (see Fig. 4.5 on the left) is a well-known robotic platform for research and education from ActivMedia. The robot is a robust differential drive platform with 8 sonars in a ring disposition, a high-performance on-board microcontroller based on a 32-bit Renesas SH2-7144 RISC microprocessor, offering great reliability and easiness of use. The Traxbot (see Fig. 4.5 on the right) is a small differential Arduino-based mobile platform, developed in our laboratory. As the Pioneer-3DX, this platform is fully integrated in the open-source Robot Operating System (ROS) framework [Qui09] and is capable of supporting a netbook on top of it [APC+13a]. Therefore, both platforms were extended with netbooks using Ubuntu 11.10 operating system and the ROS framework with Fuerte[4] version on top of them. To explore the scenario, the robots were teleoperated using a wiimote[5] ROS node with the Wii remote controller.

The three sensors were assembled in an aluminium support mounted in the front of the robots (see Fig. 4.5). This provides a better analysis by benefiting from the natural air flow generated by the robots' movements during the scenario exploration. Moreover, this configuration took into consideration a better horizontal positioning of the field of view for the thermopile array sensor. To preprocess the received data from the sensors, an Arduino Uno board embedded within both platforms was used. The main features of the driver developed for the three sensors is summarized in Fig 4.4.

[4] http://ros.org/wiki/fuerte

[5] http://www.ros.org/wiki/wiimote

```
Algorithm 1. MrSensing Arduino Driver
void sensorSetup()
        set 16bit counter for measure the wide of sensor ;
        Set clock 1024/16MHz,unit is 6.4us;
        enable capture interrupt and overflow interrupt;
        join i2c bus (address optional for master) ;

        DUST PIN INPUT
        set the heaterSelPin as digital output.;
        fanpin OUTPUT;
        when heaterSelPin is set, heater is switched off;
int MRsensing::getDustSensor(){
        return quantity;
void MRsensing::getThermopileSensor(int thermopile_tab[]) {
        begin Transmission TPA81ADDR ;
        Wait for incoming idx thermopile frame;
        receive a byte as character ;
int MRsensing::getAlcoholSensor()
        switch on the heater of Alcohol sensor;
        read the analog value;
```

Figure 4.4: Sensors Arduino driver.

The dust sensor was connected to a digital port, the alcohol sensor to an analogic port and the thermopile array sensor via Inter-Integrated Circuit (I2C) Arduino ports. The data exchanged between the Arduino board and the netbooks was handled using a ROS driver developed in our previous work through serial communication [APC+13a].

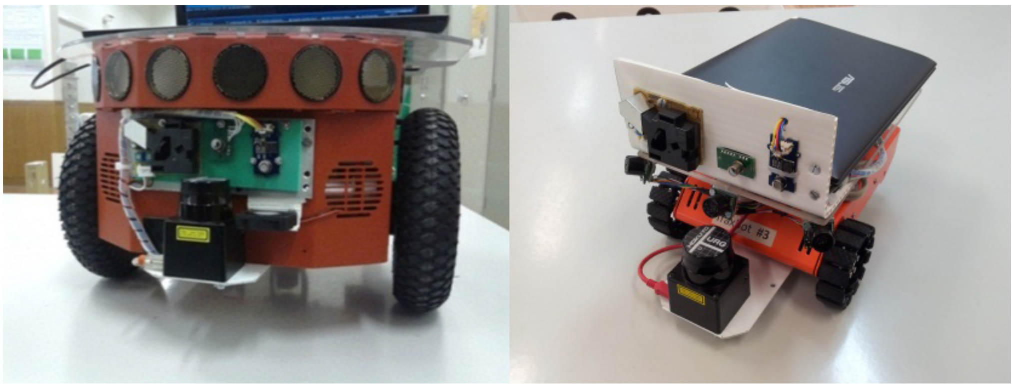

Figure 4.5: The Pioneer 3-DX equipped with the set of sensors (left) and the TraxBot equipped with a similar set of sensors (right).

4.5 Summary

This chapter presented the three sensors used in this dissertation project and their assembly in the two mobile robots. The next chapter presents the SVM classifier designed to detect contexts with robots during a search and rescue mission in an urban incident.

Chapter 5

SVM-based Classification and Context Recognition

5.1 Training database

To minimize undesired external contamination during the training process of the SVM, an experimental multi-sensor testbed platform setup was built (Fig. 5.1). This testbed was designed as an isolated and controlled environment. The testbed presented on Fig.3a is based on a sealed glass aquarium that was transformed to create air flow inside the test area with the integration of two 120 mm fans fixed on the top of aquarium: one for air inflow and another for air outflow. Clean or contaminated controlled air flow samples were introduced within the testbed to measure all achievable range of classes.

An additional fan was afterwards equipped near the alcohol sensor for a faster settling time of the readings (Fig.3 b). An Arduino Uno board with embedded Atmel 328 microcontroller was used to preprocess the output data from the sensors. Afterwards, the data was sent through a serial connection to a computer using Robot Operating System (ROS) [Qui09] taking into account the future use of the classifier *ml_classifier*[1] in the real experiments.

[1]http://www.ros.org/wiki/ml_classifiers

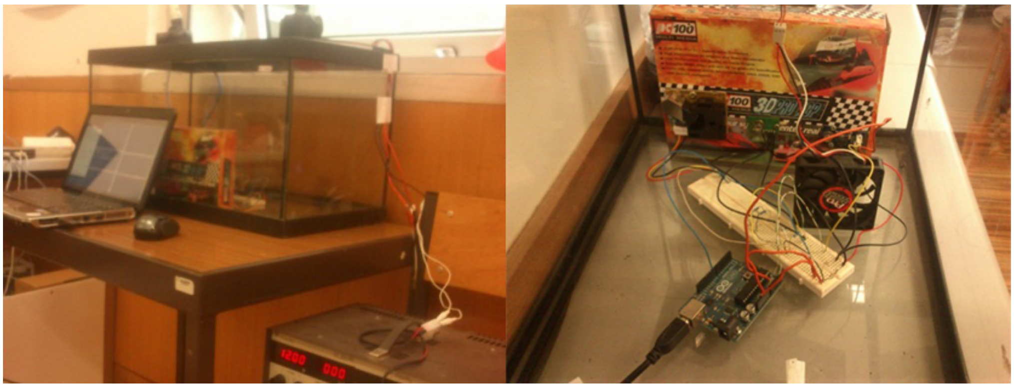

Figure 5.1: Experimental setup for training database. **a)** Testbed; **b)** Acquisition and pre-processig electronic setup.

5.1.1 Training database - Creation

In this project, several preliminary tests under different conditions were carried out for acquisition of the training data. The data returned from the sensors was acquired as:

$$X = \begin{bmatrix} T_1 & D_1 & A_1 \\ & \vdots & \\ T_n & D_n & A_n \end{bmatrix}$$

wherein the number of rows n represents the number of acquired samples, i.e., trials. An example of the acquired output are presented in Table 5.1.

T_n	D_n	A_n
20	110	570
21	110	575
20	110	578
21	110	581
21	110	582

Table 5.1: Output acquired from the sensors. Column 1- TPA81 Thermopile Array (T_n), Column 2- Dust sensor Model PPD42NS (D_n), Column 3- Alcohol Sensor (A_n).

The LS-SVMlab Toolbox[2] for Matlab was used for the initial training and learning based on the data acquired from the sensors. This was a preliminary step to evaluate the

[2]http://www.esat.kuleuven.be/sista/lssvmlab/

chosen classes and the reliability of the sensors. All preliminary experiments were carried out on the setup presented in Fig. 5.1 dividing the space into four distinct typical classes from USAR applications:

- 1. **Contamination (X1)** Air contamination means that alcohol sensor is above the 500mv. Contamination can be caused by gas, petrol, or some kind of alcohol container.
- 2. **Smoke (X2)** Smoke is detected for an output of the dust sensor above 20.000 particles, with at least 1 µm diameter in the read-ing area.
- 3. **Fire (X3)** Fire needs information from the dust sensor, the thermo-pile sensor, and the alcohol sensor. The dust sensor allows detecting smoke, the thermopile sensor the temperature gradient, and the alcohol sensor the type of fire (e.g., fire emanating from a chemical explosion).

X1, X2, X3 are matrices with the test results for the dif-ferent training cases. At least, a final class may be defined to assess the safe situation:

- 4. **Secure (X4)** This class was introduced to minimize the error of the classifier.

This class was introduced to minimize the error of the classifier.

$$X = \begin{bmatrix} S1_{x1} & S2_{x1} & S3_{x1} \\ S1_{x2} & S2_{x2} & S3_{x2} \\ S1_{x3} & S2_{x3} & S3_{x3} \end{bmatrix} Y = [Class]$$

For classification purposes, the on-the-fly data (i.e., testing data) is represented as:

$$X = \begin{bmatrix} T_{t1} & D_{t1} & A_{t1} \\ & \vdots & \\ T_{tn} & D_{tn} & A_{tn} \end{bmatrix}$$

Table 5.2 shows the output variable. Every sample has a class matching from the training database, represented by the numbers 1, 2, 3 and 4 according to $X1, X2, X3$ and $X4$ previously described. After adding 20% noise to the training data $X_i, i.e., X_i^t = 1.2 \times X_i$, with $i = 1, 2, 3$, one may observe in Table 5.3 that the SVM is still able to accurately identify each class.

Sample	T_n	D_n	A_n
899	23	0	642
900	22	0	642
901	24	26218	651
902	23	26218	653

Table 5.2: Training data: Samples 899 and 900 represent contamination training using alcohol while samples 901 and 902 were retrieved using smoke training with paper.

Sample	*Class*
899	1
900	1
901	2
902	2

Table 5.3: Output Classification matches the Training data from table 5.2.

In Table 5.4, the noise was incremented by 30% to the training data $X_i, i.e., X_i^t = 1.3 \times X_i, with i = 1, 2, 3$. It is now possible to observe a classification error in 949 and 950 samples, which the SVM incorrectly classifies class 2 by class 3 in some situations Table 5.5.

Sample	T_n	D_n	A_n
947	22	21402	610
948	23	21402	608
949	24	21402	607
950	23	21402	604
951	22	21402	602

Table 5.4: Trainig Data, Samples of Smoke Training.

Sample	*Class*
947	2
948	2
949	3
950	3
951	2

Table 5.5: Output Classification.

Figure 4 illustrates the classification regions based on the two sensors that the SVM classifier judges as the most important, i.e., the ones that presents more independency between themselves. The classifier assigns dust sensor and temperature sensor as the ones with more relevant differences between different classes.

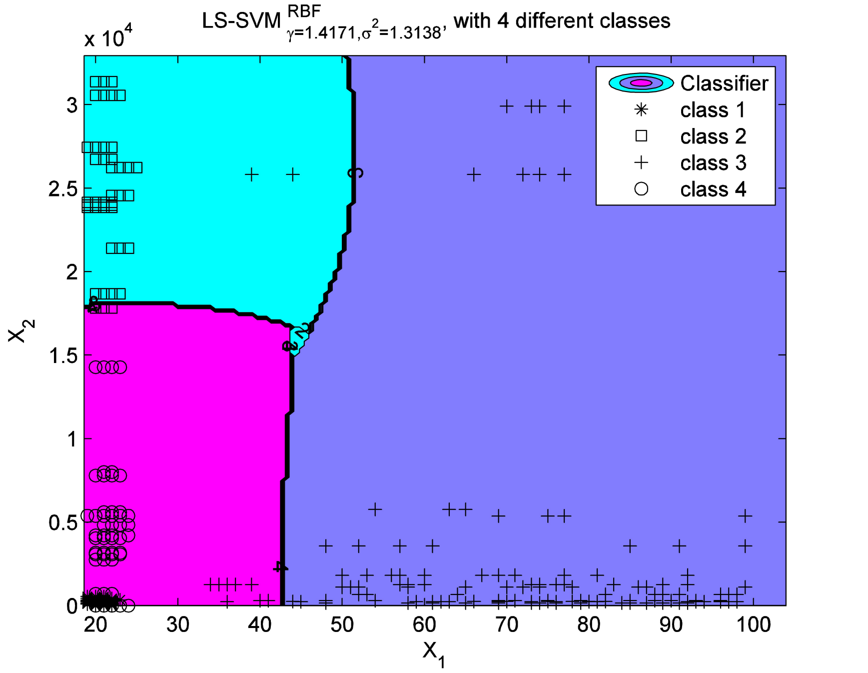

Figure 5.2: Classes representation

5.2 Summary

This chapter describes the creation of the training database for the SVM classifier, this database was important to make the selection of the best values from the multiple extensive tests and for the different SVM classifiers used and created in this project.
Chapter 6 presents the experimental results and discussion, the offline and online classifier are described and results are presented for the proof of the concept.

Chapter 6

Experimental Results and Discussion

6.1 Experiments with a single mobile robot

Some tests with the mobile robot depicted in Fig. 6.1 left were conducted in an indoor laboratory setting, in a scenario with dimensions 4.0×4.6 meters, with several obstacles (Fig. 6.1). A laptop using Ubuntu 11.10 operating system and ROS [Qui09] framework Fuerte version was placed on top of the Pioneer-3DX. To explore the scenario, the robot was teleoperated using a wiimote ROS node with the Wii remote controller.

6.1.1 Offline classification

Two points of interest were introduced in the experimental arena to simulate critical conditions for the classification. More specifically, the gas air contamination was simulated with alcohol and petrol inside containers (Fig. 6.1a), while the fire outbreak (Fig. 6.1b) was emulated using a 500 watts spotlight ideal to produce heat. The SVM classifier works offline after the tests with the data recorded from the set of sensors to detect the different classes. During the experiments, it was possible to match the different classes throughout the several trials with minor misclassification errors. The classes in the SVM Library are represented with circles of different colors, namely, red for fire, green for smoke or dust, and blue for air contamination.

Figure 6.1: Real scenario with two point of interest for SVM classification. **a)** Contamination using alcohol and petrol; **b)** Fire outbreak emulated using a 500 watts spotlight.

Comparing Fig. 6.1 from the real scenario configuration and the output from the reading and classification in Fig. 6.2, we can observe that the classification output matches the real scenario. We can observe that the fire class (red circles) is early mapped without any misclassification error. This was predictable due to the high sensing capability of the thermopile array sensor of up to a distance of two meters. The spread of the contamination (blue circles) caused by the natural air flow along the arena till the end point is also easily detectable by the sensors.

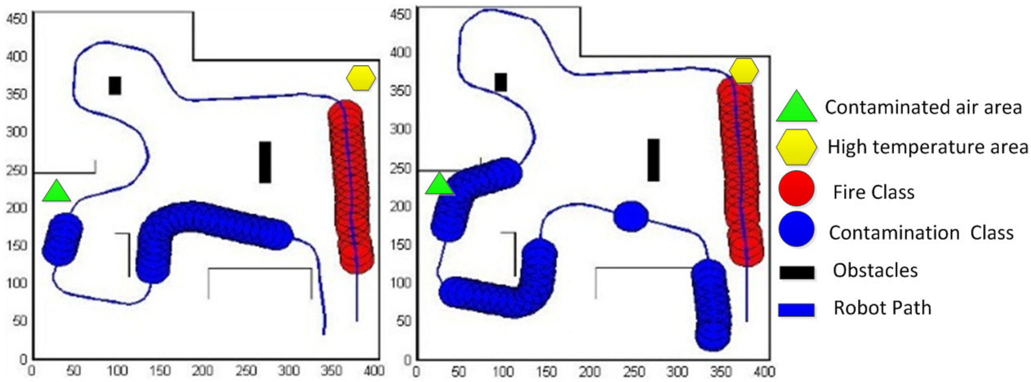

Figure 6.2: Two classificaton maps during the experimental tests.

6.1.2 Online classification

Three points of interest were added in the experimental arena to simulate the necessary critical conditions for classification purposes. More specifically, the fire outbreak (Fig.

6.3a) was emulated using a 500 watts spotlight, ideal to produce heat, while the gas air contamination was simulated by inserting alcohol in an enclosed region within the scenario (Fig. 6.3b). Particles insertion for the assessment of the smoke class was not considered due to environmental constraints associated with the laboratory.

Figure 6.3: Real scenario with three point of interest for SVM classification. a) Fire outbreak emulated using a 500 watts spotlight. b) Contaminated enclosed area with alcohol.

To directly classify the contextual information, the ROS SVM classifier ml_classifier[1], described in Fig. 6.4, was used.

[1]http://www.ros.org/wiki/ml_classifiers

```
import roslib; roslib.load_manifest('ml_classifiers')
import
rospy
from std_msgs.msg import String
from std_msgs.msg import Float64
from geometry_msgs.msg import Vector3
import ml_classifiers.srv
import ml_classifiers.msg

#Wrapper for calls to ROS classifier service and management of classifier
data

class ClassifierWrapper:
        def __init__(self):
        #Set up Classifier service handles

def classificationCallback(data):
        #rospy.loginfo(rospy.get_name() + ": I heard %s" % data.data)
        #gets data from the sensors ROS subriber
        #get training data and compare with the values from the
        sensors

        actual value = [[data.x,data.y,data.z]];
        output classification = classifyPoints('actual value',training
        data);
        publish output classification;

if __name__ == '__main__':

        Publisher mrsensing_classiffication;
        init node mrsensing;
        Subscriber mrsensing dataSensors ;
```

Figure 6.4: Classification algorithm *ml_classifier*.

The SVM classifier works in an online fashion based on the training data previously acquired (section IV). During the exploration mode, the SVM classifier was continuously running so as to detect the different classes. In the process, the acquired data from the set of sensors is streamed, as it can be observed in the rxgraph ROS tool (Fig. 6.5).

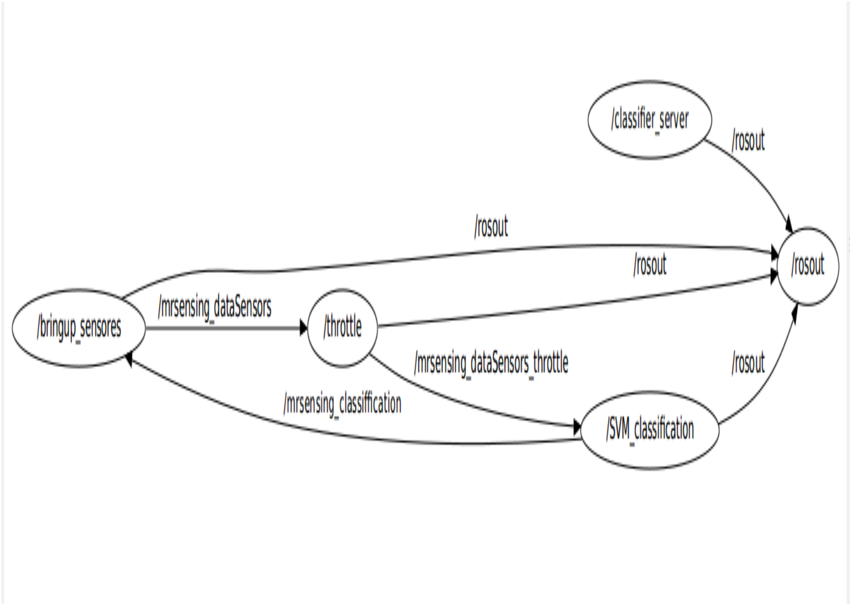

Figure 6.5: ROS topic SVM classification diagram provided by the ROS tool rxgraph.

In these experiments, a distributed ROS core system (a.k.a. multi-master ROS system) for classification was implemented in each robot laptop. A third desktop computer running a ROS core network was added for analysis purposes. The map of the arena was considered to be known a priori for localization purposes by using the Adaptive Monte Carlo Localization[2] (AMCL) algorithm. The AMCL is a probabilistic localization system that uses a particle filter to track the pose of the robot in the map. To that end, both robots were equipped with Hokuyo laser range finders.
The ROS 3D visualization tool rviz[3] was used for an augmented representation of the output classes.

[2]http://www.ros.org/wiki/amcl
[3]http://www.ros.org/wiki/rviz

```
Publisher marker_pub;
Subscriber classification_sub;
Subscriber dataSensors_sub;
Subscriber odom_sub;
Subscriber amcl_sub;

dataSensorsCallback(const geometry_msgs::Vector3::ConstPtr& msg)
  sensors message



odomCallback(const nav_msgs::Odometry::ConstPtr& msg)
  odometry message;
  define visualization Marker CUBE;
  set the frame ID and timestamp.
  marker pose from odometry
  read sensor and classification
  marker with color by cause

int main( int argc, char** argv )
  init classification_markers_robot0;
  marker publisher visualization robot_0
  odom subscribe robot_0 odomCallback
  classification subscribe mrsensing classiffication,classificationCallback;
   dataSensorssubscribe mrsensing dataSensors dataSensorsCallback;
```

Figure 6.6: Algorithm rviz_markers.

Figure 6.7a depicts a virtual representation of the arena in rviz and the virtual model of the robot used in the real test. Figure 6.7b represents the ideal output of the classes on the virtual arena. This ideal representation was retrieved using the setup from Fig. 5.1b, in which the average value from 30 readings coming the set of sensors was considered for each 0.20 $\times$ 0.20 meters cell within the scenario for a total amount of 460 cells.

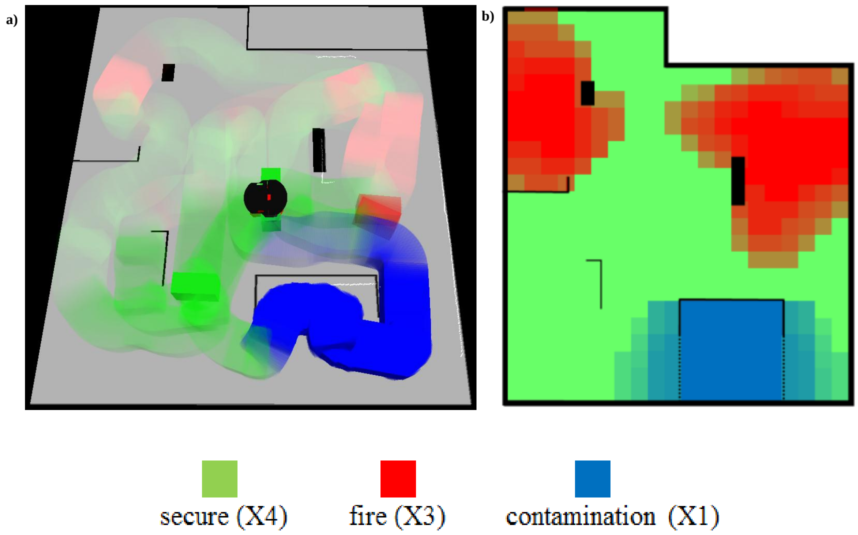

Figure 6.7: **a)** Virtual arena with one robot in rviz. **b)** Ideal representiton of the classification regions.

6.2 Experiments with cooperative mobile robots

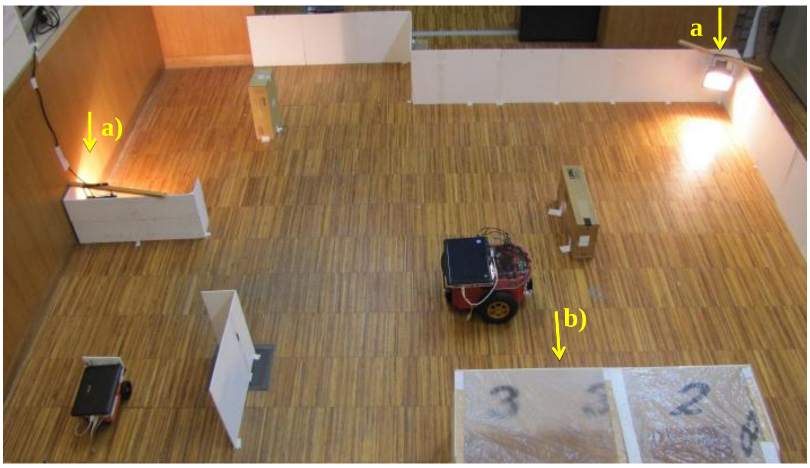

Figure 6.8: Real scenario with three point of interest for SVM classification. a) Fire outbreak emulated using a 500 watts spotlight. b) Contaminated enclosed area with alcohol.

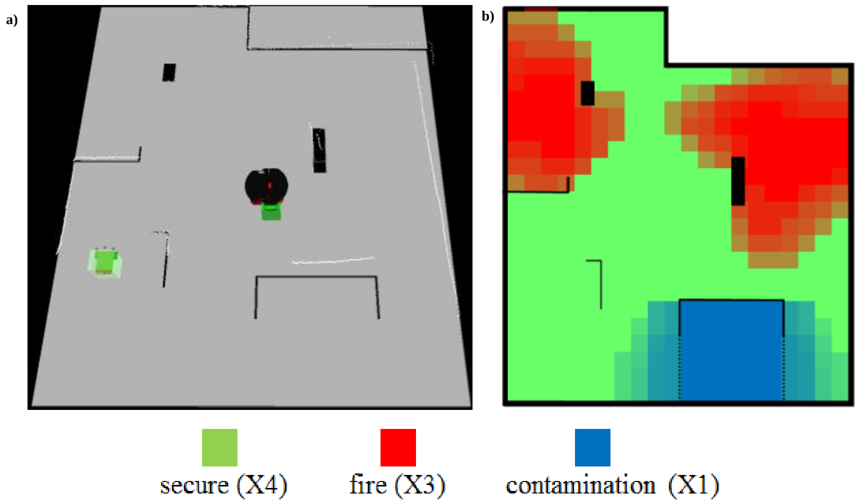

Figure 6.9: **a)** Virtual arena with two robots in rviz. **b)** Ideal representiton of the classification regions.

The rviz representation of each class was achieved by filling the virtual arena with markers of different colors, according with the classification output sent from the ml_classifier. Green cells for secure cells (*X4*), blue cells for contamination cells (*X1*) and red cells for fire cells (*X3*). Then, the intensity of the color was defined to be proportional to the output value from the relevant sensor.

In Fig. 6.10, a comparison from the output of the tests with a single mobile robot and with two robots was considered, wherein one can observe the completeness of the mission after 3 minutes. For instance, in Fig. 6.10b the concentration of the output classes covers almost all the area of the arena, thus getting closer to the ideal representation from Fig. 6.9b.

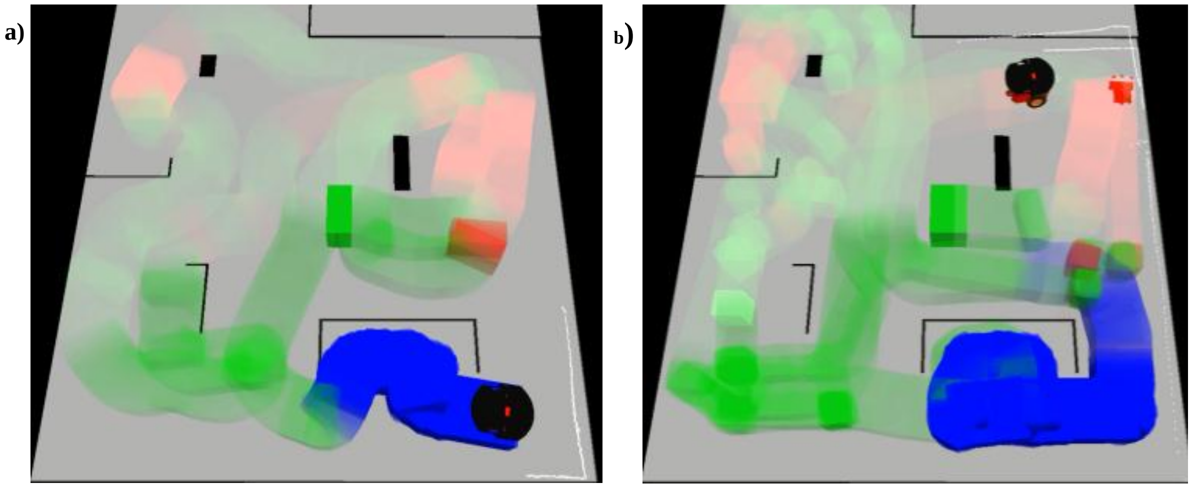

Figure 6.10: Output classification at 3 minutes of the running test with: a) One robot; b) Two robots.

This environmental mapping with one and two robots can be better perceived in the video of the experimental trials[4] reported in a paper submitted recently to the 11th IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR2013) [NMA+13a], which was accepted for presentation. The ROS drivers to control and acquiring data from the sensors set assembled in the StingBot and Traxbot robots. Developed in ISR, University of Coimbra are available online[5].

6.3 Summary

In this chapter, experimental results were presented and discussed, including offline and online SVM classification.
In the next chapter the main conclusions of the work are distilled and future work directions are described.

[4]http://www2.isr.uc.pt/~nunoferreira/videos/SSRR2013/
[5]http://www.ros.org/wiki/mrl_robots_sensors

Chapter 7

Conclusion and future work

This work presented a multi-sensor setup to infer contextual information with a mobile robotic platform and multiple robots platforms within urban catastrophic incidents. The concept multi-sensor fusion was introduced as a highly important strategy to combine different sensors, so as to achieve results which would be impossible otherwise.

Then a survey of the most important classification methods for multi-sensor information fusion was studied and a comparison between the previous methods was carried out and based on these results, the SVM was chosen. A multi-sensor embedded system to monitor gas concentration, smoke density and temperature was design, implemented and tested within laboratorial experiment. After that the training database for the SVM classifier was created, this database was important to make the selection of the best values from the multiple extensive tests and for the different SVM classifiers used and created in this project. The SVM classifiers were tested with one and two mobile robots, and results were represented online in a map of relevant variables.

In the future special attention should be given to the group communication architecture, because robots should be able to share information between themselves and teams of humans (e.g., teams of firefighters) in an efficient way by using the notion of shared context between these. More robots should be equipped with the same set of sensors, and tested in the same conditions.

References and Bibliography

[AAM01] S. Alag, A. M. Agogino, M. Morjaria, *“A methodology for intelligent sensor measurement, validation, fusion, and fault detection for equipment monitoring and diagnostics”,* AI EDAM, Volume 15, Issue 04, September 2001, pp 307-320.

[ALP+11a] H. Aliakbarpour, L. Almeida, P. Menezes, J. Dias, *“Multi-sensor 3D Volumetric Reconstruction Using CUDA”,* Publisher 3D Display Research Center, December 2011.

[ANO08a] R.Araújo, U. Nunes, L. Oliveira, P. Sousa, P. Peixoto , *“Support Vector Machines and Features for Environment Perception in Mobile Robotics*”, Coimbra, Portugal, 2008.

[APC+12] A. Araújo, D. Portugal, M. Couceiro, C. Figueiredo and R. P. Rocha. *“TraxBot: Assembling and Programming of a Mobile Robotic Platform*”, in Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART’2012), Vilamoura, Algarve, Portugal, February 6-8, 2012.

[APC+12a] A. Araújo, D. Portugal, M. Couceiro, C. Figueiredo and R. P. Rocha. *“Small and Compact Mobile Robots Surveying and Comparing Platforms*”, in Proceedings of the 1st International Automatics Conference of the Technical University of Sofia (AUTOMATICS’2012), Sozopol, Bulgaria, June 1-4, 2012.

[APC+13a] A. Araújo, D. Portugal, M. S. Couceiro and R. P. Rocha, *“Integrating Arduino-based Educational Mobile Robots in ROS”*, In Proc, 13th International Conference on Autonomous Robot Systems and Competitions, Lisbon, April 2013.

[Bur98] C. Burges, *"A Tutorial on Support Vector Machines for Pattern Recognition"*, Data Mining and Knowledge Discovery June 1998, Volume 2.

[BW02] L.Biel, P.Wide, Dept. of Technol., Orebro Univ., Sweden, *"An intelligent model approach for combination of sensor information"*, in Proceedings Haptic Virtual Environments and Their Applications, IEEE International Workshop 2002 HAVE, Orebro, Sweden, 2002.

[BW07] T. Born , A. Wright, *"Layered Mode Selection Logic Control with Fuzzy Sensor Fusion Network"*, Proc. SPIE 6561, Unmanned Systems Technology IX, 65610L, Orlando, (May 02, 2007).

[CFL+12a] M. S. Couceiro, C. M. Figueiredo, J. M. A. Luz, M. J. Delorme. "*Zombie Infection Warning System Based on Fuzzy Decision-Making*", In: R. Smith? (ed) Mathematical Modelling of Zombies, University of Ottawa Press, in press, 2012.

[CFM12a] M. S. Couceiro, N. M. F. Ferreira & J. A. T. Machado. "*Hybrid Adaptive Control of a Dragonfly Model*", Journal of Communications in Nonlinear Science and Numerical Simulation, Volume 17, Issue 2, pp. 893-903, Elsevier, 2012.

[CM02] J.Casper, Dr. R.Murphy, *"Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center"*, 33:367–385, 2002.

[CMR+12a] M. S. Couceiro, J. A. T. Machado, R. P. Rocha, N. M. F. Ferreira. "*A fuzzified systematic adjustment of the robotic Darwinian PSO*", Robotics and Autonomous Systems, Vol. 60, Issue 12, pp. 1625-1639, 2012.

[CPR13] Couceiro, M. S., Portugal, D., & Rocha, R. P. (2013). "*A Collective Robotic Architecture in Search and Rescue Scenarios*". Proceedings of the 28th Symposium On Applied Computing, SAC2013, pp. 64-69, March 18–22, Coimbra, Portugal.

[Elm02] Wilfried Elmenreich, *"An Introduction to Sensor Fusion"*, November 19, 2002 Vienna University of Technology, Austria.

[Fen12] M.Fengying, *"Sensor networks-based Monitoring and Fuzzy Information Fusion System for underground Gas disaster"*, 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2012).

[FG08] H. Frigui, L. Gader, *"Context-Dependent Multi-Sensor Fusion for Landmine Detection"*, in Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International,Boston, MA, 7-11 July 2008.

[Gop98] Sucharita Gopal, *"Artificial Neural Networks for Spatial Data Analysis"*, NCGIA Core Curriculum in GIScience, http://www.ncgia.ucsb.edu/giscc/units/u188/u188.html, posted December 22, 1998.

[GRO76] S. Grossberg, *"Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. Biological Cybernetics"*, 23:121-134, 1976.

[GRO76a] S. Grossberg, *"Adaptive pattern recognition and universal recoding: II. Feedback, expectation, olfaction, and illusions. Biological Cybernetics"*, 23:187-202, 1976.

[GJC10] G. Mountrakis, J. Im, C. Ogole, *"Support vector machines in remote sensing: A review"*, in ISPRS Journal of Photogrammetry and Remote Sensing, 2010.

[HPH+08] De-Kun Hu ; Hui Peng ; Ju-Hong Tie ,Software Dept., Chengdu Univ. of Inf. Technol., Chengdu , *"A Multi-Sensor Information Fusion Algorithm based on SVM"*, Apperceiving Computing and Intelligence Analysis, 2008. ICACIA 2008, 13-15 Dec. 2008.

[HL97] D.L. Hall, J. Llinas, *"An introduction to multisensor data fusion"*, Proceedings of the IEEE, 85(1), pp. xx-yy, 1997.

[KKK94] P. E. Keller, R. T. Kouzes, L. J. Kangas, *"Three Neural Network Based Sensor Systems for Environmental Monitoring*", in Electro/94 International. Conference Proceedings. Combined Volumes, 10-12 May 1994.

[KRZ11] Krzysztofcyran, *"Secondary and tertiary structure stability analysis and prediction*", http://lib.bioinfo.pl/courses/view/501.

[KT02] H. Kikuchi, N. Takagi, *"de Morgan Bisemilattice of Fuzzy Truth Value"*, in Proceedings of the 32nd IEEE International Symposium on Multiple-Valued Logic (ISMVLí02).

[LBL+05] Dae-Sik Lee, Sang-Woo Ban, Minho Lee, and Duk-Dong Lee, *"Micro Gas Sensor Array With Neural Network for Recognizing Combustible Leakage Gases*", in IEEE SENSORS JOURNAL, VOL. 5, NO. 3, JUNE 2005.

[LCP+12a] J. Miguel A. Luz , M. Couceiro, D. Portugal, Rui Rocha, H. Araujo, G. Dias, *"Comparison of Classification Methods for Golf Putting Performance Analysis*", Coimbra, Portugal.

[LM09] Z. Li, Y. Ma, *"A new method of multi-sensor information fusion based on SVM*", in Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, Baoding, pp.2-15, July 2009.

[LMA10] S. Larionova, L. Marques, A.T. de Almeida, *"Sensor Fusion for Automated Landmine Detection with a Mobile Robot*", in Emerging Robotics and Sensor Technologies for Humanitarian Demining and Risky Interventions, M. Habib and Y. Baudoin, Woodhead Pub, ISBN 1 84569 786 3, 2010.

[LS07] R.C.Luo, K.L. Su, *"Autonomous Fire-Detection System Using Adaptive Sensory Fusion for Intelligent Security Robot*", in IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 12, NO. 3, JUNE 2007.

[LS10] X.Ling,L.*Shen, "Application of Fuzzy Closeness and Probabilistic Neural Network in Multi-sensor Fusion*", Wuhan,China,10-12 Dec. 2010.

[MCC04] D. McCulloch, An Investigation into Novelty Detection, http://www.enm.bris.ac.uk/teaching/projects/2004_05/dm1654/kernel.html.

[MMN00] A. Martinelli, F. Martinelli, S.Nicosia, P. Valigi, *“Multisensor Fusion For Mobile Robot Positioning and Navigation*”, 2000.

[Nic03] S. P. Niculescu, *“Artificial neural networks and genetic algorithms in QSAR*”, Journal of Molecular Structure: THEOCHEM, Volume 622, Issues 1-2 March 2003.

[NMA+13a] N. L. Ferreira, M. S. Couceiro, A. Araújo and R. P. Rocha “*Multi-Sensor Fusion and Classification with Mobile Robots for Situation Awareness in Urban Search and Rescue using ROS*” in 11th IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR2013). [Accepted].

[Nob10] W.Noble, *“What is a support vector machine?*”, 2006 Nature Publishing Group.

[OCS+05] L. Oliveira, A. Costa, L. Schnitman, J. Souza, *“An Architecture of Sensor Fusion for Spatial Location of Objects in Mobile Robotics*”, Progress in Artificial Intelligence Lecture Notes in Computer Science Volume 3808, 2005, pp 462-473.

[PNA11a] Premebida, C. , Nunes, U. , Araújo, R.,Ludwig, O., *“Evaluation of Boosting-SVM and SRM-SVM Cascade Classifiers in Laser and Vision-based Pedestrian Detection*”, 2011 14th International IEEE Conference on, Washington, DC, 5-7 Oct. 2011.

[POM+10] G. Pavlina, P. Oudea, M. Marisa, J. Nunninka, T. Hoodb, *“A multi-agent systems approach to distributed bayesian information fusion*”, in Information Fusion.

[PR12a] D. Portugal and R. P. Rocha, "*Decision Methods for Distributed Multi-Robot Patrol*", In Proceedings of the 2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR’2012), College Station, Texas, USA, November 5-8, 2012.

[Qui09] M. Quigley, et al. "*ROS: an open-source Robot Operating System*" in Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA 2009), Kobe, Japan, May, 2009.

[RDA08] J. Rett, J. Dias, J.-M. Ahuactzin "Laban Movement Analysis using a Bayesian model and perspective projections" In: Brain, Vision and AI - Edited by Cesare Rossi.

[Sha10] S.Sharma, "*implementation of artificial neural network for odor identification using e-nose*", NCCI 2010 -National Conference on Computational Instrumentation CSIO Chandigarh, INDIA, 19-20 March 2010.

[SKV11] A. Sharma , R. Kumar , P. Varadwaj, A. Ahmad , G. Ashraf, *"A Comparative Study of Support Vector Machine*", Artificial Neural Network and Bayesian Classifierer for Mutagenicity Prediction, Jhalwa, Allahabad, 211012, Uttar Pradesh, India,2011 Jun 14.

[SMC+04] C. Sutton, C. Morrison, P. Cohen, J. Moody, J. Adibi, *"A Bayesian Blackboard for Information Fusion*", Fusion 2004: Seventh International Conference on Information Fusion; Stockholm; Sweden; 28 June-1 July 2004. 2004.

[SST86] Shafer, Steven A.; Stentz, Anthony; and Thorpe, Charles E., *"An architecture for sensor fusion in a mobile robot"* (1986). Robotics Institute. Paper 585. http://repository.cmu.edu/robotics/585.

[Ste96] C. Stergiou, *"What is a Neural Network?"*, Suprise 96 Journal ,Article 1 (cs11) London,1996, http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/cs11/article1.html.

[SVP08] Shishir B. , G. Venayagamoorthy , B.Paudel, *"Embedded Neural Network for Fire Classification Using an Array of Gas Sensors*", SAS 2008 – IEEE Sensors Applications Symposium Atlanta, GA, February 12-14, 2008.

[TTT+03] F. Temurtas, C. Tasaltin, H. Temurtas, N. Yumusak, Z. Ziya Ozturk, *"Fuzzy Logic and Neural Network Applications on Gas Sensor Data: Concentration*

Estimation”, Lecture Notes in Computer Science Volume 2869, 2003, pp 179-186,18th International Symposium, Antalya, Turkey, November 3-5, 2003.

[WJL+12] P. Wang,W. Jiang , X. Li , S. Kang ,J.i Xin, *“Research for Multi-sensor Information Fusion Algorithm of Search and Rescue Robot Based on Embedded Control Network*”, in Journal of computers, vol. 7, no. 5, may 2012.

[VA96] D. Vlachos, J. Avaritsiotis, *“Fuzzy neural networks for gas sensing*”, in Sensors and Actuators B 33 (1996) 77-82 , 157-73 Zographou, Athens, Greece.

[XS02] N. Xiong, P. Svensson, *“Multi-sensor management for information fusion: issues and approaches”*, Elsevier, June 2002.

[YYH+10] Y.Yao,Jing Yang ,C. Huang ,W. Zhu , *“Fire Monitoring System Based on Multi-sensor Information Fusion*”, in Information Engineering and Electronic Commerce (IEEC), 2010 2nd International Symposium on 23-25 July 2010.

[Zad65] L.A.Zadeh, *“Fuzzy Sets**”, Information and Control, 8, 338-353 1965.

[ZLH08] X. Zhao, Q. Luo, B. Han , *“Survey on robot multi-sensor information fusion technology*”, Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on 25-27 June 2008.

[ZMC11] D.Zhang,X, Ma ,A.Chang, *“Design of Gas fire-extinguishing Control panel Based on Multi-sensor Information Fusion”* , in Multimedia Technology (ICMT), 2011 International Conference on 26-28 July 2011

[ZSS11] S. Zaman, W. Slany and G. Steinbauer, "*ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues*", In Proc. of the IEEE Saudi International Electronics, Communications and Photonics Conference, Riad, Saudi-Arabia, 2011.

Annex 1: font Driver Mrsensors

Annex 2: header file Mrsensors

Annex 3: arduino ROS driver

Annex 4: ml_classifier Algorithm

Annex 5: rviz markers class robot0 (traxbot)

Annex 6: rviz markers class robot1 (pioneer)

Annex 7: arduino Pioneer node

Annex 8: frames 1

Annex 9: frames 2

Annex 10: paper for the 11th IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR2013) [submitted]

Annex 1

```cpp
/**********************************************************************
*
* Software License Agreement (BSD License)
*
*  Copyright (c) 2012, ISR University of Coimbra.
*  All rights reserved.
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above
*     copyright notice, this list of conditions and the following
*     disclaimer in the documentation and/or other materials provided
*     with the distribution.
*   * Neither the name of the ISR University of Coimbra nor the names of its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
*  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
*  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
*  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
*  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
*  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
*  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
*  POSSIBILITY OF SUCH DAMAGE.
*
*  Author: Nuno Ferreira on 15/05/2013
*********************************************************************/

#include "MRsensing.h"
#include <Wire.h>



unsigned long time_rise=0;
unsigned long time_fall=0;
unsigned long time_mus=0;
unsigned int ov_counter=0;
int quantity=0;//2011/12/29 change by bruce
float time_sum=0;
float rate=0;

void MRsensing::sensorSetup() {

  //set 16bit counter for measure the wide of sensor
  TCCR1A = 0;
  TCCR1B |=1<<(CS12)|0<<(CS11)|1<<(CS10);//Set clock 1024/16MHz,unit is 6.4us
  TIMSK1 |=1<<(ICIE1)|1<<(TOIE1); //enable capture interrupt and overflow interrupt
  TCNT1 = 0;
```

```
  delay(3000);

  Wire.begin();        // join i2c bus (address optional for master)

  pinMode(DUST_PIN,INPUT);
  pinMode(ALCOHOL_SelPin,OUTPUT);    // set the heaterSelPin as digital output.
  pinMode(fan_pin,OUTPUT);
  digitalWrite(ALCOHOL_SelPin,HIGH); //when heaterSelPin is set, heater is switched off.
  digitalWrite(fan_pin,HIGH);

  sei();//enable interrupt
}


int MRsensing::getLDRsensor() {

  int LDRsensorValue = 0;
  LDRsensorValue = analogRead(LDR_Pin);
  return LDRsensorValue;

}



int MRsensing::getAlcoholSensor() {

  int sensorValue = 0;
  digitalWrite(ALCOHOL_SelPin,LOW);                //switch on the heater of Alcohol sensor
  sensorValue = analogRead(ALCOHOL_InDatPin);       //read the analog value
  sensorValue = 1023 - sensorValue;

  return sensorValue;
}


int MRsensing::getDustSensor(){
  return quantity;
}



void MRsensing::getThermopileSensor(int thermopile_tab[]) {

  int idx=0;

  for (idx=1; idx<=9; idx++) {

    Wire.beginTransmission(TPA81ADDR);
    Wire.write(idx);
    Wire.endTransmission();
    Wire.requestFrom(TPA81ADDR,  1);
    while(Wire.available() < 1) {     // Wait for incoming idx thermopile frame
    }

    thermopile_tab[idx-1]= Wire.read(); // receive a byte as character
  }
```

```

}

int MRsensing::getBearing(){

   byte highByte, lowByte, fine;              // highByte and lowByte store high and low bytes of the bearing and fine stores decimal place of bearing
   char pitch, roll;                          // Stores pitch and roll values of CMPS10, chars are used because they support signed value
   int bearing;                               // Stores full bearing

   Wire.beginTransmission(CMPS10);           //starts communication with CMPS10
   Wire.write(2);                              //Sends the register we wish to start reading from
   Wire.endTransmission();

   Wire.requestFrom(CMPS10, 4);              // Request 4 bytes from CMPS10
   while(Wire.available() < 4);               // Wait for bytes to become available
   highByte = Wire.read();
   lowByte = Wire.read();
   pitch = Wire.read();
   roll = Wire.read();

   bearing = ((highByte<<8)+lowByte)/10;      // Calculate full bearing
   fine = ((highByte<<8)+lowByte)%10;         // Calculate decimal place of bearing

  return bearing;

}

//duty measure
ISR(TIMER1_OVF_vect)
{
    if(ov_counter==7)
        {
            PORTD^=0x40;
            ov_counter=0;
            //Serial.println(time_sum);
            rate=(float)(time_sum/336000);
            if(rate<=8)
                {
                    quantity=rate*562.5;//8 equal 4500 pcs Particle according to the datasheet.
                    }
            else
                quantity=4500+(rate-8)*750;

                        //Serial.print("quantity is :");
                        //Serial.println(quantity);
            //Serial.print("rate is :");
            //Serial.println(rate,8);
            time_sum=0;
            }
    else
        {
            ov_counter++;
            //digitalWrite(6,HIGH);
```

```
                //Serial.println(ov_counter);
                }
        }


ISR(TIMER1_CAPT_vect)
{

        if((PORTB^0x01)==1)
            {
                //time_fall=ICR1;
                time_fall=micros();
                TCCR1B=0x45; //change to rising capture and with 1024 prescaler
                digitalWrite(13,HIGH);
                //TIFR1|=1<<(TOV1);//reset the flag
                }
        else
            {
                time_rise=micros();
                TCCR1B=0x05; //change to negative and with 1024 prescaler
                digitalWrite(13,LOW);
                //TIFR1|=1<<(TOV1);//reset the flag
                if(time_rise>time_fall)
                time_mus=20000+(time_rise-time_fall);//20000 is countervail for program run
                time_sum+=+time_mus;
            }

};

```

Annex 2

```
/**********************************************************************
*
* Software License Agreement (BSD License)
*
*  Copyright (c) 2012, ISR University of Coimbra.
*  All rights reserved.
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above
*     copyright notice, this list of conditions and the following
*     disclaimer in the documentation and/or other materials provided
*     with the distribution.
*   * Neither the name of the ISR University of Coimbra nor the names of its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
*  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
*  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
*  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
*  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
*  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
*  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
*  POSSIBILITY OF SUCH DAMAGE.
*
* Author: Nuno Ferreira on 15/05/2013
*********************************************************************/

#include <EEPROM.h>
#include "Arduino.h"


#define ALCOHOL_InDatPin 0  //Alcohol Sensor DAT Pin is connected to Analog Input Pin 0 (A0)
#define ALCOHOL_SelPin 15  //Alcohol Sensor SEL Pin is connected to Analog Input Pin 1 (A1). In this case it is used as digital ouput. 15 is mapped to A1
#define TPA81ADDR 0x68
#define DUST_PIN 8
#define fan_pin 7
#define CMPS10 0x60
#define LDR_Pin 2      // Analog Input Pin (A2) Light Reading sensor



class MRsensing
{
  public:
  void sensorSetup();
  int getAlcoholSensor();
  int getDustSensor();
```

```
    void getThermopileSensor(int thermopile_tab[]);
    int getBearing();
    int getLDRsensor();

    private:


};
```

Annex 3

```
/*********************************************************************
*
* Software License Agreement (BSD License)
*
*  Copyright (c) 2013, ISR University of Coimbra.
*  All rights reserved.
*
*  Redistribution and use in source and binary forms, with or without
*  modification, are permitted provided that the following conditions
*  are met:
*
*   * Redistributions of source code must retain the above copyright
*     notice, this list of conditions and the following disclaimer.
*   * Redistributions in binary form must reproduce the above
*     copyright notice, this list of conditions and the following
*     disclaimer in the documentation and/or other materials provided
*     with the distribution.
*   * Neither the name of the ISR University of Coimbra nor the names of its
*     contributors may be used to endorse or promote products derived
*     from this software without specific prior written permission.
*
*  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
*  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
*  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
*  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
*  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
*  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
*  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
*  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
*  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
*  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
*  POSSIBILITY OF SUCH DAMAGE.
*
* Modified by:   Andre Araujo & David Portugal on 04/01/2013
* Modified by:     Nuno Ferreira & João Santos on 15/05/2013
* Version: Traxbot_Stingbot_DriverROS_v1
*********************************************************************/

// Arduino libraries
#include <EEPROM.h>
#include <stdlib.h>
#include <Wire.h>
#include <string.h>
#include <math.h>

// Traxbot robot & Omni3MD lib
#include "Robot.h"

// RobotSerial Communication lib
#include "RobotSerialComm.h"

// MRsensing ambiental sensors lib
#include "MRsensing.h"

Omni3MD omni;
Robot robot;
MRsensing sensor;
double ROBOT_ID = 0;
```

```

// Arguments for the reply
unsigned int reply_arg[5];

// RobotSerial Communication
RobotSerialComm port;

//Streaming without interrupt:
boolean stream1=false;
boolean stream2=false;
boolean stream3=false;
boolean stream4=false;

//Variables for motion control
double lin_speed_si=0.0;    //Linear speed in m/s
double ang_speed_si=0.0;    //Rotational speed in rad/s


// ******* Setup *******
void setup(){


  // Serial port stuff
  Serial.begin(BAUD_RATE);  // defined in "Robot.h"

  // I2C connection
  omni.i2c_connect(OMNI3MD_ADDRESS); //set i2c connection
  delay(10);                           // pause 10 milliseconds

  omni.stop_motors();                 // stops all motors
  delay(10);

  omni.set_i2c_timeout(0); // safety parameter -> I2C communication must occur every [byte timeout] x 100 miliseconds for motor movement
  delay(5);                // 5ms pause required for Omni3MD eeprom writing

  // Encoders Reset
  omni.set_enc_value(1,0);   // presets the encoder value [byte encoder, word encValue]
  omni.set_enc_value(3,0);   // presets the encoder value [byte encoder, word encValue]

  //set_prescaler(byte encoder, byte value)  value: 0 - 1 pulse; 1 - 10 pulses; 2 - 100 pulses; 3 - 1000 pulses; 4 - 10000 pulses  (requires 10ms)
  omni.set_prescaler(1, 0);  //sets the prescaler to 100; encoder count will increment by 1 each 100 pulses [byte encoder, byte value]
  delay(10);                 // 10ms pause required for Omni3MD eeprom writing
  omni.set_prescaler(3, 0);
  delay(10);


  ROBOT_ID = robot.EEPROMReadDouble(0);    // TraxBot #1,#2,#3 - StingBot #4,#5 - Pioneers #6,#7,#8,#9,#10
  word ramp_time;
  double axis_radius,whell_radius;

  if (ROBOT_ID <=3) {        // To TraxBot #1,#2,#3
```

```
      ramp_time = 2500;
      axis_radius = 87;
      whell_radius = 27;
    } else {                   // To StingBot #4,#5
      ramp_time = 1500;
      axis_radius = 110;
      whell_radius = 19;
    }

    omni.set_PID(Kp,Ki,Kd); // Adjust paramenters for PID control [word Kp, word Ki, word Kd]
    delay(15);                 // 15ms pause required for Omni3MD eeprom writing

    omni.set_ramp(ramp_time,0);   // set acceleration ramp and limiar take off parameter gain[word ramp_time, word Kl]
    delay(10);                 // 10ms pause required for Omni3MD eeprom writing

    omni.set_differential(axis_radius,whell_radius,gearbox_factor,encoder_cpr);
    delay(20);

    // Give 5v to power sonars, digital pin 13
    delay(250);
    pinMode(13, OUTPUT);
    digitalWrite(13, HIGH);

    // Give 5v to digital pin 12
    pinMode(12, OUTPUT);
    digitalWrite(12, LOW);
      /* Sensors Setup */
    sensor.sensorSetup();
}


// ******* Helper functions *******

void sendEncodersReads(){

    reply_arg[0] = omni.read_enc3();
    reply_arg[1] = omni.read_enc1();
    port.reply(OMNI_READ_ENCODERS, reply_arg, 2);
}

void sendSonarsReads(){

    reply_arg[0] = robot.getRange(FRONT_SONAR);
    reply_arg[1] = robot.getRange(LEFT_SONAR);
    reply_arg[2] = robot.getRange(RIGHT_SONAR);
    port.reply(READ_SONARS, reply_arg, 3);
}

void sendEncodersSonarsReads(){

    reply_arg[0] = omni.read_enc3();
    reply_arg[1] = omni.read_enc1();
    reply_arg[2] = robot.getRange(FRONT_SONAR);
    reply_arg[3] = robot.getRange(LEFT_SONAR);
    reply_arg[4] = robot.getRange(RIGHT_SONAR);
```

```
    port.reply(READ_ENCODERS_SONARS, reply_arg, 5);
}

void sendRobotInfo(){

    double value = robot.EEPROMReadDouble(0); //read Address "0" robot id (expected to be written previously)
    reply_arg[0] = round( omni.read_temperature() * 100 );
    reply_arg[1] = round( omni.read_firmware() * 100 );
    reply_arg[2] = round( omni.read_battery() * 100 );
    reply_arg[3] = ROBOT_FIRMWARE_VERSION;
    reply_arg[4] = (int)value;                // Robot ID
    port.reply(ROBOT_INFO, reply_arg, 5);
}

void sendSensorsInfo(int *thermopile_array){

    int tmax, x;
    sensor.getThermopileSensor(thermopile_array);

    for(x = 0; x < 8 ; x++){
      if(x == 0) tmax = thermopile_array[x];
      if(thermopile_array[x] > tmax) tmax = thermopile_array[x];
    }

    if (ROBOT_ID <= 5) {
      reply_arg[0] = omni.read_enc3();
      reply_arg[1] = omni.read_enc1();
    } else {
      reply_arg[0] = 999;
      reply_arg[1] = 999;
    }
    reply_arg[2] = sensor.getAlcoholSensor();
    reply_arg[3] = sensor.getDustSensor();
    reply_arg[4] = tmax;
    port.reply(MRSENSING_START_STREAM, reply_arg, 5);
}

void sendSensorsLDRInfo(int *thermopile_array){

    int tmax, x;
    sensor.getThermopileSensor(thermopile_array);

    for(x = 0; x < 8 ; x++){
      if(x == 0) tmax = thermopile_array[x];
      if(thermopile_array[x] > tmax) tmax = thermopile_array[x];
    }

    if (ROBOT_ID <= 5) {
      reply_arg[0] = omni.read_enc3();
      reply_arg[1] = omni.read_enc1();
    } else {
      reply_arg[0] = 999;
      reply_arg[1] = 999;
    }
    reply_arg[2] = sensor.getAlcoholSensor();
    reply_arg[3] = sensor.getDustSensor();
    reply_arg[4] = tmax;
    reply_arg[5] = sensor.getLDRsensor();
    port.reply(SENSORS_LDR_START_STREAM, reply_arg, 6);
```

```
}


void sendBearing(){
    reply_arg[0] = sensor.getBearing();
    port.reply(COMPASS_START_STREAM, reply_arg, 1);
}



// ******* Main loop *******
void loop(){

    unsigned int arg[5];
    int action = port.getMsg(arg);

    if(action==0 && stream1==true){
      action=ACTION_START_STREAM;
    }

    if(action==0 && stream2==true){
      action=MRSENSING_START_STREAM;
    }

    if(action==0 && stream3==true){
      action=COMPASS_START_STREAM;
    }

    if(action==0 && stream4==true){
      action=SENSORS_LDR_START_STREAM;
    }


    // If we got an action...Process it:
    switch(action){

            case OMNI_CALIBRATION:                 // "@1e", no reply
                omni.calibrate(1,0,0);
                delay(95);
                break;

            case OMNI_SET_PID:                     //@2,"KP","KI","KD"e, no reply
                omni.set_PID(arg[0], arg[1], arg[2]);
                break;

            case OMNI_SET_PRESCALER:               //@3,"enc","value"e, no reply
                omni.set_prescaler(arg[0], arg[1]);
                break;

            case OMNI_SET_ENC_VALUE:             //@4,"enc","enc_value"e, no reply
                omni.set_enc_value(arg[0], arg[1]);
                break;

            case ROBOT_INFO:                     //@5e, reply: @5,"temp","firm","bat","r_firm","r_id"e
                sendRobotInfo();
                break;
```

```

            case OMNI_READ_ENCODERS:          //@6e,  reply: @6,"enc1(R)","enc2(L)"e
                sendEncodersReads();
                break;

            case READ_SONARS:                //@7e, reply: @7,"son1(F)","son2(L)","son3(R)"e
                sendSonarsReads();
                break;

            case READ_ENCODERS_SONARS:       //@8e, reply: @8,"enc1(R)","enc2(L)","son1(F)","son2(L)","son3(R)"e
                sendEncodersSonarsReads();
                break;

            case LINEAR_MOVE_PID:            //@9,"speed1","speed3"e, no reply
                omni.mov_lin3m_pid(arg[0], 0, arg[1]);
                break;

            case LINEAR_MOVE_NOPID:          //@10,"speed1","speed2"e, no reply
                omni.mov_lin3m_nopid(arg[0], 0, arg[1]);
                break;

            case MOVE_DIFFERENTIAL_SI:          //@11,"vel_linear","vel_angular"e, no reply
                lin_speed_si= ((double)arg[0]/1000);
                ang_speed_si= ((double)arg[1]/1000);
                omni.mov_dif_si(lin_speed_si, ang_speed_si);
                break;

            case MOVE_POSITIONAL:               //@12,"motor_nr","speed","encoder_Position"e, no reply
                omni.mov_pos(arg[0], arg[1], arg[2], 1);  // move motor1 at speed1 until encoder count reaches the defined position and then stop with holding torque
                delay(1);                                  // wait 1ms for Omni3MD to process information
                break;

            case STOP_MOTORS:                //@13e, no reply
                omni.stop_motors();
                break;

            case ENCODERS_RESET:             //@14e, no reply
                robot.encodersReset();
                break;

            case ACTION_GET_DEBUG:           //@15e, reply (to the console): @13,"0/1"e
                reply_arg[0] = port.getDebug();
                port.reply(ACTION_GET_DEBUG, reply_arg, 1);
                break;

            case ACTION_SET_DEBUG:           //@16,"0/1"e, no reply
                port.setDebug(arg[0]);
                break;

```

```
            case ACTION_GET_STREAM:           //@17e, reply @15,"0/1"e
                reply_arg[0] = stream1;
                port.reply(ACTION_GET_STREAM, reply_arg, 1);
                break;

            case ACTION_START_STREAM:      // "@18e,  reply: @8,"enc1(R)","enc2(L)","son1(F)","son2(L)","son3(R)"e (repeatedly)
                stream1 = true;
                sendEncodersSonarsReads();
                //delay(65);                //encoders read update (+- 15Hz)
                break;

            case ACTION_STOP_STREAM:        // "@19e,  no reply
                stream1 = false;
                break;

        case READ_ALCOHOL_SENSOR:        // "@20e,  reply: @21,"Analog output of Alcohol Sensor in mV"e
                reply_arg[0] = sensor.getAlcoholSensor();
                port.reply(READ_ALCOHOL_SENSOR, reply_arg, 1);
                break;

            case READ_DUST_SENSOR:        // "@21e,  reply: @22,"Dust sensor values in PPM"e
                reply_arg[0] = sensor.getDustSensor();
                port.reply(READ_DUST_SENSOR, reply_arg, 1);
                break;

            case READ_THERMOPILE_SENSOR:        // "@22e,  reply: @20,"Frame1","Frame2","Frame3","Frame4","Frame5","Frame6","Frame7","Frame8"e
                int thermopile_arrayy[8];
                sensor.getThermopileSensor(thermopile_arrayy);
                reply_arg[0] = thermopile_arrayy[0];
                reply_arg[1] = thermopile_arrayy[1];
                reply_arg[2] = thermopile_arrayy[2];
                reply_arg[3] = thermopile_arrayy[3];
                reply_arg[4] = thermopile_arrayy[4];
                reply_arg[5] = thermopile_arrayy[5];
                reply_arg[6] = thermopile_arrayy[6];
                reply_arg[7] = thermopile_arrayy[7];
                port.reply(READ_THERMOPILE_SENSOR, reply_arg, 8);
                break;

            //case MRSENSING_START_STREAM:        // "@23e,  reply: @23,"Alcohol Sensor","Dust sensor","Frame1","Frame2","Frame3","Frame4","Frame5","Frame6","Frame7","Frame8"e
              case MRSENSING_START_STREAM:        // "@23e,  reply: @23,"enc1(R)","enc2(L)","Alcohol Sensor","Dust sensor","TempMax"e

                stream2 = true;
                int thermopile_array[8];
                sendSensorsInfo(thermopile_array);
                break;

            case MRSENSING_STOP_STREAM:        // "@24e,  no reply
                //Serial.println("[STOP]");
                stream2 = false;
                break;

```

```
            case SENSORS_LDR_START_STREAM:        // "@25e,  reply: @25,"enc1(R)","enc2(L)","Alcohol Sensor","Dust sensor","TempMax","LDR"e

                stream4 = true;
                int thermopilee_array[8];
                sendSensorsLDRInfo(thermopilee_array);
                break;

            case SENSORS_LDR_STOP_STREAM:        // "@26e,  no reply
                //Serial.println("[STOP]");
                stream4 = false;
                break;

            case COMPASS_START_STREAM:        // "@27e,  reply: @25,"Bearing"e
                stream3 = true;
                sendBearing();
                break;

            case COMPASS_STOP_STREAM:        // "@28e,  no reply
                stream3 = false;
                break;

            default:
                break;


   } // switch
    //delay(1000);
} // loop()

// EOF
```

Annex 4

```python
#!/usr/bin/env python

import roslib; roslib.load_manifest('ml_classifiers')
import rospy
from std_msgs.msg import String
from std_msgs.msg import Float64
from geometry_msgs.msg import Vector3
import ml_classifiers.srv
import ml_classifiers.msg


#Wrapper for calls to ROS classifier service and management of classifier data

class ClassifierWrapper:

    def __init__(self):
        #Set up Classifier service handles
        print 'Waiting for Classifier services...'
        rospy.wait_for_service("/ml_classifiers/create_classifier")
        self.add_class_data = rospy.ServiceProxy(
            "/ml_classifiers/add_class_data",
            ml_classifiers.srv.AddClassData, persistent=True)
        self.classify_data = rospy.ServiceProxy(
            "/ml_classifiers/classify_data",
            ml_classifiers.srv.ClassifyData, persistent=True)
        self.clear_classifier = rospy.ServiceProxy(
            "/ml_classifiers/clear_classifier",
            ml_classifiers.srv.ClearClassifier, persistent=True)
        self.create_classifier = rospy.ServiceProxy(
            "/ml_classifiers/create_classifier",
            ml_classifiers.srv.CreateClassifier, persistent=True)
        self.load_classifier = rospy.ServiceProxy(
            "/ml_classifiers/load_classifier",
            ml_classifiers.srv.LoadClassifier, persistent=True)
        self.save_classifier = rospy.ServiceProxy(
            "/ml_classifiers/save_classifier",
            ml_classifiers.srv.SaveClassifier, persistent=True)
        self.train_classifier = rospy.ServiceProxy(
            "/ml_classifiers/train_classifier",
            ml_classifiers.srv.TrainClassifier, persistent=True)
        print 'OK\n'


    def addClassDataPoint(self, identifier, target_class, p):
        req = ml_classifiers.srv.AddClassDataRequest()
        req.identifier = identifier
        dp = ml_classifiers.msg.ClassDataPoint()
        dp.point = p
        dp.target_class = target_class
        req.data.append(dp)
        resp = self.add_class_data(req)


    def addClassDataPoints(self, identifier, target_classes, pts):
        req = ml_classifiers.srv.AddClassDataRequest()
        req.identifier = identifier
        for i in xrange(len(pts)):
            dp = ml_classifiers.msg.ClassDataPoint()
            dp.point = pts[i]
```

```
            dp.target_class = target_classes[i]
            req.data.append(dp)
        resp = self.add_class_data(req)


    def classifyPoint(self, identifier, p):
        req = ml_classifiers.srv.ClassifyDataRequest()
        req.identifier = identifier
        dp = ml_classifiers.msg.ClassDataPoint()
        dp.point = p
        req.data.append(dp)
        resp = self.classify_data(req)
        return resp.classifications[0]


    def classifyPoints(self, identifier, pts):
        req = ml_classifiers.srv.ClassifyDataRequest()
        req.identifier = identifier
        for p in pts:
            dp = ml_classifiers.msg.ClassDataPoint()
            dp.point = p
            req.data.append(dp)

        resp = self.classify_data(req)
        return resp.classifications


    def clearClassifier(self, identifier):
        req = ml_classifiers.srv.ClearClassifierRequest()
        req.identifier = identifier
        resp = self.clear_classifier(req)


    def createClassifier(self, identifier, class_type):
        req = ml_classifiers.srv.CreateClassifierRequest()
        req.identifier = identifier
        req.class_type = class_type
        resp = self.create_classifier(req)


    def loadClassifier(self, identifier, class_type, filename):
        req = ml_classifiers.srv.LoadClassifierRequest()
        req.identifier = identifier
        req.class_type = class_type
        req.filename = filename
        resp = self.load_classifier(req)


    def saveClassifier(self, identifier, filename):
        req = ml_classifiers.srv.SaveClassifierRequest()
        req.identifier = identifier
        req.filename = filename
        resp = self.save_classifier(req)


    def trainClassifier(self, identifier):
        req = ml_classifiers.srv.TrainClassifierRequest()
        req.identifier = identifier
        resp = self.train_classifier(req)

```

```
def classificationCallback(data):
    #rospy.loginfo(rospy.get_name() + ": I heard %s" % data.data)

    cw = ClassifierWrapper()
    cw.createClassifier('test','ml_classifiers/SVMClassifier')

    import xlrd
    import xlwt
    import sys
    sample = 3
    pts = []
    targs = []

    wb = xlrd.open_workbook('/home/ds_pimp/Desktop/ml_classf_excel/x1.xlsx')
    sh = wb.sheet_by_index(0)
    for rownum in range(sample):
    pts.append(sh.row_values(rownum))


    wb = xlrd.open_workbook('/home/ds_pimp/Desktop/ml_classf_excel/x2.xlsx')
    sh = wb.sheet_by_index(0)
    for rownum in range(sample):
    pts.append(sh.row_values(rownum))


    wb = xlrd.open_workbook('/home/ds_pimp/Desktop/ml_classf_excel/x3.xlsx')
    sh = wb.sheet_by_index(0)
    for rownum in range(sample):
    pts.append(sh.row_values(rownum))

    wb = xlrd.open_workbook('/home/ds_pimp/Desktop/ml_classf_excel/x4.xlsx')
    sh = wb.sheet_by_index(0)
    for rownum in range(sample):
    pts.append(sh.row_values(rownum))


   # print pts
    for rownum in range(sample):
    targs.append('1')
    for rownum in range(sample):
    targs.append('2')
    for rownum in range(sample):
    targs.append('3')
    for rownum in range(sample):
    targs.append('4')


    cw.addClassDataPoints('test', targs, pts)
    cw.trainClassifier('test')

    #testpts = [[20.0, 500.0, 560.0]]

    testpts = [[data.x,data.y,data.z]]

    print testpts
    resp = cw.classifyPoints('test',testpts)
```

```
    print resp

    pub.publish(String(str(resp)))


if __name__ == '__main__':

    pub = rospy.Publisher('/mrsensing_classiffication', String)
    rospy.init_node('mrsensing')
    rospy.Subscriber("/mrsensing_dataSensors_throttle", Vector3, classificationCallback)

    rospy.spin()
```

Annex 5

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <string>
#include <vector>

#include <ros/ros.h>
#include <visualization_msgs/Marker.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>

float x=0, y=0, z=0, w=0;
float x_temp=0, y_temp=0, z_temp=0, w_temp=0;
float dust_sensor=0, acohol_sensor=0, thermopile_sensor=0;
int cnt=0;

ros::Publisher marker_pub;
ros::Subscriber classification_sub;
ros::Subscriber dataSensors_sub;
ros::Subscriber odom_sub;
ros::Subscriber amcl_sub;

// void classificationCallback(std::string * data) {
//
// }

void dataSensorsCallback(const geometry_msgs::Vector3::ConstPtr& msg) {

  acohol_sensor=msg->x;
  dust_sensor=msg->y;
  thermopile_sensor=msg->z;

}


//void amclCallback(const geometry_msgs::PoseWithCovarianceStamped::ConstPtr& msg) {
void odomCallback(const nav_msgs::Odometry::ConstPtr& msg) {

  x=msg->pose.pose.position.x;
  y=msg->pose.pose.position.y;
  z=msg->pose.pose.orientation.z;
  w=msg->pose.pose.orientation.w;


  uint32_t shape = visualization_msgs::Marker::CUBE;
  //shape = visualization_msgs::Marker::SPHERE;
  //shape = visualization_msgs::Marker::ARROW;
  //shape = visualization_msgs::Marker::CYLINDER;
  //shape = visualization_msgs::Marker::CUBE;

  visualization_msgs::Marker marker;
  // Set the frame ID and timestamp.  See the TF tutorials for information on these.
  marker.header.frame_id = "robot_0/base_link";
  marker.header.stamp = ros::Time::now();

```

```
  // Set the namespace and id for this marker.  This serves to create a unique ID
  // Any marker sent with the same namespace and id will overwrite the old one

  marker.ns = "basic_shapes_robot0";
  marker.id = cnt;
  cnt++;
  marker.type = shape;
  marker.action = visualization_msgs::Marker::ADD;
  marker.pose.position.x = 0;
  marker.pose.position.y = 0;
  marker.pose.position.z = 0;
  marker.pose.orientation.x = 0.0;
  marker.pose.orientation.y = 0.0;
  marker.pose.orientation.z = 0;
  marker.pose.orientation.w = 0;

  marker.scale.x = 0.20;
  marker.scale.y = 0.23;
  marker.scale.z = 0.3;

  if((thermopile_sensor>40) && (thermopile_sensor<145)){
  marker.color.r = 1.0f;
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  } else if ((thermopile_sensor>145) && (thermopile_sensor<160)) {

  marker.color.r = 1.0f;
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.08;
  } else if (thermopile_sensor>160) {

  marker.color.r = 1.0f;
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.3;
  } else {

  marker.color.r = 0.0f;
  marker.color.g = 1.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  }





 if((acohol_sensor>360) && (acohol_sensor<390)){
  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 0.02;
  } else if ((acohol_sensor>390) && (acohol_sensor<410)) {

  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
```

```cpp
  marker.color.b = 1.0f;
  marker.color.a = 0.08;
  } else if (acohol_sensor>410) {

  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 0.3;
  } else if (thermopile_sensor<40){

  marker.color.r = 0.0f;
  marker.color.g = 1.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  }


//    thermopile_sensor=thermopile_sensor/100;
//    if (thermopile_sensor>100 || thermopile_sensor==0) {
//     thermopile_sensor=1;
//    }

  marker.lifetime = ros::Duration();

  marker_pub.publish(marker);


}



int main( int argc, char** argv )
{
  ros::init(argc, argv, "classification_markers_robot0");
  ros::NodeHandle n;
  ros::Rate r(1);
  marker_pub = n.advertise<visualization_msgs::Marker>("robot_0/visualization_marker_robot", 1);
  odom_sub = n.subscribe("robot_0/odom", 1, odomCallback);
  //amcl_sub = n.subscribe("robot_0/amcl_pose", 1, amclCallback);
  //classification_sub = n.subscribe("mrsensing_classiffication", 1, classificationCallback);
  dataSensors_sub = n.subscribe("robot_0/mrsensing_dataSensors", 1, dataSensorsCallback);

  ros::spin();

}
```

Annex 6

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <unistd.h>
#include <string>
#include <vector>

#include <ros/ros.h>
#include <visualization_msgs/Marker.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Vector3.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>

float x=0, y=0, z=0, w=0;
float x_temp=0, y_temp=0, z_temp=0, w_temp=0;
float dust_sensor=0, acohol_sensor=0, thermopile_sensor=0;
int cnt=0;

ros::Publisher marker_pub;
ros::Subscriber classification_sub;
ros::Subscriber dataSensors_sub;
ros::Subscriber odom_sub;
ros::Subscriber amcl_sub;

void classificationCallback(std::string * data) {

   class_svm=data;

}


void dataSensorsCallback(const geometry_msgs::Vector3::ConstPtr& msg) {

  acohol_sensor=msg->x;
  dust_sensor=msg->y;
  thermopile_sensor=msg->z;

}


//void amclCallback(const geometry_msgs::PoseWithCovarianceStamped::ConstPtr& msg) {

void odomCallback(const nav_msgs::Odometry::ConstPtr& msg) {

  x=msg->pose.pose.position.x;
  y=msg->pose.pose.position.y;
  z=msg->pose.pose.orientation.z;
  w=msg->pose.pose.orientation.w;

  //ROS_INFO("x=%f y=%f z=%f w=%f",x,y,z,w);


  uint32_t shape = visualization_msgs::Marker::CUBE;
  //shape = visualization_msgs::Marker::SPHERE;
  //shape = visualization_msgs::Marker::ARROW;
  //shape = visualization_msgs::Marker::CYLINDER;
  //shape = visualization_msgs::Marker::CUBE;

```

```
  visualization_msgs::Marker marker;
  // Set the frame ID and timestamp.  See the TF tutorials for information on these.
  marker.header.frame_id = "robot_1/base_link";
  marker.header.stamp = ros::Time::now();

  // Set the namespace and id for this marker.  This serves to create a unique ID
  // Any marker sent with the same namespace and id will overwrite the old one

  marker.ns = "basic_shapes_robot1";
  marker.id = cnt;
  cnt++;
  marker.type = shape;
  marker.action = visualization_msgs::Marker::ADD;
  marker.pose.position.x = 0;
  marker.pose.position.y = 0;
  marker.pose.position.z = 0;
  marker.pose.orientation.x = 0.0;
  marker.pose.orientation.y = 0.0;
  marker.pose.orientation.z = 0;
  marker.pose.orientation.w = 0;

  marker.scale.x = 0.20;
  marker.scale.y = 0.43;
  marker.scale.z = 0.3;


  if(class_svm=1) {
  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 0.3;
  } else if (class_svm=3) {
  marker.color.r = 1.0f;
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.3;
  } else if (class_svm=4) {
  marker.color.r = 0.0f;
  marker.color.g = 1.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  }

  if((thermopile_sensor>40) && (thermopile_sensor<145)){
  marker.color.r = 1.0f;
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  } else if ((thermopile_sensor>145) && (thermopile_sensor<150)) {

  marker.color.r = 1.0f;
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.08;
  } else if (thermopile_sensor>150) {

  marker.color.r = 1.0f;
```

```
  marker.color.g = 0.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.3;
  } else {

  marker.color.r = 0.0f;
  marker.color.g = 1.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  }


 if((acohol_sensor>360) && (acohol_sensor<390)){
  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 0.02;
  } else if ((acohol_sensor>390) && (acohol_sensor<410)) {

  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 0.08;
  } else if (acohol_sensor>410) {

  marker.color.r = 0.0f;
  marker.color.g = 0.0f;
  marker.color.b = 1.0f;
  marker.color.a = 0.3;
  } else if (thermopile_sensor<40){

  marker.color.r = 0.0f;
  marker.color.g = 1.0f;
  marker.color.b = 0.0f;
  marker.color.a = 0.02;
  }


//   thermopile_sensor=thermopile_sensor/100;
//   if (thermopile_sensor>100 || thermopile_sensor==0) {
//    thermopile_sensor=1;
//   }

  marker.lifetime = ros::Duration();
  marker_pub.publish(marker);


}



int main( int argc, char** argv )
{
  ros::init(argc, argv, "classification_markers_robot1");
  ros::NodeHandle n;
  ros::Rate r(1);
  marker_pub = n.advertise<visualization_msgs::Marker>("robot_1/visualization_marker_robot", 1);
  odom_sub = n.subscribe("robot_1/odom", 1, odomCallback);
  //amcl_sub = n.subscribe("robot_1/amcl_pose", 1, amclCallback);
```

```
  //classification_sub = n.subscribe("mrsensing_classiffication", 1, classificationCallback);
  dataSensors_sub = n.subscribe("robot_1/mrsensing_dataSensors", 1, dataSensorsCallback);

  ros::spin();

}

```

Annex 7

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <vector>

#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>            // odom
#include <geometry_msgs/Twist.h>        // cmd_vel

#include <cereal_port/CerealPort.h>
#include <geometry_msgs/Vector3.h>
#include <std_msgs/String.h>
#include <std_msgs/Float32.h>




ros::Publisher pub_sensors;
ros::Publisher pub_LDRsensor;

cereal::CerealPort serial_port;


bool signof (int n) { return n >= 0; }
bool confirm_communication = true;
int ID_Robot = 0;



//Receive encoder ticks and send 'odom' and 'tf'
void robotDataCallback(std::string * data){

    if (confirm_communication){
      //ROS_INFO("Robot -- Communication OK! Received: \"%s\"", data->c_str());
      ROS_INFO("Traxbot is Streaming Data.");
      confirm_communication = false;
    }

    int first_at = data->find_first_of("@", 0);
    int second_at = data->find_first_of("@", first_at+1);
    int first_comma = data->find_first_of(",", 0);
    int second_comma = data->find_first_of(",", first_comma+1);

    //protection against broken msgs from the buffer (e.g., '@6,425@6,4250,6430e')
    if ( second_at > -1 || second_comma == -1){
      ROS_WARN("%s ::: ENCODER MSG IGNORED", data->c_str());
      return;
    }

    int left_encoder_count, right_encoder_count, alcohol_sensor, dust_sensor, temp_max, ldr;
    sscanf(data->c_str(), "@25,%d,%d,%d,%d,%d,%de", &right_encoder_count, &left_encoder_count, &alcohol_sensor, &dust_sensor, &temp_max, &ldr);   //encoder msg parsing

    geometry_msgs::Vector3 msg;
    msg.x=alcohol_sensor;
```

```
    msg.y=dust_sensor;
    msg.z=temp_max;

    std_msgs::Float32 msg_ldr;
    msg_ldr.data=ldr;

    // Publish the message
    pub_LDRsensor.publish(msg_ldr);
    pub_sensors.publish(msg);
    ros::spinOnce();
}


int main(int argc, char** argv){ //typical usage: "./traxbot_node /dev/ttyACMx"

    ros::init(argc, argv, "traxbot_node");
    ros::NodeHandle n;
    ros::NodeHandle pn("~");
    std::string port;

    if (argc<2){
     port="/dev/ttyACM0";
     ROS_WARN("No Serial Port defined, defaulting to \"%s\"",port.c_str());
     ROS_WARN("Usage: \"rosrun [pkg] robot_node /serial_port\"");
    }else{
      port=(std::string)argv[1];
      ROS_INFO("Serial port: %s",port.c_str());
    }

    // ROS publishers and subscribers
        pub_sensors = n.advertise<geometry_msgs::Vector3>("/mrsensing_dataSensors", 1);
    pub_LDRsensor = n.advertise<std_msgs::Float32>("/LDR_Sensor", 1);


    // baud_rate and serial port:
    int baudrate;
    pn.param<std::string>("port", port, port.c_str());
    pn.param("baudrate", baudrate, 115200);

    // Open the serial port to the robot
    try{ serial_port.open((char*)port.c_str(), baudrate); }
    catch(cereal::Exception& e){
        ROS_FATAL("Robot -- Failed to open serial port!");
        ROS_BREAK();
    }

    //wait (2.5 seconds) until serial port gets ready
    ros::Duration(2.5).sleep();

    // Ask Robot ID from the Arduino board (stored in the EEPROM)
    ROS_INFO("Starting Traxbot...");
    serial_port.write("@5e");
    std::string reply;

    try{ serial_port.readBetween(&reply,'@','e'); }
    catch(cereal::TimeoutException& e){
      ROS_ERROR("Initial Read Timeout!");
    }
```

```cpp

    int VDriver, Temperature, OMNI_Firmware, Battery;
    sscanf(reply.c_str(), "@5,%d,%d,%d,%d,%de", &Temperature, &OMNI_Firmware, &Battery, &VDriver, &ID_Robot);   //encoder msg parsing

    ROS_INFO("Traxbot ID = %d", ID_Robot);
    if (ID_Robot < 1 || ID_Robot > 10){
        ROS_WARN("Attention! Unexpected Traxbot ID!");
    }
    ROS_INFO("OMNI Board Temperature = %.2f C", Temperature*0.01);
    ROS_INFO("OMNI Firmware = %.2f", OMNI_Firmware*0.01);
    ROS_INFO("Arduino Firmware Version = %d.00", VDriver);

    if (VDriver > 1500){
        ROS_ERROR("Reset Robot Connection and try again.");
        return(0);
    }

    // Start receiving streaming data
    if( !serial_port.startReadBetweenStream(boost::bind(&robotDataCallback, _1), '@', 'e') ){
          ROS_FATAL("Robot -- Failed to start streaming data!");
          ROS_BREAK();
    }
    serial_port.write("@25e");

    ros::spin(); //trigger callbacks and prevents exiting
      return(0);
}
```

Annex 8

view_frames Result
Recorded at time: 1375309592.813
/map
Broadcaster: /amcl_traxbot
Average rate: 10.213 Hz
Most recent transform: 1.946 sec old
Buffer length: 4.798 sec
Broadcaster: /amcl_pioneer
Average rate: 10.189 Hz
Most recent transform: 1.905 sec old
Buffer length: 4.809 sec
/robot_0/odom
/robot_1/odom
Broadcaster: /traxbot_node
Average rate: 147.721 Hz
Most recent transform: 1.898 sec old
Buffer length: 4.915 sec
Broadcaster: /ROSARIA
Average rate: 10.204 Hz
Most recent transform: 1.994 sec old
Buffer length: 4.802 sec
/robot_0/base_link
/robot_1/base_link
Broadcaster: /robot_0/robot_state_publisher_traxbot
Average rate: 15.205 Hz
Most recent transform: -0.439 sec old
Buffer length: 4.867 sec
Broadcaster: /robot_0/robot_state_publisher_traxbot
Average rate: 15.205 Hz
Most recent transform: -0.439 sec old
Buffer length: 4.867 sec
Broadcaster: /robot_0/robot_state_publisher_traxbot
Average rate: 15.205 Hz
Most recent transform: -0.439 sec old
Buffer length: 4.867 sec
Broadcaster: /base_link_to_laser_0
Average rate: 10.174 Hz
Most recent transform: 1.833 sec old
Buffer length: 4.816 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.203 Hz
Most recent transform: -0.473 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.203 Hz
Most recent transform: -0.473 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.208 Hz
Most recent transform: 0.079 sec old
Buffer length: 4.800 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.208 Hz
Most recent transform: 0.079 sec old
Buffer length: 4.800 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.203 Hz
Most recent transform: -0.473 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.208 Hz
Most recent transform: 0.079 sec old
Buffer length: 4.800 sec
/robot_0/acrylic_support
/robot_0/tracks_bat_caster
/robot_0/wheels_sonars
/robot_0/laser
/robot_1/back_sonar
/robot_1/front_sonar
/robot_1/p3dx_left_wheel
/robot_1/p3dx_right_wheel
/robot_1/top_plate
/robot_1/swivel
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.203 Hz
Most recent transform: -0.473 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.203 Hz
Most recent transform: -0.473 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.208 Hz
Most recent transform: 0.079 sec old
Buffer length: 4.800 sec
/robot_1/p3dx_left_hubcap
/robot_1/p3dx_right_hubcap
/robot_1/center_wheel
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.203 Hz
Most recent transform: -0.473 sec old
Buffer length: 4.900 sec
/robot_1/center_hubcap

Annex 9

view_frames Result
Recorded at time: 1375313807.416
/map
Broadcaster: /amcl_pioneer
Average rate: 10.172 Hz
Most recent transform: 2.030 sec old
Buffer length: 4.817 sec
Broadcaster: /amcl_traxbot
Average rate: 10.229 Hz
Most recent transform: 2.027 sec old
Buffer length: 4.888 sec
/robot_1/odom
/robot_0/odom
Broadcaster: /ROSARIA
Average rate: 10.204 Hz
Most recent transform: 2.109 sec old
Buffer length: 4.802 sec
Broadcaster: /traxbot_node
Average rate: 9.967 Hz
Most recent transform: 2.100 sec old
Buffer length: 4.816 sec
/robot_1/base_link
/robot_0/base_link
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.202 Hz
Most recent transform: 0.065 sec old
Buffer length: 4.868 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.202 Hz
Most recent transform: 0.065 sec old
Buffer length: 4.868 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.202 Hz
Most recent transform: 0.065 sec old
Buffer length: 4.868 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.202 Hz
Most recent transform: -0.485 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.202 Hz
Most recent transform: -0.485 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.202 Hz
Most recent transform: -0.485 sec old
Buffer length: 4.900 sec
Broadcaster: /base_link_to_laser_1
Average rate: 10.192 Hz
Most recent transform: 2.001 sec old
Buffer length: 4.808 sec
Broadcaster: /base_link_to_laser_0
Average rate: 10.164 Hz
Most recent transform: 2.000 sec old
Buffer length: 4.821 sec
Broadcaster: /robot_0/robot_state_publisher_traxbot
Average rate: 15.209 Hz
Most recent transform: -0.470 sec old
Buffer length: 4.866 sec
Broadcaster: /robot_0/robot_state_publisher_traxbot
Average rate: 15.209 Hz
Most recent transform: -0.470 sec old
Buffer length: 4.866 sec
Broadcaster: /robot_0/robot_state_publisher_traxbot
Average rate: 15.209 Hz
Most recent transform: -0.470 sec old
Buffer length: 4.866 sec
/robot_1/p3dx_left_wheel
/robot_1/p3dx_right_wheel
/robot_1/swivel
/robot_1/back_sonar
/robot_1/front_sonar
/robot_1/top_plate
/robot_1/laser
/robot_0/laser
/robot_0/acrylic_support
/robot_0/tracks_bat_caster
/robot_0/wheels_sonars
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.202 Hz
Most recent transform: -0.485 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.202 Hz
Most recent transform: -0.485 sec old
Buffer length: 4.900 sec
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 15.202 Hz
Most recent transform: 0.065 sec old
Buffer length: 4.868 sec
/robot_1/p3dx_left_hubcap
/robot_1/p3dx_right_hubcap
/robot_1/center_wheel
Broadcaster: /robot_1/robot_state_publisher_pioneer
Average rate: 30.202 Hz
Most recent transform: -0.485 sec old
Buffer length: 4.900 sec
/robot_1/center_hubcap

Annex 10

Multi-Sensor Fusion and Classification with Mobile Robots for Situation Awareness in Urban Search and Rescue using ROS

Nuno L. Ferreira, Micael S. Couceiro, *Student Member, IEEE,* Andre Araújo, and Rui P. Rocha, *Member, IEEE*

***Abstract*— Multi-sensor information fusion theory concerns the environmental perception activities to combine data from multiple sensory resources. Mobile robots can gather information from the environment by combining information from different sensors as a way to organize decisions and augment human perception. This is especially useful to retrieve contextual environmental information in catastrophic incidents where human perception may be limited (*e.g.*, lack of visibility). To that end, this paper proposes a specific configuration of sensors assembled in a mobile robot, which can be used as a proof concept to measure important environmental variables in an urban search and rescue (*USAR*) mission, such as toxic gas density, temperature gradient and smoke particles density. This data is processed through a support vector machine classifier with the purpose of detecting relevant contexts in the course of the mission. The outcome provided by the experimental experiments conducted with *TraxBot* and *Pioneer-3DX* robots under the Robot Operating System framework opens the door for new multi-robot applications on *USAR* scenarios.**

I. Introduction

Within traditional methods, the information acquired from multiple sensors is processed separately, cutting off the possible connections and dependencies between the acquired information, thus possibly losing significant characteristics from the environment [1]. For instance, in this work, a dust sensor is included to detect smoke as a possible existence of fire in the vicinities. However, as dust and smoke are particle composites, this may induce in a misclassification error. As opposed to the traditional method, several computing methods, usually denoted as multi-sensor information fusion methods [2], allow to analyze and synthesize information from different nodes. Such approach has been widely used for real-time processing, *e.g.*, [1][3].

The topic regarding multi-sensor information on mobile robot environmental monitoring has been recently exploited in the literature. For instance, the work of Larionova *et al.* [4] describes a multi-sensor information approach for landmine detection. Similarly to our approach, the authors extract the most relevant features used for the adequate classification. Afterwards, the well-known principal component approach (*PCA*) is adopted to assess the detection of landmines. Alternatively, the approach of Belur *et al.* [5] went further in terms of information levels at which the fusion is accomplished. The authors took into consideration the objectives of the fusion process, the application domain and the types of sensors employed, or the sensor suite configuration for each situation. More directed to the mobile robotic field, Jason *et al.* [6] presented the need of integrating multiple sensors to accomplish tasks such as map building, object recognition, obstacle avoidance, self-localization and path planning, surveying several sensor fusion categories. More recently, the work of Julien *et al.* [7] presented an information-theoretic approach to distributively control multiple robots equipped with sensors to infer the state of an environment. To that end, the authors proposed a non-parametric Bayesian method for representing the robots' beliefs and likely observations to enable distributed inference and coordination.

Despite the large scope of applicability of multi-sensor fusion on robotics, only some few works have recently focused on catastrophic incidents, as it is the example of the Cooperation between Human and rObotic teams in catastroPhic Incidents (*CHOPIN*) R&D Project [2]. The *CHOPIN* project aims at exploiting the human-robot symbiosis in the development of human rescuers' support systems for urban search and rescue (*USAR*) missions. One of the test scenarios that was chosen to develop a proof of concept is the occurrence of fire outbreaks in a large basement garage. In this use case, the project aims to demonstrate the deployment of a fleet of ground mobile robots to cooperatively explore the basement garage wherein the fire is progressing, thus helping human rescuers to detect and localize fire outbreaks and victims [8].

Following the trend of research, this work benefits from the Robotic Operating System (*ROS*) framework [9], so as to perform real world experimentation while having access to a large number of tools for both analysis and visualizations (*e.g. rviz and rxgraph). ROS* is currently the most popular robotic framework in the world, being the closest one to become the standard that the robotics community urgently needed [9].

This paper presents the first steps towards the implementation of a multi-sensor fusion strategy on such a team of cooperative mobile robots. From the analysis of related work, one may conclude that there are two main topics that need to be addressed: the multi-sensor fusion architecture, and the method to infer information from multi-sensor data. Hence, before presenting our approach, let us introduce some concepts regarding these two topics.

II. Multi-Sensor System Architectures

Since a single sensor generally can only perceive limited or partial information about the environment, multiple similar and dissimilar sensors are required to provide sufficient local information with different focus and from different viewpoints in an integrated manner. Information from heterogeneous sensors can be combined using data fusion algorithms to obtain observable data [10]. A multi-sensor system has the advantage to broaden machine perception and enhance awareness of the state of the world compared to what could be acquired with a single sensor system [11].

Therefore, multiple sensors are needed in response to the increasingly learning nature of the environment to be sensed. This motivates the emerging interest in research into contextual environmental information in catastrophic incidents (*e.g.*, urban fire). It is also beneficial to avoid overwhelming storage and computational requirements in a sensor and data rich

This work was supported by the CHOPIN research project (PTDC/EEA-CRO/119000/2010), by PhD scholarships SFRH/BD/64426/2009 and SFRH/BD /73382/2010, and by Institute of Systems and Robotics (project PEst-C/EEI/UI0048/2011), all of them funded by the Portuguese science agency "Fundação para a Ciência e a Tecnologia" (FCT).
N. Ferreira, M. Couceiro, A. Araújo and R. P. Rocha are with the Institute of Systems and Robotics, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal, email: {nunoferreira, micaelcouceiro, aaraujo, rprocha}@isr.uc.pt.

[2] http://chopin.isr.uc.pt/

environment, by controlling the data gathering process such that only the truly necessary data is collected and stored. The simplest task of sensor management is to choose the optimal sensor parameter values, given one or more sensors, with respect to a given task. This is called active perception wherein sensors need to be optimally configured for a specific purpose.

The basic purpose of sensor management is to adapt the sensor's behavior to dynamic environments. By having limited sensing resources, sensors may not be able to serve all desired tasks and achieve all their associated goals. Therefore, a reasonable process has to be made. In other words, more urgent or important tasks should be given higher priority in their competition for resources. The first step for the sensor management system should be to utilize evidences gathered to decide upon objects of interest and to prioritize which objects to look at in the near future. An illustrative scenario requiring sensor coordination is shown in Fig. 1, wherein 3 mobile robots equipped with different sensor devices cooperatively explore an area of interest.

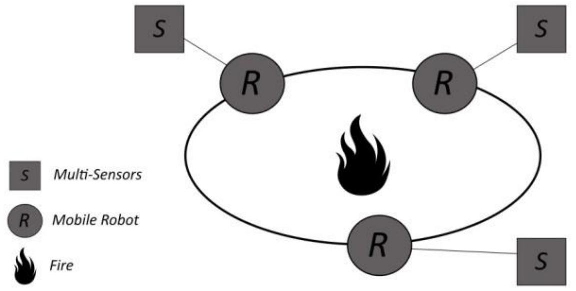

Figure 1. A team of mobile robots with multi-sensors cooperatively observing an area of fire in different points.

However, to achieve some sort of decision-making, each robot needs to be capable of inferring its local contextual information. To that end, for this learning process, pattern classification techniques are needed.

III. CLASSIFICATION METHODS

The literature provides more methods for multi-sensor information that one may be able to use; the options are almost limitless. In brief, the classification is the process of supervised learning where the data is separated into different classes on the basis of one or more characteristics. Artificial Neural Network (ANN), Fuzzy Logic, Bayesian Probability and Support Vector Machine (SVM) are some of the most used classification technics in sensor fusion for back-propagation learning algorithms. In this work, we use and describe with some detail one of the most well-known classification methods – the Support Vector Machine (*SVM*). At the end of this section, the rationale behind the choice of SVM is presented.

A. Support Vector Machine (SVM)

In machine learning, a *SVM* is a supervised learning model involving a learning algorithm to analyze data and recognize patterns, used for classification and regression analysis [1], [13], [14] and [15]. Given a set of training examples, each marked as belonging to one of two categories, an *SVM* training algorithm builds a model that assigns new examples into one category or the other. An *SVM* model is a representation of examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall in. In addition to performing linear classification, *SVM*s can efficiently perform non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. The technique originated from the work of Vapnik on the Principle of Risk Minimization, in the area of Statistical Learning [14][16]. The technique can take two different variants: in the case of linear space, the hyperplanes of separation are determined by an optimization algorithm; in the case of non-linear space, a kernel function is applied and the new space obtained is denominated the feature space [1]. Its optimal function can be expressed as:

$$f(x) = w \times \emptyset(x) + n, \quad (1)$$

in which w is a vector and n a scalar. The dimensionality of $\emptyset(x)$ can be very large, making w hard to represent explicitly in memory as,

$$w = \sum_{i=1}^{m} \propto_i \emptyset(xi). \quad (2)$$

So the decision function is represented as:

$$f(x) = \sum_{i=1} \propto_i \emptyset(x_i)\emptyset(x) + b = \sum_{i=1} \propto_i K(x_i, x) + b, \quad (3)$$

and the dual dormation as

$$min\, P(w,b) = \frac{1}{2}\left|\left|\sum_{i=1}^{m} \propto_i (x_i)\right|\right| + C\sum_{i=1} H_1[y_i f(x_i)] \ . \quad (4)$$

B. Comparison of Classification Methods

The literature is not consensual on deciding upon the most adequate classification method. In fact, in most situations, it depends on the requirements, either in terms of computational and memory complexity or in terms of type and dependency between measured variables. In [17], a comparative study of *SVM*, Artificial Neural Network (*ANN*) and Bayesian Classifier (*BC*) was carried out. The performance of the classifiers were compared to determine the best model for prediction of mutagenicity of the dataset. A higher sensitivity regarding the *SVM* (69.14%) was found, outperforming the *ANN* (40.20%) and the *BC* (58.44%). Also, the precision of the *SVM* model (74.9%) was comparatively higher than both the *ANN* (70.00%) and the *BC* (72.38%) models. Moreover, the *SVM* was able to predict 15% and 5.5% less false negatives than the *ANN* and the *BC* models, respectively. The overall accuracy of the *SVM* was found to be 71.73%, whereas the accuracy of the *ANN* and the *BC* approaches were 59.72% and 66.14%, respectively. Fig. 2 represents the measure of efficiency of the three classifiers.

In a very different domain, the work on [18] presented a comparative study between Linear Discriminant Analysis (*LDA*), Quadratic Discriminant Analysis (*QDA*), Naive Bayes with Normal (Gaussian) distribution (*NV*) [19], Naive Bayes with Kernel Smoothing Density Estimate (*NVK*) and Least Squares *SVM* with Radial Basis Function Kernel, for golf putting performance analysis.

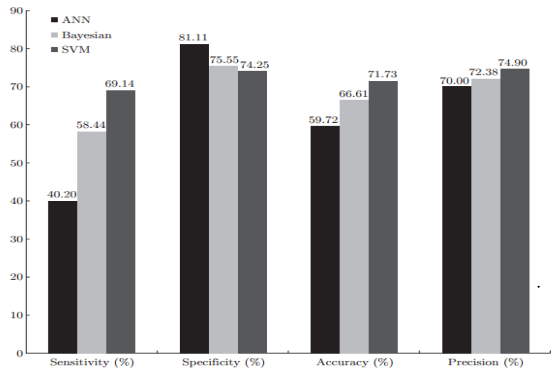

Figure 2. Measure of efficiency of the three classifiers [17].

The parameters of the putter's trajectory mathematical model were used as sample of the classification algorithms. To that end, 6 expert golfers (*i.e.*, 6 classes) performed 30 putt executions in which the horizontal trajectory was fitted by a sinusoidal function. The five classification methods were compared through the analysis of the confusion matrix and the area under the Receiver Operating Characteristic curve (*AUC*). From TABLE I it was possible to confirm that the SVM presented the most consistent results for the accurate classification of each golfer.

TABLE I. AVERAGE VALUE OF THE AUC [18].

Class	*LDA*	*NV*	*NVK*	*SVM*
1	0.619	0.601	0.680	0.744
2	0.650	0.623	0.685	0.737
3	0.566	0.582	0.761	0.737
4	0.507	0.585	0.675	0.690
5	0.622	0.651	0.766	0.797
6	0.493	0.602	0.718	0.745

C. Discussion and Decision

The most widely used data fusion methods employed in robotics comes from the fields of statistics, estimation and control. However, the applicability of these methods in robotics has a number of unique features and challenges. In particular, and as the autonomy is the main goal, the results must be presented and interpreted in a form from which autonomous decisions (*e.g.*, recognition or navigation) can be made. In this study, it was possible to enumerate a set of efficient alternatives to heavy probabilistic methods. Within such set, the *SVM* present itself as a recent technique suitable for binary classification tasks, which is related to and contains elements of non-parametric applied statistics, neural networks and machine learning. As we can see in [17] and [18], the results arising from alternative models are acceptable, but the *SVM* was found to be more efficient in the overall analysis. Since *SVM* uses kernel, it contains a non-linear transformation without assumptions about the functional form of the transformation, which makes data linearly separable. The transformation occurs implicitly on a robust theoretical basis and, contrarily to fuzzy logic or *NV*, human expertise judgment beforehand is not needed. *SVM* provides a good out-of-sample generalization. This means that, by choosing an appropriate generalization grade, the *SVM* can be robust even when the training sample has some bias.

IV. PROPOSED MULTI-SENSOR FUSION SYSTEM

The context of this work involves Urban Search and Rescue (*USAR*) emergency scenarios, focusing on fire outbreaks occurring in large basement garages. To that end, and as proof-of-concept, three low-cost sensors were chosen:

A.1 **Dust sensor** (model *PPD42NS*[3]). Manufactured by Grove, this sensor returns a modulated digital output based on the detected Particulate Matters (*PM*). The output responds to *PM* whose size is around 1 micro meter or larger. Considering D as the number of particles with, at least, 1µm diameter, the output of the dust sensor is define as:

$$0 \leq D \leq 40000 \; [pcs/litre] \tag{5}$$

A.2 **Thermopile Array sensor** (model TPA81[4]). This sensor is characterized by its ability to output an array of 8 elements of 8 bits each. The analog value corresponds directly to the temperature. Hence, one may define the thermopile output as:

$$\begin{gathered} 10 \leq T_i \leq 100 \; [℃] \\ (T_i, i = 1, \ldots, 8.\; T_i \; 8 \; bits \; entry T = \max_i v_i) \end{gathered} \tag{6}$$

A.3 **Alcohol Sensor** (model MQ303A[5]). This sensor has the feature to output (A) a voltage inversely proportional to the alcohol concentration in the air:

$$0 \leq A \leq 700 \; [mV] \tag{7}$$

The choice of these three sensors took into account the environmental variables involved in the *CHOPIN* project. As previously stated, as a single sensor may induce to misclassification, the dust sensor was chosen to work with the thermopile array to detect fire, and with the alcohol sensor to detect air contamination, like gas leaks.

A. Experimental Setup for Training Database

To minimize undesired external contamination during the training process of the *SVM*, an experimental multi-sensor testbed platform setup was built (Fig. 3). This testbed was designed as an isolated and controlled environment. The testbed presented on Fig.3a is based on a sealed glass aquarium that was transformed to create air flow inside the test area with the integration of two 120 mm fans fixed on the top of aquarium: one for air inflow and another for air outflow. Clean or contaminated controlled air flow samples were introduced within the testbed to measure all achievable range of classes. An additional fan was afterwards equipped near the alcohol sensor for a faster settling time of the readings (Fig.3b).

An *Arduino Uno* board with embedded *Atmel 328* microcontroller was used to preprocess the output data from the sensors. Afterwards, the data was sent through a serial connection to a computer using *Robot Operating System* (*ROS*) [9], taking into account the future use of the classifier *ml_classifier*[6] in the real experiments.

[3] http://www.sca-shinyei.com/pdf/PPD42NS.pdf
[4] http://www.robot-electronics.co.uk/htm/tpa81tech.htm
[5] http://www.seeedstudio.com/depot/images/product/MQ303A.pdf
[6] http://www.ros.org/wiki/ml_classifiers

a) b)

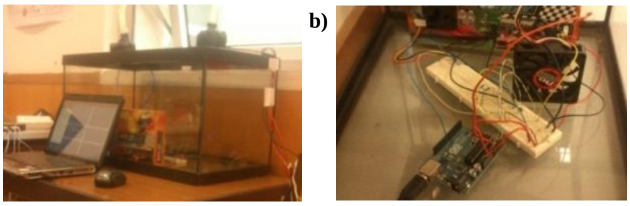

Figure 3. Experimental setup for training database. **a)** Testbed; **b)** Acquisition and pre-processig electronic setup.

B. *SVM Classification and Results*

In this project, several preliminary tests under different conditions were carried out for acquisition of the training data. The data returned from the sensors was acquired as:

$$X = \begin{bmatrix} T_1 & D_1 & A_1 \\ & \vdots & \\ T_n & D_n & A_n \end{bmatrix},$$

wherein the number of rows n represents the number of acquired samples, *i.e.*, trials. An example of the acquired output are presented in TABLE II.

TABLE II. OUTPUT ACQUIRED FROM THE SENSORS. COLUMN 1- TPA81 THERMOPILE ARRAY (T_n), COLUMN 2- DUST SENSOR MODEL PPD42NS (D_n), COLUMN 3- ALCOHOL SENSOR (A_n).

T_n	D_n	A_n
20	110	570
21	110	575
20	110	578
21	110	581
21	110	582

The *LS-SVMlab Toolbox* [7] for *Matlab* was used for the initial training and learning based on the data acquired from the sensors. This was a preliminary step to evaluate the chosen classes and the reliability of the sensors. All preliminary experiments were carried out on the setup presented in Fig. 3 dividing the space into four distinct typical classes from *USAR* applications:

1. **Contamination** (X1)
   Air contamination means that alcohol sensor is above the 500mv. Contamination can be caused by gas, petrol, or some kind of alcohol container.
2. **Smoke** (X2)
   Smoke is detected for an output of the dust sensor above 20.000 particles, with at least 1 μm diameter in the reading area.
3. **Fire** (X3)
   Fire needs information from the dust sensor, the thermopile sensor, and the alcohol sensor. The dust sensor allows detecting smoke, the thermopile sensor the temperature gradient, and the alcohol sensor the type of fire (*e.g.*, fire emanating from a chemical explosion).

X1, X2, X3 are matrices with the test results for the different training cases. At least, a final class may be defined to assess the safe situation:

4. **Secure** (X4)
   This class was introduced to minimize the error of the classifier.

$$X = \begin{bmatrix} S1_{x1} & S2_{x1} & S3_{x1} \\ S1_{x2} & S2_{x2} & S3_{x2} \\ S1_{x3} & S2_{x2} & S3_{x3} \end{bmatrix} \; \mathrm{Y} = [\mathrm{Class}]$$

[7] http://www.esat.kuleuven.be/sista/lssvmlab/

For classification purposes, the on-the-fly data (*i.e.*, testing data) is represented as:

$$X_t = \begin{bmatrix} T_{t1} & D_{t1} & A_{t1} \\ & \vdots & \\ T_{tn} & D_{tn} & A_{tn} \end{bmatrix}$$

TABLE III shows the output variable. Every sample has a class matching from the training database, represented by the numbers 1, 2, 3 and 4 according to X1, X2, X3 and X4 previously described. After adding 20% noise to the training data X_i, *i.e.*, $X_i^t = 1.2 \times X_i$, with $i = \{1,2,3\}$, one may observe in TABLE IV that the *SVM* is still able to accurately identify each class.

TABLE III. TRAINING DATA: SAMPLES 899 AND 900 REPRESENT CONTAMINATION TRAINING USING ALCOHOL WHILE SAMPLES 901 AND 902 WERE RETRIEVED USING SMOKE TRAINING WITH PAPER BURNING.

Sample	T_n	D_n	A_n
899	23	0	642
900	22	0	642
901	24	26218	651
902	23	26218	653

TABLE IV. OUTPUT CLASSIFICATION MATCHES THE TRAINING DATA FROM TABLE III.

Sample	*Real Class*	*Estimated Class*
899	1	1
900	1	1
901	2	2
902	2	2

In TABLE V, the noise was incremented by 30% to the training data X_i, *i.e.*, $X_i^t = 1.3 \times X_i$, with $i = \{1,2,3\}$,. It is now possible to observe a classification error in 0 which the *SVM* incorrectly classifies class 2 by class 3 in some situations (TABLE VI).

TABLE V. TRAINING DATA, SAMPLES OF SMOKE TRAINING.

Sample	T_n	D_n	A_n
947	22	21402	610
948	23	21402	608
949	24	21402	607
950	23	21402	604
951	22	21402	602

Figure 4 illustrates the classification regions based on the two sensors that the *SVM* classifier judges as the most important, *i.e.*, the ones that presents more independency between themselves. The classifier assigns dust sensor and temperature sensor as the ones with more relevant differences between different classes.

TABLE VI. OUTPUT CLASSIFICATION.

Sample	*Real Class*	*Class*
947	2	2
948	2	2
949	2	3
950	2	3
951	2	2

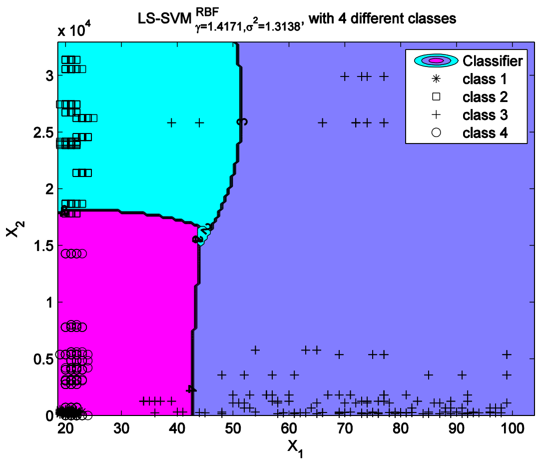

Figure 4. Classes representation.

V. Experimental Results

The same set of sensors presented in section IV was assembled in a *Pioneer-3DX* [20] and in a *TraxBot* [21] robot. The *Pioneer-3DX* is a well-known robotic platform for research and education from *ActivMedia*. The robot is a robust differential drive platform with 8 sonars in a ring disposition, a high-performance on-board microcontroller based on a 32-bit *Renesas SH2-7144 RISC* microprocessor, offering great reliability and easiness of use. The *Traxbot* is a small differential Arduino-based mobile platform, developed in our laboratory. As the *Pioneer-3DX*, this platform is fully integrated in the open-source *ROS* framework [9] and is capable to support a netbook on top of it [22]. Therefore, both platforms were extended with netbooks using *Ubuntu 11.10* operating system and the *ROS* framework with *Fuerte*[8] version on top of them. To explore the scenario, the robots were teleoperated using a *wiimote*[9] *ROS* node with the *Wii* remote controller.

The three sensors were assembled in an aluminium support mounted in the front of the robots (Fig. 5). This provides a better analysis by benefiting from the natural air flow generated by the robots' movements during the scenario exploration. Moreover, this configuration took into consideration a better horizontal positioning of the field of view for the thermopile array sensor. As stated at the end of section IV-B, to pre-process the received data from the sensors, an *Arduino Uno* board embedded within both platforms was used. The dust sensor was connected to a digital port, the alcohol sensor to an analogic port and the thermopile array sensor via *I2C Arduino* ports. The data exchanged between the *Arduino* board and the netbooks was handled using a *ROS* driver developed in our previous work [22].

a)

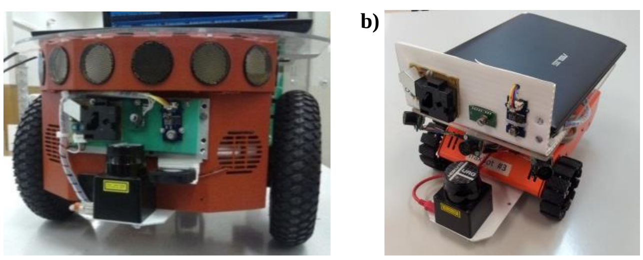

Figure 5. Robots eqquiped with the set of sensors: **a)** Pioneer-3DX; **b)** TraxBot.

[8] http://ros.org/wiki/fuerte
[9] http://www.ros.org/wiki/wiimote

A. Experiments with Mobile Robots

Some tests with mobile robots were conducted in an indoor scenario with 4.0 × 4.6 meters endowed with several obstacles (Fig. 6). Three points of interest were added in the experimental arena to simulate the necessary critical conditions for classification purposes. More specifically, the fire outbreak (Fig. 6a) was emulated using a 500 watts spotlight, ideal to produce heat, while the gas air contamination was simulated by inserting alcohol in an enclosed region within the scenario (Fig. 6b). Particles insertion for the assessment of the smoke class was not considered due to environmental constraints associated with the laboratory.

Figure 6. Real scenerio with three point of interest for *SVM* classification. **a)** Fire outbreak emulated using a 500 watts spotlight. **b)** Contaminated enclosed area with alcohol.

To directly classify the contextual information, the *ROS SVM* classifier *ml_classifier*[10] was used. The *SVM* classifier works in an online fashion based on the training data previously acquired (section IV). During the exploration mode, the *SVM* classifier was continuously running so as to detect the different classes. In the process, the acquired data from the set of sensors is streamed, as it can be observe in the *rxgraph ROS* tool (Fig.7).

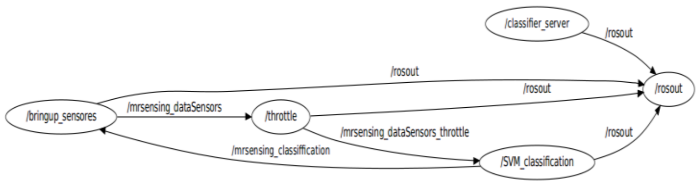

Figure 7. *ROS* topic SVM classification diagram provided by the ROS tool *rxgraph*.

In this experiments, a distributed *ROS* core systems for classification was implemented in each robot laptop. A third desktop computer running a *ROS* core network was added for analysis purposes. The map of the arena was considered to be known, *a priori*, for localization purposes by using an *AMCL*[11] algorithm. The *AMCL* is a probabilistic localization system that uses a particle filter to track the pose of the robot in the map. To that end, both robots were equipped with Hokuyo range finder laser.

The *ROS* 3D visualization tool *rviz*[12] was used for an augmented representation of the output classes. Figure 8a depicts virtual representation of the arena in *rviz* and the virtual model of the robots used in real test. Figure 8b represents the

[10] http://www.ros.org/wiki/ml_classifiers
[11] http://www.ros.org/wiki/amcl
[12] http://www.ros.org/wiki/rviz

ideal output of the classes on the virtual arena. This ideal representation was retrieved using the setup from Fig. 3b, in which the average value from 30 readings coming the set of sensors was considered for each 0.20 × 0.20 meters cell within the scenario for a total amount of 460 cells.

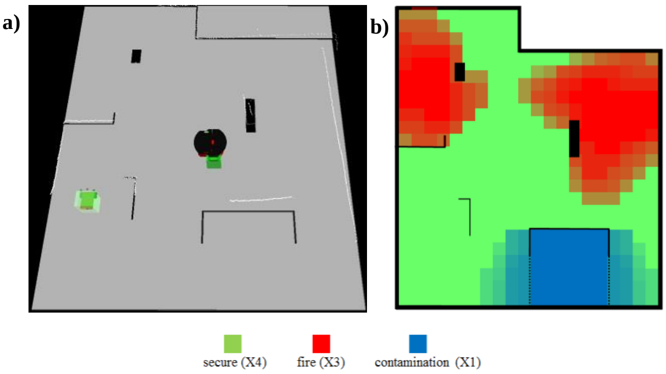

Figure 8. **a)** Virtual arena with two robots in *rviz*. **b)** Ideal represantiton of the classification regions.

The *rviz* representation of each class was achieved by filling the virtual arena with markers of different colors, according with the classification output sent from the *ml_classifier*. Green cells for secure cells (X4), blue cells for contamination cells (X1) and red cells for fire cells (X3). Then, the intensity of the color was defined to be proportional to the output value from the relevant sensor.

In Fig. 9 a comparison from the output of the tests with a single mobile robot and with two robots was considered, wherein one can observe the completeness of the mission after 3 minutes. For instance, in Fig. 9b the concentration of the output classes covers almost all the area of the arena, thus getting closer to the ideal representation from Fig. 8b.

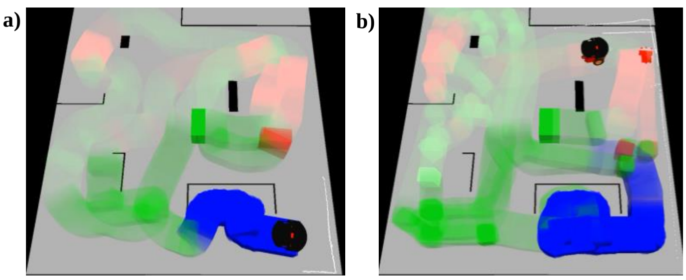

Figure 9. Output classification at 3 minutes of the runnig test with: **a)** One robot; **b)** Two robots.

This environmental mapping with one and two robots can be better perceived in the video of the experimental trials[13].

VI. Conclusion and Future Work

This work presented a multi-sensor setup to assess contextual information within mobile robotics platforms under catastrophic incidents. By validating this valuable approach in real platforms, the foundations were laid for a whole series of possible new multi-robot applications on *USAR* scenarios. Moreover, special attention should be given to the group communication architectures. Robots should be able to share information between themselves and teams of humans (*e.g.*, first responders) in an efficient way by communicating context commonly shared between teams of humans in such incidents.

[13] http://www2.isr.uc.pt/~nunoferreira/videos/SSRR2013/

References

[1] Z. LI, Y. Ma, A new method of multi-sensor information fusion based on SVM,in Proceedings of the Eighth International Conference on Machine Learningand Cybernetics, Baoding, 12-15 July 2009.

[2] D.L. Hall,J. Llinas, , "An introduction to multisensor data fusion", Proceedings of the IEEE, vol 85, issue 1, 1997.

[3] Shafer, Steven A.; Stentz, Anthony; and Thorpe, Charles E., "An architecture for sensor fusion in a mobile robot" (1986). Robotics Institute. Paper 585.

[4] Larionova, S., Marques, L., & de Almeida, A. T. (2006, October). Multi-Stage Sensor Fusion for Landmine Detection. In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on (pp. 2943-2948). IEEE.

[5] Dasarathy, Belur V. "Sensor fusion potential exploitation-innovative architectures and illustrative applications." Proceedings of the IEEE 85.1 (1997): 24-38.

[6] Gu, Jason, et al. "Sensor fusion in mobile robot: some perspectives." Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on. Vol. 2. IEEE, 2002.

[7] Julian, Brian J., et al. "Distributed robotic sensor networks: An information-theoretic approach." The International Journal of Robotics Research 31.10 (2012): 1134-1154.

[8] Micael S. Couceiro, David Portugal & Rui P. Rocha. "A Collective Robotic Architecture in Search and Rescue Scenarios", SAC2013 - 28th Symposium on Applied Computing, pp. 64-69, March 18-22, Coimbra, Portugal, 2013.

[9] M. Quigley, et al. "ROS: an open-source Robot Operating System" in Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA 2009), Kobe, Japan, May, 2009.

[10] H. Aliakbarpour, L. Almeida, P. Menezes, J. Dias, Multi-sensor 3D Volumetric Reconstruction Using CUDA, Publisher 3D Display Research Center, December 2011.

[11] X. Zhao, Q. Luo, B. Han , Survey on robot multi-sensor information fusion technology, Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on 25-27 June 2008.

[12] N. Xiong, P. Svensson, Multi-sensor management for information fusion: issues and approaches, FOI, S-172 90 Stockholm, Sweden 18 October 2001.

[13] C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery June 1998, Volume 2.

[14] R.Araújo, U. Nunes, L. Oliveira, P. Sousa, P. Peixoto , Support Vector Machines and Features for Environment Perception in Mobile Robotics,Coimbra,Portugal, 2008.

[15] De-Kun Hu ; Hui Peng ; Ju-Hong Tie ,Software Dept., Chengdu Univ. of Inf. Technol., Chengdu , A Multi-Sensor Information Fusion Algorithm based on SVM,Apperceiving Computing and Intelligence Analysis, 2008. ICACIA 2008, 13-15 Dec. 2008.

[16] R. Bravo, O. Christian, A. Salazar, Spastic hemiplegia gait characterization using support vector machines: Contralateral lower limb, Rev. Fac. Ing. UCV v.21 n.2 Caracas 2006.

[17] A.SHARMA , R. KUMAR , P.VARADWAJ, A. AHMAD , G. ASHRAF, A Comparative Study of Support Vector Machine, Artificial Neural Network and Bayesian Classifierer for Mutagenicity Prediction,Jhalwa, Allahabad, 211012, ttar Pradesh, India,2011 Jun 14.

[18] J. Miguel A. Luz , M. Couceiro, D. Portugal, Rui Rocha, 5 H. Araujo, G.Dias, Comparison of Classification Methods for Golf 3 Putting Performance Analysis, Coimbra,Portugal.

[19] Bishop, C. M. (2006). Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc. Secaucus, NJ, USA.

[20] S. Zaman, W. Slany and G. Steinbauer, "ROS-based Mapping, Localization and Autonomous Navigation using a Pioneer 3-DX Robot and their Relevant Issues", In Proc. of the IEEE Saudi International Electronics, Communications and Photonics Conference, Riad, Saudi-Arabia, 2011.

[21] A. Araújo, D. Portugal, M. Couceiro, C. Figueiredo and R. Rocha, "TraxBot: Assembling and Programming of a Mobile Robotic Platform". In Proc. of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), Vilamoura, Portugal, Feb 6-8, 2012.

[22] A. Araújo, D. Portugal, M. S. Couceiro and R. P. Rocha, "Integrating Arduino-based Educational Mobile Robots in ROS", In Proc, 13th International Conference on Autonomous Robot Systems and Competitions, Lisbon, April 2013.