Modular Design of Monitors for Cyber-Physical Systems from Formal Specifications

BY

RUGGERO BALTERI
B.S., Politecnico di Torino, Turin, Italy, 2012
B.S., Tongji University, Shanghai, China, 2013

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2015

Chicago, Illinois

Defense Committee:

Miloš Žefran, Chair and Advisor
Prasad Sistla
Luciano Lavagno, Politecnico di Torino

ACKNOWLEDGMENTS

This thesis would not have been possible without the individuals who extended their invaluable help, support, expertise, compassion and understanding in not only the development of this study, but in also my personal development academically, professionally and emotionally.

First, I want to thank professors Žefran and Sistla, whose precise guidance and constant encouragement helped me throughout the entire research project. Their unwavering support and understanding proved to be an invaluable asset for me throughout the entirety of this work.

My utmost gratitude must also be given to professor Lavagno for his great patience and extraordinarily steadfast remote support. Despite the distance, he constantly showed kindness, concern and consideration for this research, making himself available at a moments notice for additional help.

I am deeply grateful to my parents for having supported me throughout this year in the United States. Their love, help and constant encouragement gave me the chance to chase my dreams.

I would like to sincerely thank all my uncles, aunts and cousins in Sicily for being always present in my moments of need and for their unconditional support since the beginning of this adventure.

To my fellow Graduate student Eric Serra, without your enthusiasm and teamwork this research would have never been completed in time. I will always fondly remember how we were

able to overcome our problems, which looked impossible at first glance, through a meticulous step-by-step process that wouldn't have been possible without your exceptional work ethic.

To all my friends, old and new, from around the globe, thank you for your encouragement and understanding. I cannot list all of your names here, but you will always have a place in my heart.

To Dan, for helping me to revise this thesis work; for your great hospitality and for your genuine support during one of the worst periods of my life.

RB

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

CPSs	Cyber-Physical Systems
ROS	Robot Operating System
SONAR	SOund Navigation And Ranging
PID	Proportional Integral Derivative

SUMMARY

Robotics is a field that is subject to rapid evolution, with its ground-breaking innovations becoming increasingly more present in our lives. In the past, automation was designed to operate in a predetermined and completely controlled environment (e.g. assembly lines). The control system was sufficiently easy to model in an exhaustive way and reasonably less prone to unpredicted failures. Nowadays, the attention of the international community is shifting towards a new implementation of systems that can operate in an unstructured environment, inherently unpredictable [1] and where everything is subject to change.

In this framework, a complex system may have countless number of states and checking its correct behaviour against all the possible inputs has been proven, in general, theoretically undecidable [2] [3]. The probabilistic nature of the system and of the environment need to be considered. To monitor the correct behaviour of a system, an on-line algorithm [4] that takes into account a probability of transition (not only a deterministic transition) of the system model has to be employed in real-time.

The main goal of the entire research was to implement and test, on a realistic system, a valid monitoring methodology that could take into account the limits provided by inaccurate models and use that retrieved data to design better and more reliable systems. Chapter 1 gives an overall introduction of the main problems on which the research is based and summarizes all the main results obtained throughout the entire thesis work. Particular emphasis has been paid to implementing a physical platform that could be used as initial prototype. Once a working

robot has been correctly engineered, several monitor implementations have been designed and tested on it. All the retrieved data has been subsequently imported into Matlab and then analysed. In this way, it was possible to perform a number of tests to investigate several problems and, among the most relevant ones, the influence of inaccurate system models inside the monitor played a central role. In parallel, special attention was also given to the influence in performance and accuracy of alternative ways in implementing monitors.

CHAPTER 1

INTRODUCTION AND GENERAL OVERVIEW

Let us consider an autonomous system, characterized by a finite number of state modes and whose physical components are subject, with fixed probability, to a set of well-defined failures. The overall system can be monitored using the readings of the same sensors employed by the control system. The focal point of the entire research is based on the central conjecture, expressed in "Probabilistic robotics" [1]:

> *"A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not."*

In this framework, to properly take into account the influence of noise on the readings, an online state-estimation algorithm has been implemented. In fact, given a sufficiently complex system, guaranteeing its correctness against all possible stimuli can be almost impossible: it has demonstrated that is more reliable checking at runtime the correct behaviour of the system. One practical application of the goals of this research can be foresee in monitoring off-the-shelf components [5], where it is essential to construct a monitor that can detect any bad behaviour by also taking into account the interface specification of the component, provided by the manufacturer.

The first part of the research focused on the creation of a prototype that could represent a valid platform for the verification of the main monitoring theories. In the end, it was decided to

design a 4-wheel model of a small car, equipped with 4 electric motors (with encoder sensors) and 3 sonar sensors. The decision of selecting this specific kind of cyber-physical system was driven by the fact that, unlike other robots (e.g. quadcopters) its control system could be tested and debugged in a safe and more controlled way. Once a working prototype was built, its tasks, thus its main control algorithm, was formally modelled and then implemented. At last, a set of different monitors, whose characteristics had been previously and formally discussed, was correctly employed. The idea that lies behind is to properly develop a structured test methodology that can lead to a more accurate design of precise monitors. In case of cyber-physical systems, to check the validity of the correctness property [6], a reasonably accurate model of the robot must be employed inside the monitor. One of the main goals of this thesis is to investigate the effects of unmodelled states and show how those impact on the monitor performances.

Chapter 2 and 3 refers to the prerequisites that are needed for the research.
Chapter 2 focuses on the theoretical background on which the research is based. Particular emphasis has been paid to the theory behind Markov Chains, Bayes' Filters and monitorability theory. Chapter 3 gives a short introduction about the Robot Operating System (ROS), a robotic middle-ware program, that allows several abstraction layers, providing an operating system-like functionality.

Chapter 4 and 5 focus on the overall design of the first working prototype. Chapter 4 describes, in great detail, the physical implementation of the device, starting from the main structure and individual components, up to the final design of the "obstacle avoidance" al-

gorithm that remotely and automatically (using data coming from the sensors) controls the robot's trajectory. Chapter 5 explains, from the implementation point of view, the characteristics of the monitor. Particular focus is given to its actual algorithm, based on particle filter, developed using ROS libraries.

Finally, Chapter 6 gives an in-depth exploration of more advanced concepts. In particular, the influence of discrepancies between the system model and the actual system is taken into account with respect to the main accuracy of the monitor. The new concept of "monitor error" is introduced, signalling a high probability that an unmodelled situation is occurring. In addition, in this chapter a new monitor that models transient modes of the system is analysed. In parallel, new types of failures ("malfunctioning of the electric motors") are taken into account together with precise analysis about the monitor accuracy and its behaviour with noisy observations. Last, Chapter 7 summarizes all the obtained results, focusing on the implications of the entire research and on future developments.

CHAPTER 2

INTRODUCTION TO PROBABILITY THEORY AND CYBER-PHYSICAL SYSTEM

A Cyber Physical System (CPS) is a system that combines communication, computation and controls. Each CPS can have Physical variables that define its realistic behaviour (e.g. Differential Equations) and Cyber variables that are used as internal representations (e.g. State Machine).

All physical systems have transients that must be correctly modelled to represent the system. In this framework, noise can be considered a transient and be described as a probability distribution.

2.1 Markov Chains

The study of Markov chains is extremely relevant to the area of research. It sets a close relationship with optional mathematical structures:

- Graphs
- Matrices
- Stochastic processes

Stochastic processes are mathematical models that describe in a probabilistic way the evolution in time of a specific physical phenomenon [7]. Let us define the stochastic variable X, defined by a set of values Ω and by a distribution density of probability:

$$\int_a^b p(x)dx = 1 \quad x \in [a, b] \tag{2.1}$$

A stochastic process X_t is defined with the following notation:

$$\{X_t, t \in T\} \tag{2.2}$$

X_t is a collection of variables characterized by the parameter t. For each value of t there is a different set of values X.

Let us define $X_t \in \Omega$ as the space of the states of X and x_t the state at time t, then a specific succession s of states X, namely $\{X_t(s), t \in [-\infty, +\infty]\}$ is called *trajectory of the process.* Note that all of these trajectories may not have the same probability.

Each stochastic process is unequivocally defined (as a family of trajectories) when it is related with a density probability $f(x_1, \ldots, x_n)$ of a finite set of values $x(t_i), i = 1, \ldots, n$ for each possible choice of $t_1, \ldots, t_n$ where n is a finite number. In this framework an independent process is defined as:

$$f(x(t_1), \ldots, x(t_n)) = \prod_{i=1}^{n} P_X(x(t_i) \tag{2.3}$$

A stochastic process with discrete time ($t = 1, 2, 3, \ldots, n \quad , n \in \mathbb{N}$) is a Markov chain and for all the states the conditional distribution of any possible future state depends only on the present state and not on the past states, that is:

$$P(X_{t+1} = i_{t+1} \mid X_t = i_t, X_{t-1} = i_{t-1}, \ldots, X_1 = i_1, X_0 = i_0) = P(X_{t+1} = i_{t+1} \mid X_t = i_t) \quad (2.4)$$

If the Markov chain is independent from time then it is called a time-homogeneous Markov chain:

$$P(X_{t+1} = j \mid X_t = i) = p_{ij} \quad (2.5)$$

where p_{ij} identifies the probability of transition from time t to time $t + 1$ from $i \to j$.

If the set of all possible states Ω is finite, then the probability p_{ij} can be represented as a square matrix, whose order is equal to the cardinality of the set Ω.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nn} \end{bmatrix} \qquad p_{ij} \geq 0, \forall i, j \in \Omega$$

Since:

$$\sum_{j \in \Omega} p_{ij} = 1, \forall i \in \Omega \quad (2.6)$$

P is a row-stochastic matrix (or right-stochastic: each row summing to 1).

Let us consider an example:

$$x_{k+1} = f(x_k, u_k, w_k) \tag{2.7}$$

where f is the function that describes the evolution of the system from state x_k to the state x_{k+1}, u_k is the control and w_k is noise (probability distribution). Let σ_0 be the probability distribution of the state x_0. It can be expressed as $\sigma_0 = \{\rho_0, \ldots, \rho_n\}$. You can express the generic σ_k that represent the generic probability of being in the state k as:

$$\sigma_k = \sigma_{k-1} \cdot P = \sum_{i=1}^{n} \rho_i \cdot p_{ik} \tag{2.8}$$

where P is the transition matrix.

Let us consider a realistic example. A company produces two kinds of beverages, CC1 = 1 , CC2 = 2. Among the consumers, it has been statistically proven that those who buy CC1 tend to buy it again 90% of the time. On the other hand, those who buy CC2 tend to buy it again just 80% of the times. All the data can be summarized in the following way:

$$P = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \qquad \Omega = \{x_1, x_2\} = \{1, 2\} \tag{2.9}$$

If a consumer buys CC2, what is the probability that after two purchases he will buy CC1?

To answer this question we need to introduce the Chapman-Kolmogorov equations. Remember that if a Markov chain is in the state i at time m, then the probability that after n

steps it will be in the state j is given by the Equation 2.5. The Chapman-Kolmogorov equations allow us to get the transition matrix of the system after n steps.

$$P_{ij}^{n+m} = \sum_{k} P_{ij}^{n} \cdot P_{ij}^{m} \tag{2.10}$$

$$\bar{P}^{n+m} = P^{n} \cdot P^{m} \tag{2.11}$$

The resulting matrix coincides with the nth power of the transition matrix.

$$\bar{P}^{2} = \bar{P}^{1+1} = \bar{P}^{1} \cdot \bar{P}^{1} = \bar{P} \cdot \bar{P} \tag{2.12}$$

To answer the previous question, ("If a consumer buys CC2, what is the probability that after two purchases he will buy CC1?"), we need to solve:

$$P(X_2 = 1 \mid X_0 = 2) = P_{21}^{2} \tag{2.13}$$

$$P^{2} = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \cdot \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix} \tag{2.14}$$

In this case the answer is 0.34.

Let us now consider another situation: if a consumer buys CC1, what is the probability that after three purchases he will buy CC1 again?

$$P(X_3 = 1 \mid X_0 = 1) = P_{11}^{3} \tag{2.15}$$

$$P^3 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \cdot \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix} = \begin{bmatrix} 0.781 & 0.219 \\ 0.438 & 0.562 \end{bmatrix} \tag{2.16}$$

The answer is 0.781.

A more interesting question would be to ask what is the percentage of people that would buy CC1. In order to solve this problem the initial distribution must be know a priori. Let us define $\pi_m^{(k)}$ as the probability of being in state m at time k. In our case we have:

$$\pi_1^{(1)} = \pi_1^{(1)} p_{11} + \pi_2^{(0)} p_{21} \qquad \pi_1^{(1)} = \sum_{m=1}^{2} \pi_m^{(0)} p_{m1} \tag{2.17}$$

$$\pi_2^{(1)} = \pi_1^{(0)} p_{12} + \pi_2^{(0)} p_{22} \qquad \pi_2^{(1)} = \sum_{m=1}^{2} \pi_m^{(0)} p_{m2} \tag{2.18}$$

In a similar way we can obtain:

$$\pi_n^{(1)} = \sum_{m=1}^{2} \pi_m^{(0)} p_{mn} \quad \pi_n^{(2)} = \sum_{m=1}^{2} \pi_m^{(1)} p_{mn} \tag{2.19}$$

After t steps we have:

$$\pi_n^{(t+1)} = \sum_{m=1}^{2} \pi_m^{(t)} p_{mn} \tag{2.20}$$

or in vectorial form:

$$\bar{\pi}^{(t+1)} = \bar{\pi}^{(t)} \cdot \bar{P} \tag{2.21}$$

Let us now consider another example with the following transition matrix and initial distribution:

$$P = \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix} \qquad \bar{\pi}^{(0)} = \begin{bmatrix} \pi_1^{(0)} & \pi_2^{(0)} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \tag{2.22}$$

we can now compute the distribution at time $t = 1$ and $t = 2$

$$\bar{\pi}^{(1)} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix} = \begin{bmatrix} 0.45 & 0.55 \end{bmatrix} \tag{2.23}$$

$$\bar{\pi}^{(2)} = \begin{bmatrix} 0.45 & 0.55 \end{bmatrix} \cdot \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix} = \begin{bmatrix} 0.435 & 0.565 \end{bmatrix} \tag{2.24}$$

It can be noticed that those values are stable for the successive time steps.

Given a generic Markov Chain $G = \{v, \epsilon\}$, where $v = \{1, \ldots, n\}$ is the set of vertices and ϵ is the set of edges, we can now define:

- Walk: a walk is a sequence of nodes $s_0, \ldots, s_n$ such that $(s_i, s_j) \in \epsilon$
- Path: a path is a walk where no node is repeated. It is trivial to notice that the longest path can not be bigger than the number of nodes n.
- Cycle: a cycle is a walk where only the first and the last node coincides.

We can now classify the nodes (states) of a Markov chain in the following way:

- A node j is accessible from i ($i \neq j$) if and only if there exists a walk from i to j.
- Two states i and j communicates if j is accessible from i and vice-versa.

- Nodes form a class C if and only if each node that belongs to C communicates with every node in the class, but no other nodes. Classes partition the states: each node belongs to only one class.
- A state i is said to be transient if and only if it exists a state j accessible from i, but not accessible from j
- A non-transient state is called a recurrent state
- A state i is periodic (having period $k > 1$) if k is the smallest number such that all the paths from state i return to i have a length that is a multiple of k.
- If a state is non-periodic, it is called aperiodic.
- If all the states in a Markov chain are recurrent, aperiodic and communicates with each other then the Markov chain is called ergodic.

2.2 Hidden Markov Model

Let us suppose we want to build a Markov chain that estimates weather conditions by looking at past trends in the following accessories: gloves, sunglasses and umbrellas (Figure 1). Let us consider that there is a relationship (fixed probability) between the purchases of those accessories respectively to snow, rain or sunny weather. At the same time, the weather transitions can be modelled using a traditional Markov chain. In order to find a solution to this problem, we need to introduce the new concept of hidden Markov models (HMMs) [8].

In a hidden Markov model [9] each state of a traditional Markov chain is related to an observable event by means of a probabilistic function. In other words, the resulting model is

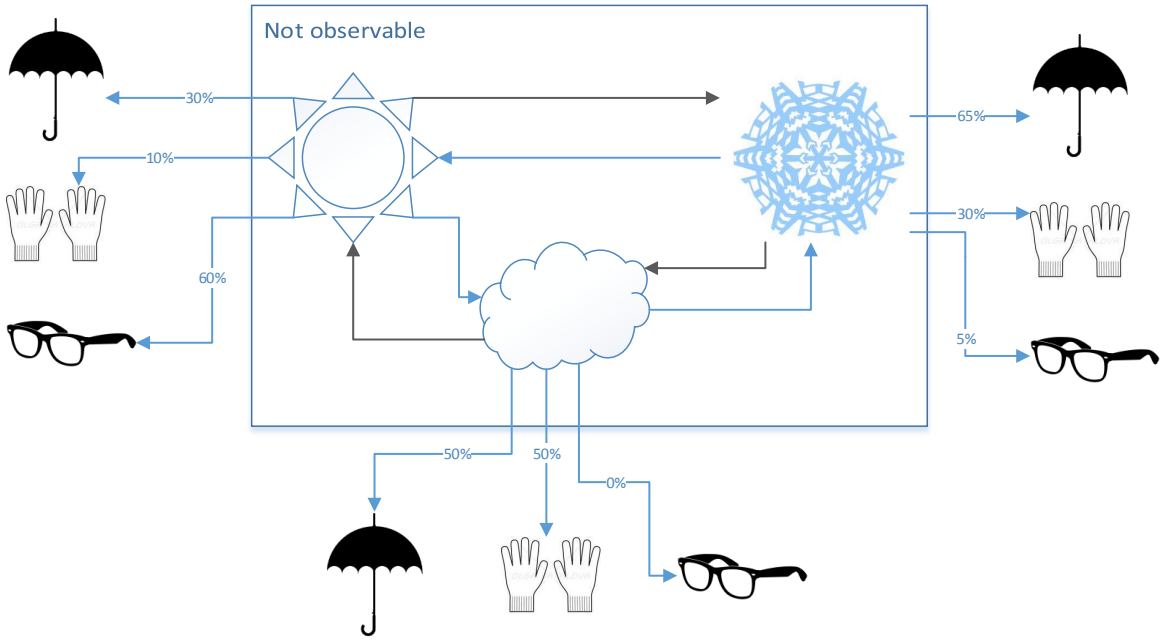


Figure 1: Example of an HMM

not a directly observable stochastic process that generates an observable output in a stochastic way [10]. A hidden Markov model can be described in the following way:

- The set of states of the Markov chain: $x \in V = \{v_1, \ldots, v_N\}$
- The probability vector at time $t = 0$

$$\pi_0 \in \mathbb{R}, \quad \pi_0^{(0)} \geq 0 \exists i \quad \sum_{i=1}^{N} \pi_0^{(i)} = 1 \tag{2.25}$$

- The transition matrix

$$P \in \mathbb{R}^{N \times N} \tag{2.26}$$

- The set of observable outputs

$$y \in W = \{w_1, \ldots, w_M\} \tag{2.27}$$

- The emission matrix $C \in \mathbb{R}^{N \times M}$, where each element c_{ij} represents the probability of observing the output w_j given that the Markov chain is in the state v_i

$$c_{ij} = p(y_t = j \mid x_t = i) \geq 0 \qquad \sum_{j=1}^{M} c_0 = 1 \tag{2.28}$$

Let us define a new concept related to hidden Markov models: the observation. It refers to the set of ordered outputs observed in sequence. We can now define $Y_h = (y_0, \ldots, y_h)$ as the set of outputs observed in $t+1$ steps. Under such conditions we can now formulate a number of problems:

- Let us consider $p(x_t \mid Y^h) = p_{\hat{t|h}} \in \mathbb{R}^N$ as the probability that the chain is in the state x_t under the observations Y^h. We can now look for the state that maximizes such probability (forward algorithm), that is:

$$\hat{x}_{t|h} = \arg \ max_i(\hat{p}_{t|h}) \tag{2.29}$$

- We can search for the set of states that have that maximum probability of being crossed depending on the observations (backward algorithm)

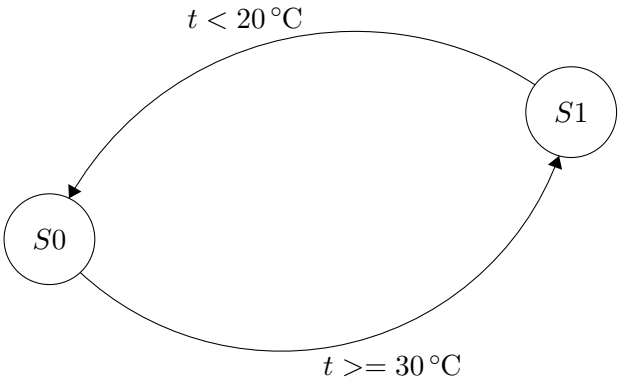
$$\hat{x}^t_{t|h} = (\hat{x}_{0|h}, \ldots, \hat{x}_{t|h}) \in \underbrace{V \times \ldots V}_{t+1 \ times} \tag{2.30}$$

2.3 Example of Transition Systems

Let us define:

- V: set of variables whose values are related to a particular domain
- A: set of states $A_{1...k}$
- C: set of conditions $C_{1...k}$
- T: set of transitions (e.g. $C_1 \rightarrow A_1$, $C_2 \rightarrow A_2$,..., $C_k \rightarrow A_k$)
- θ: initial condition that, if verified by a state, can be considered as an initial state.

Let us consider an example with discrete time. There is a closed room with an heater and a automatic window. If the windows is closed, the heater increases the temperature of the room by 3 °C every time step (state $S0$). If the window is open, for each time step the cold air from the outside decreases the temperature of the room of -5 °C (state $S1$). The system is automatic and has hysteresis: if the temperature of the room is greater or equal than 30 °C the windows is opened, whereas if it is less than 20 °C the heater is turned on. The system modes are represented in 2.3.



The set of transitions of the system can be summarized in the following way:

$state = S0 \wedge p < 30\,^{\circ}\mathrm{C} \Rightarrow t := t + 3$

$state = S0 \wedge p \geq 30\,^{\circ}\mathrm{C} \Rightarrow t := t + 3 \wedge state = S1$

$state = S1 \wedge p \geq 20\,^{\circ}\mathrm{C} \Rightarrow t := t - 5$

$state = S1 \wedge p < 30\,^{\circ}\mathrm{C} \Rightarrow t := t - 5 \wedge state = S0$

2.4 Bayes' Filter

The Bayes' theorem has a fundamental importance in the study of probabilistic models [11]. Let us consider two variables: x, the state of the system and y, the observation of the system. The chain rule (also called the general product rule) can be used to derive the density function:

$$P(x, y) = P(x \mid y)P(y) = P(y \mid x)P(x) \tag{2.31}$$

The Bayes' theorem can be easily obtained from the Equation 2.31:

$$P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)} \tag{2.32}$$

In the Equation 2.32 the marginal distribution $P(y)$ is used to normalize the posterior distribution:

$$\int_x P(x \mid y)dx = \int_x \frac{P(y \mid x)P(x)}{P(y)}dx = \int_x \frac{P(x \mid y)}{P(y)}dx = \int_x \frac{P(y)}{P(y)}dx = 1 \tag{2.33}$$

The Bayes' theorem is then a direct method to combine the observed output with the previous estimation of the probability density associated to $P(x)$. In this way it is possible to obtain a new density probability that takes into account the measure y. In other words, the new information obtained throughout observation is used to evaluate the probability distribution associated with the state x with respect to the prior probability density function.

Let us take a closer look about the functions $P(y \mid x)$ and $P(x)$:

- $P(x)$ is the prior probability density function: it is related to the uncertainty with which you know the expected value of a state, before the observation y.
- In order to obtain further information about the state x, a new observation y is performed. All observations are modelled using a conditional probability density function $P(y \mid x)$ that describes for each state $x \in X$ the probability of obtaining the observation $y \in Y$ (the probability of measuring y given x). This conditional probability plays an essential role in designing the model of a sensor.

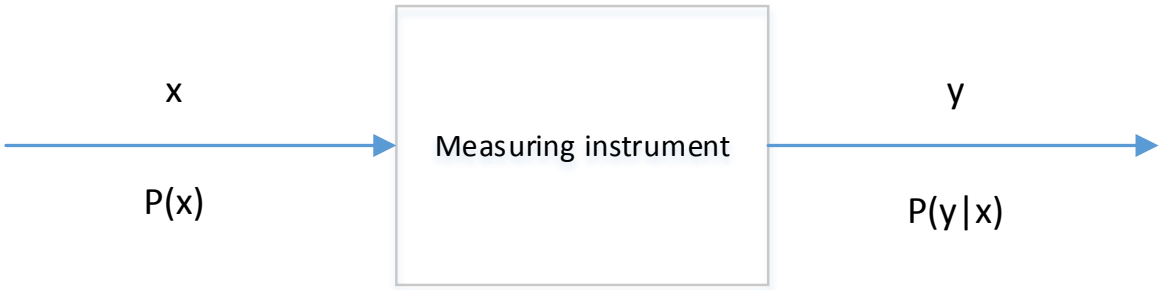


Figure 2: Model of a sensor

Let us now focus the attention to the implementation of the Bayes' filter by means of recursion. Given $Y^k \triangleq \{y_k, Z^{k-1}\}$ we have:

$$P(x, Y^k) = P(x \mid Y^k)P(Y^k) = P(y_k, Y^{k-1} \mid x)P(x) = P(y_k \mid x)P(Y^{k-1} \mid x)P(x) \tag{2.34}$$

Since $P(Y^k) \setminus P(Y^{k-1}) = P(y_k \mid Y^{k-1})$ we can easily get:

$$P(x \mid Y^k) = \frac{P(y_k \mid x)P(x \mid Y^{k-1})}{P(y_k \mid Y^{k-1})} \tag{2.35}$$

The main advantage of having this recursive algorithm lies in the fact that in each iteration step only the posterior distribution $P(x \mid Y^k)$ has to be saved. Each time a new observation y_k is read together with a new probability function $P(y_k \mid x)$, the posterior distribution becomes the prior one and the two probability functions, once multiplied and normalized, become the new posterior distribution. This posterior distribution that takes into account all the past sensor readings and control is called *belief*.

In robotics, the Bayes filter can be described in a similar way. There are two main steps:

- Prediction: given the $bel(x_{k-1})$, u_k (input) and $p(x_k \mid u_k, x_{k-1})$ (model of the system) the Bayes' filter computes the partial belief $\overline{bel}(x_k)$:

$$\overline{bel}(x_k) = \int_{x_{k-1}} p(x_k \mid u_k, x_{k-1}) \cdot bel(x_{k-1}) dx_{k-1} \tag{2.36}$$

- Correction: given $\overline{bel}(x_k)$ (predicted belief), y_n (sensor reading) and $p(y_k \mid x_k)$ (sensor model) the final belief (base rule) can be computed:

$$bel(x_k) = \eta p(y_k \mid x_k)\overline{bel}(x_k) \tag{2.37}$$

where η is a normalization constant. Substituting Equation 2.31 into Equation 2.37 it yields to the Bayes recursion equation:

$$bel(x_k) = \eta \cdot p(y_k \mid x_k) \cdot \int_{x_{k-1}} p(x_k \mid u_k, x_{k-1}) \cdot bel(x_{k-1}) dx_{k-1} \tag{2.38}$$

In most of the situations, it is not possible to obtain an analytical solution to Equation 2.38 and during the whole research the approximated numerical solution was computed using the Particle filter algorithm.

2.5 Particle Filter

As underlined in the paper "Particle filters for state estimation of jump Markov linear systems" [12]:

> *"Increasingly, for many application areas, it is becoming important to include elements of nonlinearity and non-Gaussianity in order to model accurately the underlying dynamics of a physical system."*

Particle filter is a recursive Bayesian filter that successively approximate the inner state of the system [13]. It can be applied to both linear and non-linear systems and trace multiple

hypothesis about the value of the state. If the characterization of the target object is not sufficient, then the filter may not give accurate estimations.

In a real situation the transition between states is always stochastic. Indeed, to monitor the correct behaviour of the control system you need a recursive algorithm that takes into account a probability of transition (not only a deterministic transition) and estimate a set of posterior states. This is accomplished using the Particle filter.

2.5.1 The basic algorithm

The goal of the particle filter is to approximate the posterior probability by means of discrete distributions made from a set of samples $\{x_k^i\}_{i=1}^{N_s}$ in the space of states (N_s particles) and a set of associated weights $\{w_k^i\}$. Let us consider:

$$p(x_k \mid y_{1:k}) \approx \sum_{n=1}^{N_s} w_k^i \delta(x_k - x_k^i) \tag{2.39}$$

where $\delta(x_k - x_k^i)$ is the Dirac function and $\approx$ relates to the fact that the equation makes a discrete approximations of the continuous posterior probability density function. The main problem is now to properly propagate the particles and evaluate the right weights. The distribution of particles in the space is not necessarily uniform. Different distribution of particles can lead to different formulas to evaluate the weights. A key function is played by the sampling process that guarantee the correctness of the approximation process; its reliability improves incrementing the number of particles N_s.

Let us consider a system whose evolution depends only on the value of the previous state and current sensors y_{k-1}. To properly model its evolution you need to also take into account the effects of noise v_{k-1} that affects the reading of the sensors. You have:

$$x_k = f_k(x_{k-1}, y_{k-1}, v_{k-1}) \tag{2.40}$$

As assumption, we consider that the probability $p(x_0 \mid y_0)$ is available. The posterior probability density function $p(x_k \mid y_{1:k})$ is then evaluated recursively in the following steps:

- Sampling: during this stage the analogue readings (affected by noise) are sampled from the sensors.
- Prediction: the algorithm propagate the state of the single particle. The probability $p(x_{k-1} \mid y_{1:k-1})$ at time k is given. The particle filter translate, deforms and enlarge the probability density function to take into account the effects of noise
- Weight evaluation: each particle carries values that are compared with the real reading and then a weight $\{w_k^i\}$ is estimated. All the weights are then renormalized, so that:

$$\sum_{n=1}^{N_s} w_k^i = 1 \tag{2.41}$$

- Re-sampling process: each particle is now drawn with probability proportional to its related weight $\{w_k^i\}$

2.6 Introduction to monitorability theory

At present, the increase in complexity of CPSs (Cyber-Physical Systems) is growing at a optimal fast rate. The inner control system is becoming more evolved and can include a countless number of states. In addition, automata are now required to operate in environments that are no longer controlled [1]. In the past, most of the robots were employed in a "safe" and well-defined place (e.g. identical tasks carried out in assembly lines). Currently, the new generation of robot systems has to operate in an environment that can be subject to rapid changes and it is inherently less predictable. In this framework, exhaustively testing CPS to verify that specific properties are not violated (e.g. keep a minimum distance between objects) in all possible situations may not be a trivial problem.

A fail-safe procedure could be based in the output of monitors that by definition are systems completely independent from the control systems of the automata. In this way a possible failure of the main algorithm or of physical components can be easily detected.

The control system of the robot is not directly observable, and throughout the entire research its behaviour is considered stochastic.

2.6.1 Initial definitions

Before introducing more advanced topics it is better to give formal definitions related to monitorability theory. S is a finite set of possible states. S^{ω} identifies an infinite set of infinite states. Let us define $\sigma = s_0, s_1, \ldots, s_n \in S$ as an infinite sequence in the finite set S. We have:

$$\forall i \in \mathbb{N} \Rightarrow \sigma[0, i] = \alpha \tag{2.42}$$

where α identifies the prefix of σ up to s_i. Note that given α_1 and α_2 as two finite sequences, then $\alpha_1\alpha_2$ is the concatenation in that order of the two sequences.

An arbitrary Markov chain G is identified by the following 3-tuple: $G = (S, R, \phi)$, where R represents a set that includes all the branches between states satisfying a binary relation $\forall s \in S \rightarrow \exists t \in S : (s,t) \in R$ and ϕ is the probability function that describes each transition $\forall s \in S \rightarrow \sum_{(s,t)\in R} \phi(s,t) = 1$.

An arbitrary Hidden Markov chain H is identified by the following 3-tuple: $H = (G, O, r_0)$, where G is a Markov chain, r_0 is the initial state and O is the output function that is: $O : S \rightarrow \Sigma$ where Σ is the set of symbols of the system. Note that the outputs of a sequence are equal to the sequence of individual output of each state of the sequence, i.e. $O(s_1, s_2, \ldots, s_n) = O(s_1)O(s_1)\ldots O(s_n)$..

Let us give a better definition of a monitor. Given the monitor M:

$$M : \Sigma^* \rightarrow \{0,1\} M(\alpha) = 0 \tag{2.43}$$

where $\alpha \in S$, then all the extensions β of α are rejected as well:

$$\beta \in \Sigma^* \rightarrow M(\alpha\beta) = 0 \tag{2.44}$$

In an analogue way, the same concept can be extended to infinite sequences Σ^ω, that is $\forall\beta \in \Sigma^\omega, M(\beta) = 0$ if there exists a prefix $\beta\prime$ of β such that $M(\beta\prime) = 0$. Given an output sequence that is *bad* then all its extensions are *bad* as well. A safety property can then be defined:

$$P = \{\beta \in \Sigma^\omega \mid M(\beta) = 1\} \tag{2.45}$$

Let us now consider the definition of a Quantized Probabilistic Hybrid Automata A [14] [15]:

$$A = (Q, V, \Delta t, \epsilon, T, c_0) \tag{2.46}$$

where Q is a state (finite) of nodes, V is set of continuous output and noise variables, Δt is the sampling time, ϵ describes the evolution of the continuous state and output, T identifies the transition $\forall q \in Q$ and c_0 is the initial discrete state. We can now identify the correctness property by an automaton A over the input alphabet S. $L(A)$ is the language of the automaton and represents all the system executions that are *good*, i.e.

$$L(A) \subseteq S^\omega \tag{2.47}$$

If the monitor detects a system state with an output that is not in $L(A)$ then it raises an alarm (e.g. flag). Let $L(M)$ be the set of all infinite sequences accepted by M, then $L(M)$ is defined as a *safety property*.

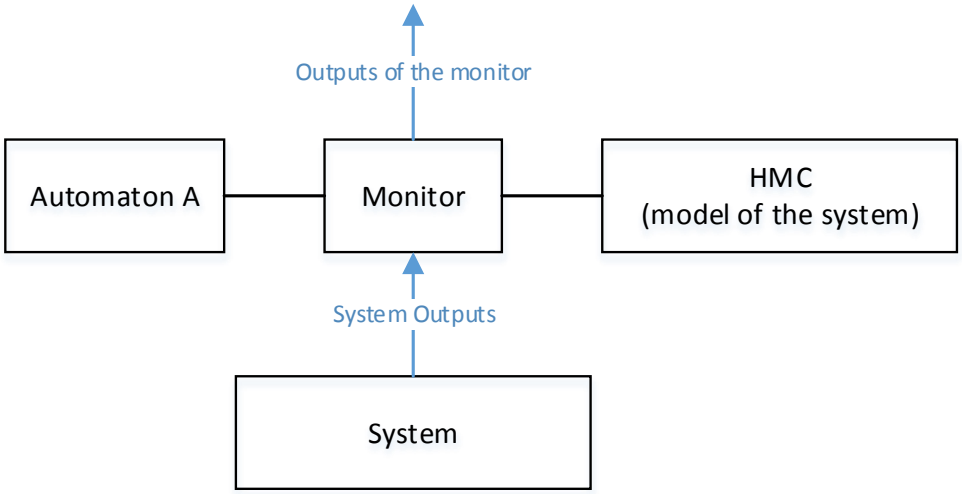


Figure 3: A basic diagram to show the role of the monitor

2.6.2 Monitorability theory

Let us consider the following definitions previously introduced in [14],[16],[17]:

Definition 1. *A monitor M is a function that recursively checks the validity of a correctness property and if violated by the system it raises an alarm.*

In order to better describe the monitors, two new measures have to be introduced in [18]:

Definition 2. *Acceptance Accuracy (AA) of a hidden Markov chain H with automaton A is the probability that a sequence of the system is accepted by the monitor M given that the underlying*

computation of the system is good, *i.e.* $\in L(A)$. *AA is used to represent the probability of having false alarms, i.e.* $(1 - AA)$.

$$AA(H, A, M) = \frac{g_a}{g_a + g_r} \tag{2.48}$$

where g_a *is the number of* GOOD *runs of the system that were accepted and* g_r *is the number of* GOOD *runs of the system that were rejected.*

Definition 3. *Rejection Accuracy (RA) is the probability that a sequence of the system is rejected by the monitor given that a computation is* bad, *i.e.* $\notin L(A)$. *RA is used to represent the probability of having missed alarms, i.e.* $(1 - RA)$.

$$RA(H, A, M) = \frac{b_r}{b_a + b_r} \tag{2.49}$$

where b_r *is the number of* BAD *runs of the system that were rejected and* b_a *is the number of* BAD *runs of the system that were accepted.*

2.6.3 Monitorability and strong monitorability

Definition 4. *A system is said to be strongly monitorable [19] with respect to a correctness property A (Street Automaton) if there exists a monitor M such that:*

$$AA(H, A, M) = RA(H, A, M) = 1 \tag{2.50}$$

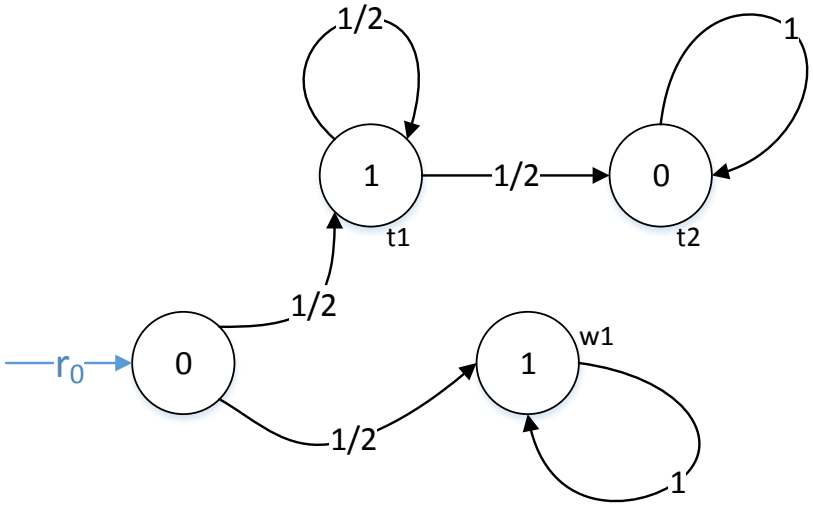


Figure 4: A basic diagram of a strongly monitorable system

Let us take a look at Figure 4. Let us consider that the correctness property rejects state $t1$ and $t2$. Then the system is strongly monitorable because for $t \to \infty$ the output is either 1 ("GOOD" prefixes) or 0 ("BAD" prefixes). The following property is then verified:

$$M(0\ 1^k\ 0) = 0, \quad \forall k > 0 \qquad alarm \tag{2.51}$$

$$M(0\ 1^k) = 1, \quad \forall k \geq 0 \qquad no\ alarm \tag{2.52}$$

Note that both rejection accuracy and acceptance accuracy are equal to 1.

Let us now focus on Figure 5. Given as before that the correctness property reject state $t1$ and $t2$, then it is trivial to notice that the system is not monitorable since the outputs of all states are 0.

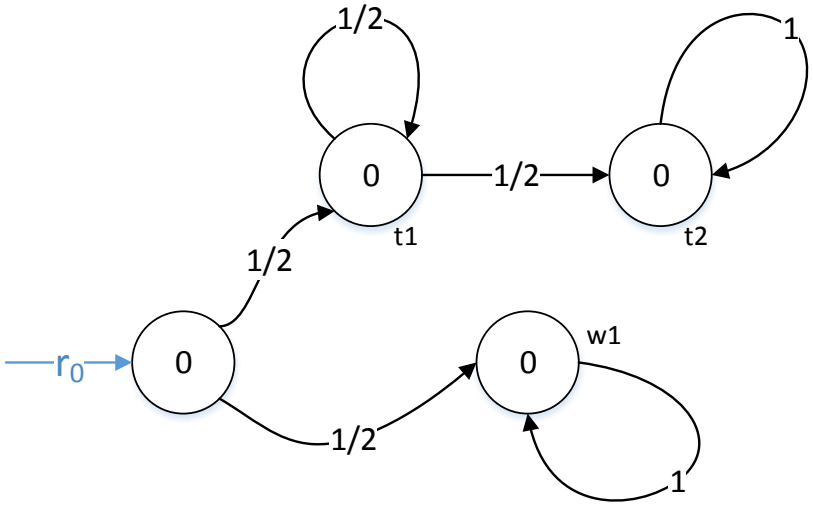


Figure 5: A basic diagram of a not monitorable system

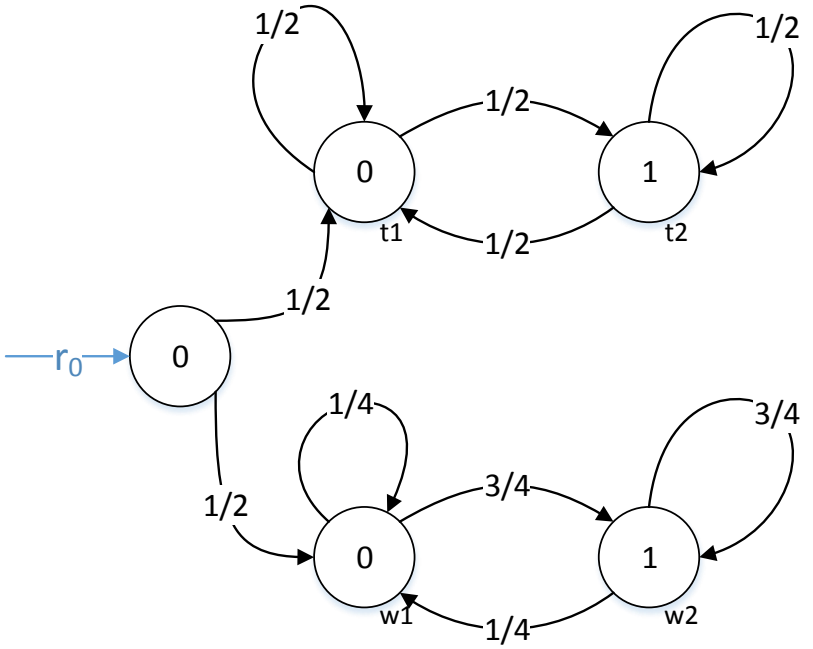


Figure 6: A basic diagram of a monitorable system

Figure 6 shows a monitorable system. Let us consider the state $t1$ and $t2$ as BAD computations. It can be noticed that the outputs of the BAD states $t1$ and $t2$ are equal respectively to the outputs of the $GOOD$ states $w1$ and $w2$. It can be proven that this system is not strongly monitorable. By applying the weak law of great numbers it can be shown that for $k \to \infty$ if the system goes to the set of states $t0$ $t1$ then the average probability of having $output = 0$ is approximately $1/2$, whereas for the set of states $w0$ $w1$ then the average probability of having $output = 0$ is approximately $1/4$.

Definition 5. *A system H is monitorable with respect to the correctness property A if $\forall x \in (0,1)$ exists a monitor M such that:*

$$AA(H, A, M) \geq x \quad and \quad RA(H, A, M) \geq x \tag{2.53}$$

2.6.4 Internal and external monitoring

Let us consider the following system H that takes output $\Sigma = \{0, 1\}$

Let us now define:

- External monitoring: the correctness property is specified on the sequence of outputs. In the example shown in Figure 7, we have $\boxdot 0$: the state $Q3$ is the only error state since its output is 1. In most cases external monitoring is easy to check since there is no state estimation, but rather a simple match between the real and the expected outputs.
- Internal monitoring: in this situation the correctness property is specified on the state sequences of the system. For instance, $M : state \to Q1$. In general, internal monitoring

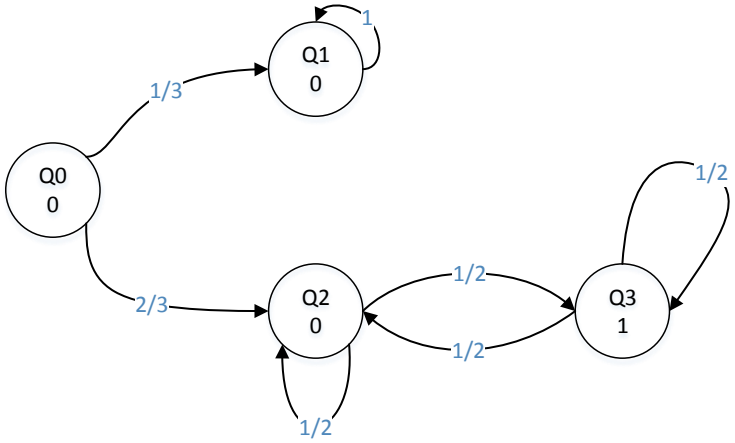


Figure 7: A simple system H

is harder to estimate since the state of the system has to be estimated by looking at the outputs. In the example shown in Figure 7, both state $Q2$ and state $Q1$ have the same outputs, however with the evolution of the system it will become clear where the system really is since the output of $Q3$ is 1.

CHAPTER 3

ROBOT OPERATING SYSTEM (ROS)

3.1 Introduction

ROS (Robot Operative System) is a set of libraries and functions (under BSD license[1]) that allows easy development of robot applications. The initial version of ROS was released in 2007, but the first stable version was officially made public on January 2010. ROS is written in `C++` and Python and currently runs only on Ubuntu Linux.

ROS as robotic middle-ware allows several abstraction layers, providing an operative system-like functionality.

3.2 Technical Overview of ROS

3.2.1 Basic Concepts

The Community level, the Computation Graph level and the File-system level are the three levels of concepts of ROS.

The Community level is the set of online ROS resources that allows users to share ROS code and knowledge. Specifically they includes:

- Distributions: Over the years several ROS distributions have been released. The most recent ones are Indigo (2014), Hydro (2013) and Groovy (2012). So far, most of the

[1]BSD was originally related to Berkeley Software Distribution, nowadays it refers to a set of open-source software licences.

ROS releases are not backward compatible. During the research, ROS Indigo was used to implement all the ROS nodes.

- ROS wiki: The primary and official forum of ROS. It is free and anyone can contribute their own tutorials, updates and corrections.
- Repositories: ROS code repositories represented a source of great help throughout the entire research, allowing to easily find previously developed ROS packages.

The Computation Graph level is a peer-to-peer network that allows all ROS processes to communicate with each other [20]. The main concepts included in the Computation Graph level are:

- Master: the ROS Master dynamically creates a list of all the ROS nodes and manages all the messages exchanged between ROS nodes. The ROS Master does not have to be physically executed on the same machine on which all the other nodes/services are run. In order to make them communicate properly, ROS Master, nodes and services must be under the same network.
- Nodes: ROS nodes are individual processes, written either in `C++` or Python (relying respectively on the roscpp or rospy client libraries). ROS allows users to run multiple nodes at the same time that can perform several tasks and dynamically exchange data.
- Messages: the most convenient way for ROS nodes to properly communicate is by exchanging messages. Every time a node is created the user can include specific message libraries depending on the kind of message they want to publish. A message (that is a

C++ data class) can be a standard primitive type (e.g. int, float) or a more complex struct customized to meet specific requirements.

- Topics: the ROS Master routes all the ROS messages using a system based on publishers and subscribers. Let us say that two nodes, A and B, need to communicate with each other, then node A has to publish coherent data (e.g. same data type) into a topic and node B has to subscribe to the same topic. Publishers and subscribers are not aware of each other and there may be multiple publishers and multiple subscribers per topic.

The ROS File-system level includes several physical resources that are stored in memory, such as:

- Packages:ROS packages represent the smallest build item in ROS and can encapsulate ROS nodes, ROS libraries and configuration files.
- Package manifests: package manifests provide information about the package (e.g. name, version, dependencies)
- Messages: messages in ROS are implemented using C++ classes. They can also be customized which requires a new C++ class to be placed inside the ROS Package folder.

3.2.2 ROS Tools

ROS allows users to create complex systems made of several nodes and, for this reason, already includes several tools for 2D and 3D visualization, logging, live plotting and analysis of sensor data. In addition, it also provides several terminal commands for debugging a running system. The following is a closer look at several of the most used tools in this project:

- RQT: RQT is a Qt-based [1] framework. Among all its functions, it allows users to visualize data, subscribe and publish to topics and create GUI nodes in ROS.
- RVIZ: RVIZ is a 3D visualization package that allows users to properly view complex data types in a coherent way (e.g. sensor data, camera messages, depth information).
- RQT graph: RQT graph is used to display a dynamic flowchart of ROS nodes and topics that are currently running. It was extensively used for debugging purposes during the whole research.
- Command Line tools: The most used command line tools are:

  ```
  rosrun <package> <executable>
  ```

  Calling this tool you can run a ROS executable without having to provide its full path.

  ```
  rostopic echo \<topic>
  ```

  It allows the user to print on screen all the messages sent to a specific topic.

  ```
  roslaunch package_name file.launch
  ```

  Using this command you can provide additional parameters (inside the `.launch/XML` file) that have to be fed to the node.

---

[1] Qt is an application framework used for developing software that runs under Windows, Linux and OS

3.3 ROS messages

Every time a ROS node publishes or subscribes to a topic, it has to specify the type of message it expects to send/receive. The following table (Table I) shows the most common standard messages supported by ROS.

TABLE I: MOST COMMON ROS MESSAGE TYPES

Primitive Type	**Serialization**	**C++**
bool	unsigned 8-bit int	uint8_t
*int*8	signed 8-bit int	int8_t
*uint*8	unsigned 8-bit int	uint8_t
*int*16	signed 16-bit int	int16_t
*uint*16	unsigned 16-bit int	uint16_t
*int*32	signed 32-bit int	int32_t
*uint*32	unsigned 32-bit int	uint32_t
*int*64	signed 64-bit int	int64_t
*uint*64	unsigned 64-bit int	uint64_t
$float32$	32-bit IEEE float	float
string	ascii string	string

3.4 ROS messages and Arduino

During the research, it is was necessary to provide a low level control of the physical sensors and actuators. Among all the different choices the micro-controller Arduino Mega represented the most reliable and well-supported solution.

Arduino Mega is a popular board that implements the ATmega1280 microprocessor. It has 16 analog inputs and 54 digital IN/OUT pins. The communication to the computer is facilitated

using a USB cable. It can be easily programmed using a Java-based integrated development environment (IDE) which supports a slightly simplified version of `C++`.

The easiest and most reliable way to make ROS and Arduino communicate is by using the `rosserial_arduino` package.

The way this package works is by automatically setting up a Serial communication with Arduino and making it dynamically exchange data with `roscore` [21]. To set everything up, the fist step is to export all of the ROS libraries. This process can be easily accomplished by using the command:

```
rosrun rosserial_arduino make_libraries.py .
```

The ROS node `make_libraries.py` compiles all the packages under that path and creates related Arduino libraries (header files) that must be added to the project. In order to subscribe and publish to topics, each Arduino Sketch must have the library `ros.h` included. Additionally, it is also necessary to include any header files related to the message you want to send/receive.

3.4.1 A simple example

Let us suppose you want to turn ON/OFF a LED using a ROS message. These are the main steps you need to follow:

1. Create a ROS node that publishes data, in this case boolean, into a topic (e.g. "LED switch").
2. Run `roscore` from the terminal.
3. Export all the ROS libraries and add them to the Arduino IDE.

4. Add the following ROS libraries to your Arduino Sketch:

```
#include <ros.h>
#include <std_msgs/Bool.h>
```

The first `include` is the generic ROS library that allows the program to instantiate a nodehandle class (used to subscribe and publish data); the second `include` is needed to properly receive messages from the ROS node of type boolean.

5. Compile and run the ROS package. Compile and upload the Arduino sketch.

6. Finally you need to run the rosserial client application. It will automatically forward all ROS messages of the topic "LED switch" to your Arduino board that is currently listening on the Serial port.

```
rosrun rosserial_python serial_node.py _port:=/dev/ttyUSB0
```

3.5 ROS on multiple machines

ROS allows users to run multiple ROS nodes across several machines connected to the same network. In the configuration used for the research there is one ROS Master (on a desktop machine) and one Client (on a laptop). The main steps for this setup are the following:

- Make sure that the Master and the Client can ping each other and communicate using a specified port (bi-directional connectivity is required)
- Install ssh services (ssh server and ssh client) on both machine. They are used by ROS to make the nodes communicate to each other.

- Each client has to set the variable `ROS_MASTER_URI` with the IP of the Master in each terminal window, before executing any ROS commands
- Machine name resolution: each machine has to advertise itself by host name (or IP address) so that it can be addressable by other machines. This process has to be done in each terminal window, before executing any ROS commands

For example, in an instance where the IP address of the master is `192.168.1.107` and the IP address of the client is `192.168.1.109`, the master has to execute the following code:

```
export ROS_MASTER_URI=http://192.168.1.107:11311
export ROS_IP=192.168.1.107
export ROS_HOSTNAME=192.168.1.107
```

whereas the client has to execute:

```
export ROS_MASTER_URI=http://192.168.1.107:11311
export ROS_IP=192.168.1.109
export ROS_HOSTNAME=192.168.1.109
```

CHAPTER 4

THE PROTOTYPE

An important goal of this thesis is to evaluate the monitoring approach on a realistic system. For this reason, during the research project, the model of a robot that runs an obstacle avoidance algorithm was engineered. The main idea was to develop a flexible platform that can represent a solid base for a set of experiments based on monitoring theory. As a base structure, the model Rover A4WD1 v2 Robot, made by the company Lynxmotion[1] was employed.

Figure 8: The final prototype

[1]www.lynxmotion.com

In Figure 9 there is summary of the main blocks needed to properly control the car. An Arduino board is connected to all the sensors/actuators. It communicates in a serial way to a small laptop (client), mounted on top of the prototype. The laptop is then connected using Wi-Fi communication to a desktop computer (master) that executes all the ROS nodes (including `roscore`)

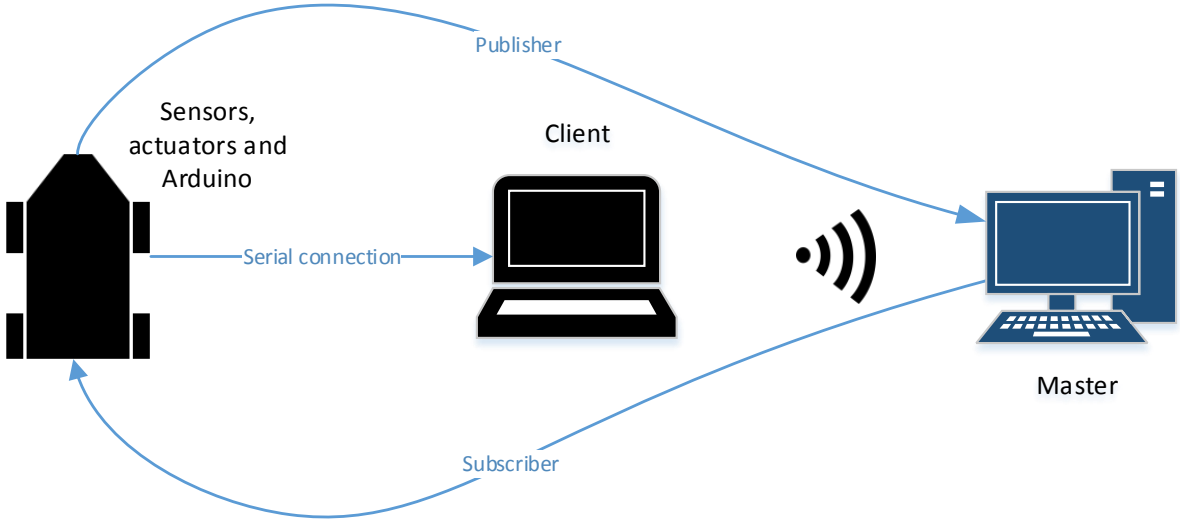


Figure 9: The main connection blocks

During the entire project, an Arduino Mega[1] board was used to interface with the sensors (3 sonar and 2 encoders) and a motor driver connected to the actuators (two couples of electric

[1]http://arduino.cc/en/Main/ArduinoBoardMega2560

motors). On top of the metallic chassis was designed a plastic structure, made with products from the company Evergreen Scale Model[1] to hold the sonar sensors, the batteries and the small laptop.

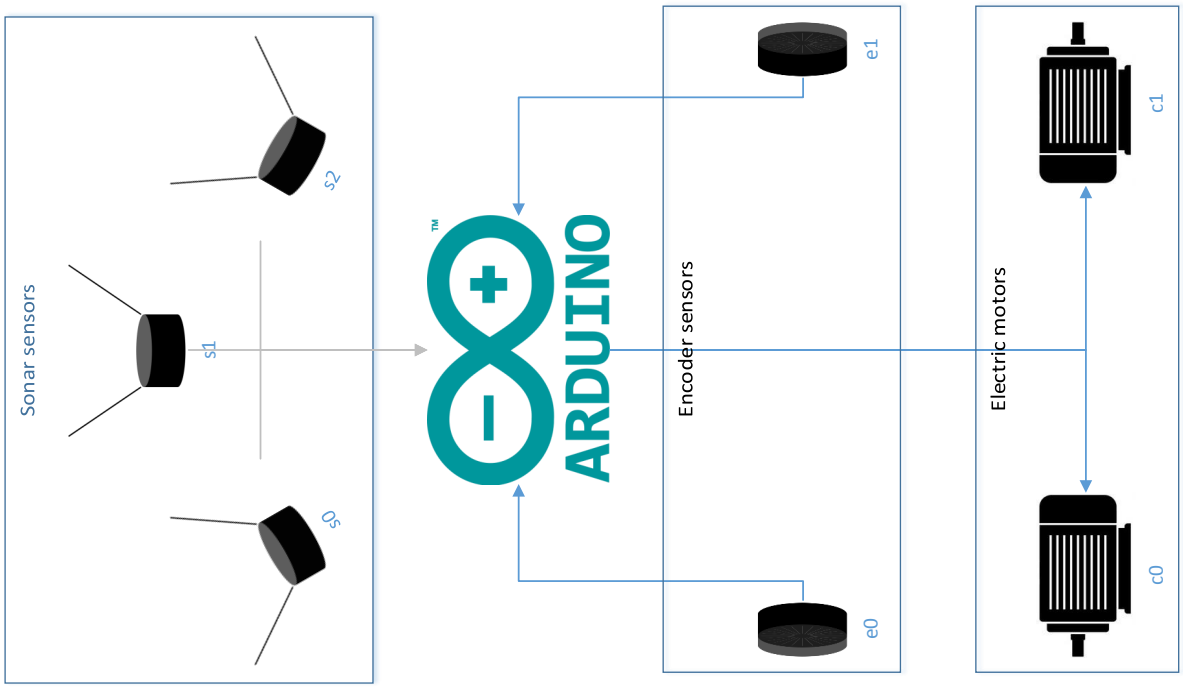


Figure 10: Physical layer using the developing platform Arduino

[1] www.evergreenscalemodels.com

4.1 The implementation

The implementation process involved the creation of a plastic chassis and the connection of all the electrical components that have been used throughout the research. Which includes:

- 4 Electric motors
- 1 Motor driver
- 2 Encoder sensors
- 3 Sonar sensors
- 1 RGB LED
- 2 Batteries
- 1 Switch
- 1 Arduino Mega Board

4.1.1 The Physical layer

All the internal connections, between the components mentioned before, have been soldered together on a "blank" prototyping board (called a *shield*) placed on top of the Arduino board (Figure 11). The use of a shield was needed to avoid unwanted interference and to reduce the risk of short-circuits among the multiple wires. The two batteries have been connected with the motor driver by employing additional connectors (in order to unplug them easily). In addition, a switch has been placed to disconnect the batteries from the motors, allowing them to charge inside the assembled prototype.

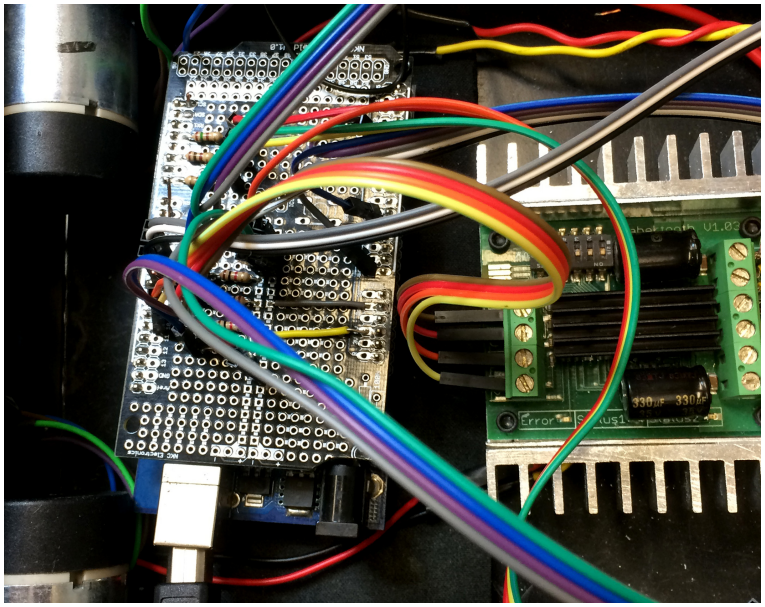


Figure 11: On the left side there is the custom Arduino shield, on the right side the motor driver is present

Since the micro-controller has a very low output current and the motor draws a higher current, a motor driver was employed to properly power the actuators.

4.1.2 Arduino pt.1 `setup()`

Arduino Mega Board is a small board with a micro-controller (ATmega1280) and complementary components. After connecting all the sensors/actuators it was possible to easily implement the algorithm, using the proprietary IDE (Integrated Development Environment) [22]. The main idea is to publish via ROS all data coming from the sensors (3 sonar and 2 encoders) and correctly subscribe to a ROS topic with the control that has to be applied to the wheels. The Arduino IDE is multi-platform and it has been designed to speed up the prototyping phase by allowing the user to directly code in `C++`, using a vast library of function that can

easily be customized to user preferences. In order to compile and upload a code in Arduino, the user has to define two functions:

- `void setup()`: this function is executed only once at the beginning of the program and it is used for the initial settings
- `void loop()`: this function is called repeatedly until the Arduino board is powered off

Let us take a closer look at the implementation of the function `void setup()` (Figure 12).

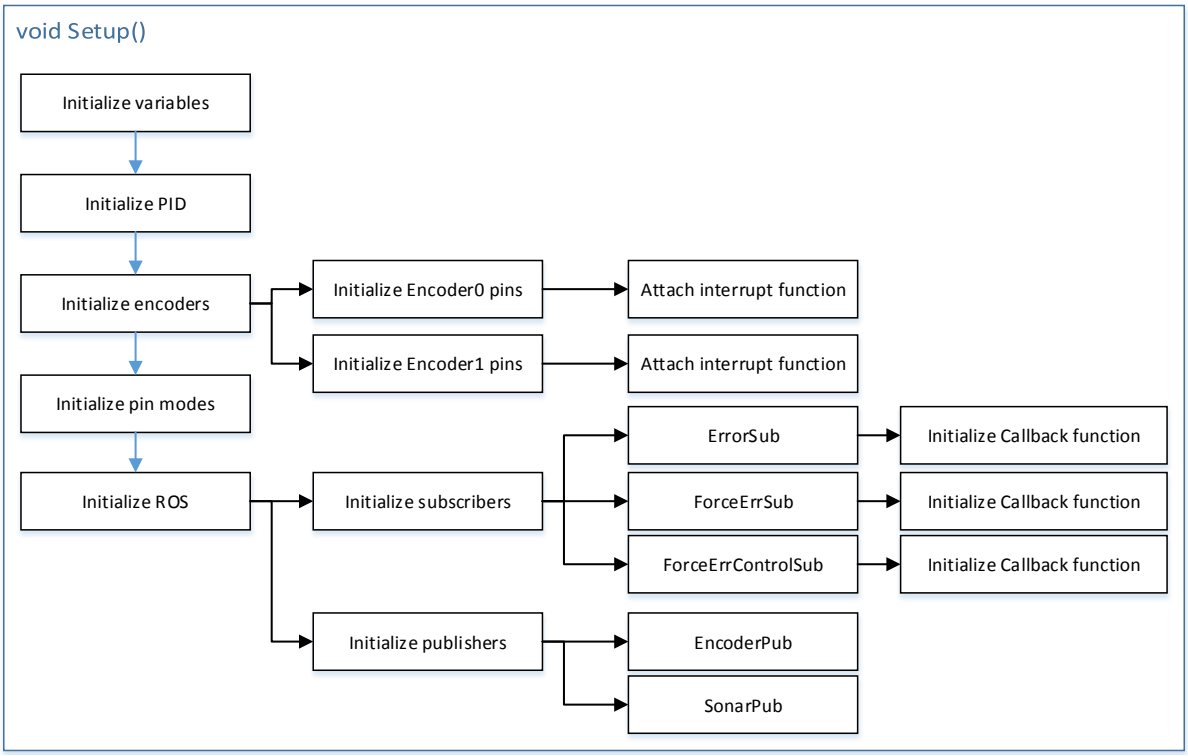


Figure 12: Flowchart of the algorithm implemented in the `setup()` function of Arduino

The first part focuses on initializing the variables needed for the correct execution of the code. These include the global variables that store data for the callback functions, for the interrupt functions and for the PID control (Proportional Integrative Derivative). Having a PID control was crucial to make sure that the wheel could spin at the desired control speed. In fact, without such precision, it would not have been possible to implement a monitor that could check the correctness of the system. During the research project, a standard Arduino PIDlibrary [1] was used. In the `setup()` function, it was needed to instantiate and initialize two `PID` objects that refer the two feedback controls of the wheels.

Given an error $e(t)$ then the control $u(t)$ is:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{d}{dt} e(t) \tag{4.1}$$

where k_p is the gain related to the proportional error ("present error"), k_i is the gain related to the integral error ("past error") and k_d is the gain related to the differential error ("future error").

The encoders' pins in the Arduino board have to then be set as input and, for each encoder, an interrupt function must be attached. Each encoder has four wires: two wires for power and two for signal. In order to detect the actual speed of the wheel, it was sufficient to evaluate the time that passes among the two consecutive calls of the related interrupt function. The second sensor, present inside the encoder, is needed to compute the direction of the wheel. Note that

---

[1] http://playground.arduino.cc/Code/PIDLibrary

even if there are 4 electric motors and 4 related encoders, due to technical limitations of the Arduino board only two encoders have been used: one for the left and one for the right set of wheels.

The main steps, needed to perform a distance measure, are as following:

- Send a voltage impulse to turn on the sonar sensor. The sonar then sends an ultrasonic impulse (42 kHz) and measures the time until it receives a response. The resolution is 1 cm and the detection range is in [20 cm, 275 cm]. The maximum usable frequency at which you can perform a reading in all conditions is 10 Hz. However, since a total number of 3 sonar have been employed, the frequency reading rate has to be lower to avoid crosstalk.
- Read the analog output pin of the sonar whose voltage is linearly dependent with the distance of the first object detected.

All the pins' modes (IN/OUT) of the pins connected to the three sonar have to be initialized to correctly perform the readings.

Last, ROS objects (e.g. node-handles, publishers and subscribers) are employed to communicate with the ROS system.

4.1.3 ROS Messages

In order to properly communicate with the prototype, there are a number of messages (Figure 13) that are exchanged between the server, the client and Arduino. The following is a closer look at the main ROS topics and related messages:

`Control.msg` is a C struct that is filled with the readings of the three sonars and is published into the topic \`Sonar_data`. The obstacle avoidance algorithm subscribes to this topic and

TABLE II: MOST USED ROS TOPICS AND MESSAGES

Topic name	**Custom Message**	**Message type**	**Message class**
\Control_data	✓	Control.msg	UInt8 c0 UInt8 c1
\Encoder_data	✓	Encoder.msg	UInt8 e0 UInt8 e1
\Sonar_data	✓	Sonar.msg	UInt8 s0 UInt8 s1 UInt8 s2
\Error_data		std_msgs	Bool data
\Force_err_data		std_msgs	UInt8 data
\Force_err_control_data	✓	Control.msg	UInt8 c0 UInt8 c1

publishes a control in \Control_data specifying an individual control for the two sets of wheels. Since the type UInt8 $\in [0, 255]$, if *control* $\in [0, 127]$ the wheels are rotating backward and if *control* $\in [129, 255]$ the wheels are rotating forward. Arduino also publishes the velocity of the individual sets of wheels in the topic \Encoder_data, subscribed to by the monitor performing *internal monitoring* of the system, which publishes a boolean flag in the topic \Error_data. If Arduino receives Error_data = true, it stops all the motors and turns on a red light using the LED on the side.

The last two topics are used for debugging purposes of the monitor node. Depending on the numerical value (0, 1, 2 and 3), sent in the topic \Force_err_data, Arduino overrides the received control data of none, left, right or both sets of wheels, applying instead the value published in the topic \Force_err_control_data. In this way it is possible to simulate a failure of one or both sets of wheels without the need to physically block the motors.

Note that all the custom messages are implemented using `C++` classes, and for these messages to correctly be compiled by Arduino, relative libraries had to be created. This step was accomplished by first developing a ROS node that was using those classes (e.g. obstacle avoidance node), compile it and then export the classes to the Arduino library folder.

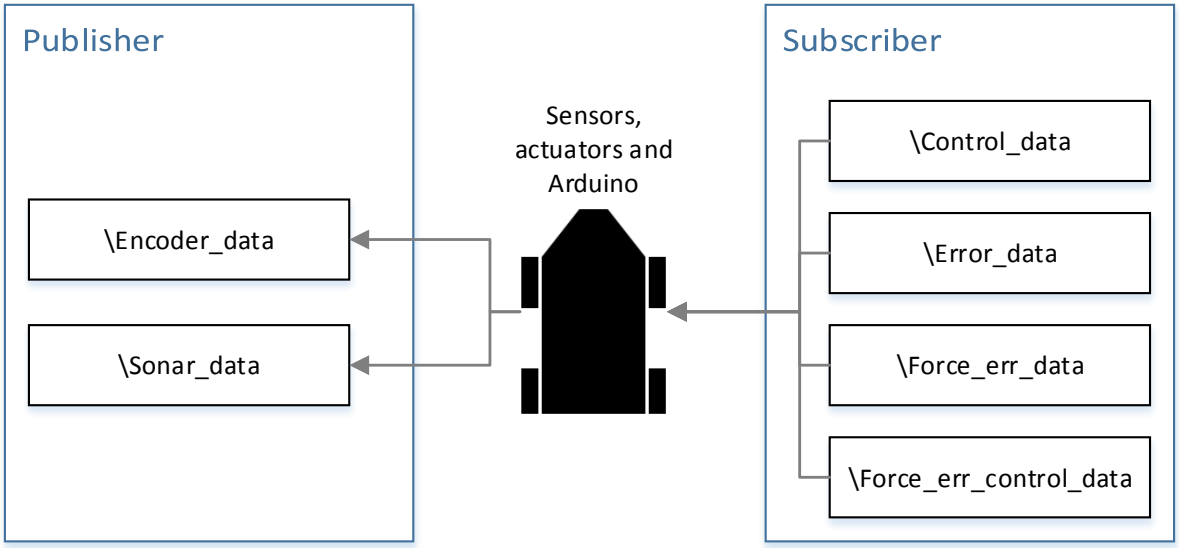


Figure 13: Topics published and subscribed from Arduino

4.1.4 Arduino pt.2 `loop()`

Before focusing on Figure 14, which describes the main algorithm inside the `loop()` function, let us take a closer look at the background functions (e.g. ROS objects) and interrupt functions. Note that by using ROS libraries and attaching the right callback functions, the program

automatically updates the value of the following variables: the control estimated by the obstacle avoidance, the error message, the force error message (simulating a failure) and the related force error control. Furthermore, two additional interrupts had to be attached to the pins connected to the encoders to assure that the velocities of both sets of wheels could always be measured with good precision. Let us now take a closer look at Figure 14.

Before executing any other instruction, Arduino individually activates the sonar sensors and performs distance measurements. In this way, it is assured that no crosstalk affects the measurements. The acquired values are checked and clipped to be inside the interval $[20, 255]$: the lower bound represents the minimum distance that can be read by a sonar sensor and the upper bound is the maximum value that can be represented using the message type `UInt8`.

Using the readings of the encoders, dynamically updated by interrupt routines, the relative velocity of the car is then estimated. Those calculations could theoretically be done inside the interrupt routines, but since those routines have to be executed as fast as possible, it has been preferred to put those additional instructions inside the `loop()` function. The reason behind this is because while the CPU of the micro-controller executes an ISR (Interrupt Service Request) it automatically disables all interrupts for stability and security purposes (e.g. stack overflow). If an interrupt routine takes too much time to be executed, then the probability that some interrupts are missed (directly affecting the velocity evaluation process) rises. Among other technical problems, this was one of the main reasons why only two encoders (out of four available) have been connected to Arduino. Once sonar and encoder sensor readings have been correctly read and saved, they are then ready to be published using ROS functions.

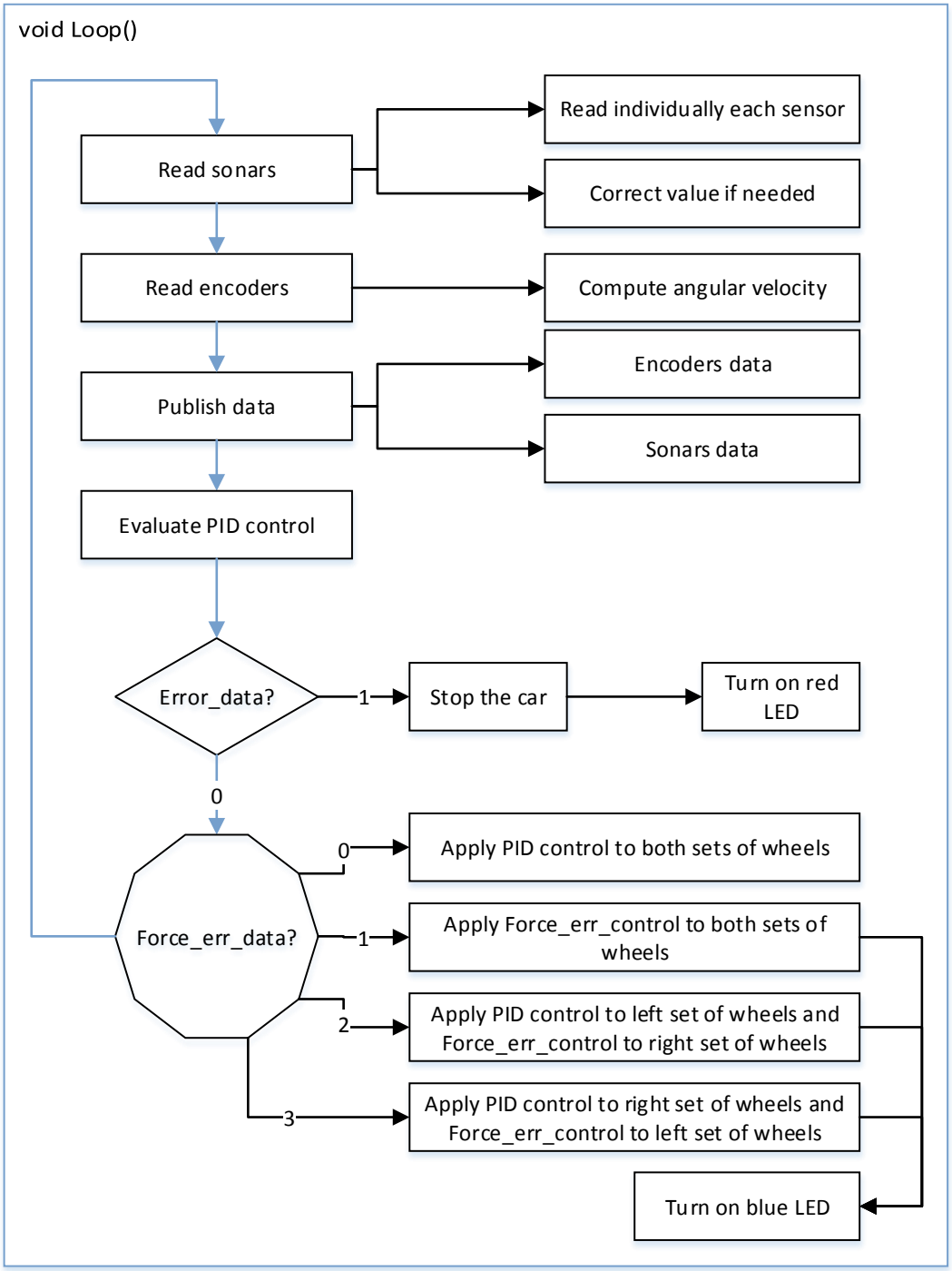


Figure 14: Flowchart of the algorithm implemented in the `loop()` function of Arduino

In the background, an interrupt service routine updates a global C struct that contains information related to the control that has to be applied to the wheels. This data is then fed to the PID object that evaluates the correct control according to the past encoder sensor readings. If an error message is sent (e.g. by the monitor running on the ROS master), then a speed equal to zero is written by Arduino in the pins connected to the controls of the motor driver, otherwise a check on \`Force_err_data` is performed to detect a control override command. According to the value of the received message, the correct control is then applied to the wheels.

The RGB LED has been placed on the right lower side of the prototype. In summary, the Arduino code has been designed so that:

- if the LED is green, no error message has been received and everything is performing correctly (no failures of sensors). The car applies the received control
- if the LED is red, an error message has been received: a failure has been detected by the monitor. No matter what the control data is received by Arduino, the car is stopped
- if the LED is blue, an override command has been sent by the master. One or both sets of wheels are directly applying an arbitrary control (not estimated by the obstacle avoidance algorithm). It is likely that the monitor will detect the related Error mode in a short time and send an error message, switching the LED to a purple light (blue light + red light)
- if the LED is purple, both an error message and a force error message have been received. No matter what control data is received by Arduino, the car is stopped

4.2 Introduction to the control system and additional ROS nodes

In order to correctly run the system, the user has first to first select a desired speed. This is accomplished by executing the node *set_desired_speed* that publishes a message in the topic \`Desired_speed_data`, read both by the monitor node and the obstacle avoidance node. Afterwards, the control system can be launched. By reading the messages published by the client (connected to Arduino), it evaluates and sends the proper control. At the same time, a monitor node checks the correct operation of the prototype by estimating the inner state of the control system (obstacle avoidance). The main goal is to develop a simple obstacle avoidance algorithm that can be easily monitored using particle filters.

4.2.1 Obstacle avoidance

The first implementation of the obstacle avoidance is designed to be extremely easy to monitor. As mentioned earlier, one of the main goals of this thesis is to develop a coherent test methodology that can lead to more accurate generation of monitors. There are a number of obstacle avoidance algorithms that could have been employed (e.g. "Smooth path planning" [23]), but in order to successfully investigate some fundamental characteristics of the monitoring theory (e.g. robustness), it has been decided to adopt a new simplified version of a general obstacle avoidance. As illustrated in Figure 16, only two system modes are present:

- $D = 1$: "Straight". The car goes straight: the controls of both sets of wheels are set to the desired speed.
- $D = 2$: "Spin". The car spins on itself, that is the control values of both sets of wheels are fixed (half of desired speed), but the direction of each set of wheel is opposite

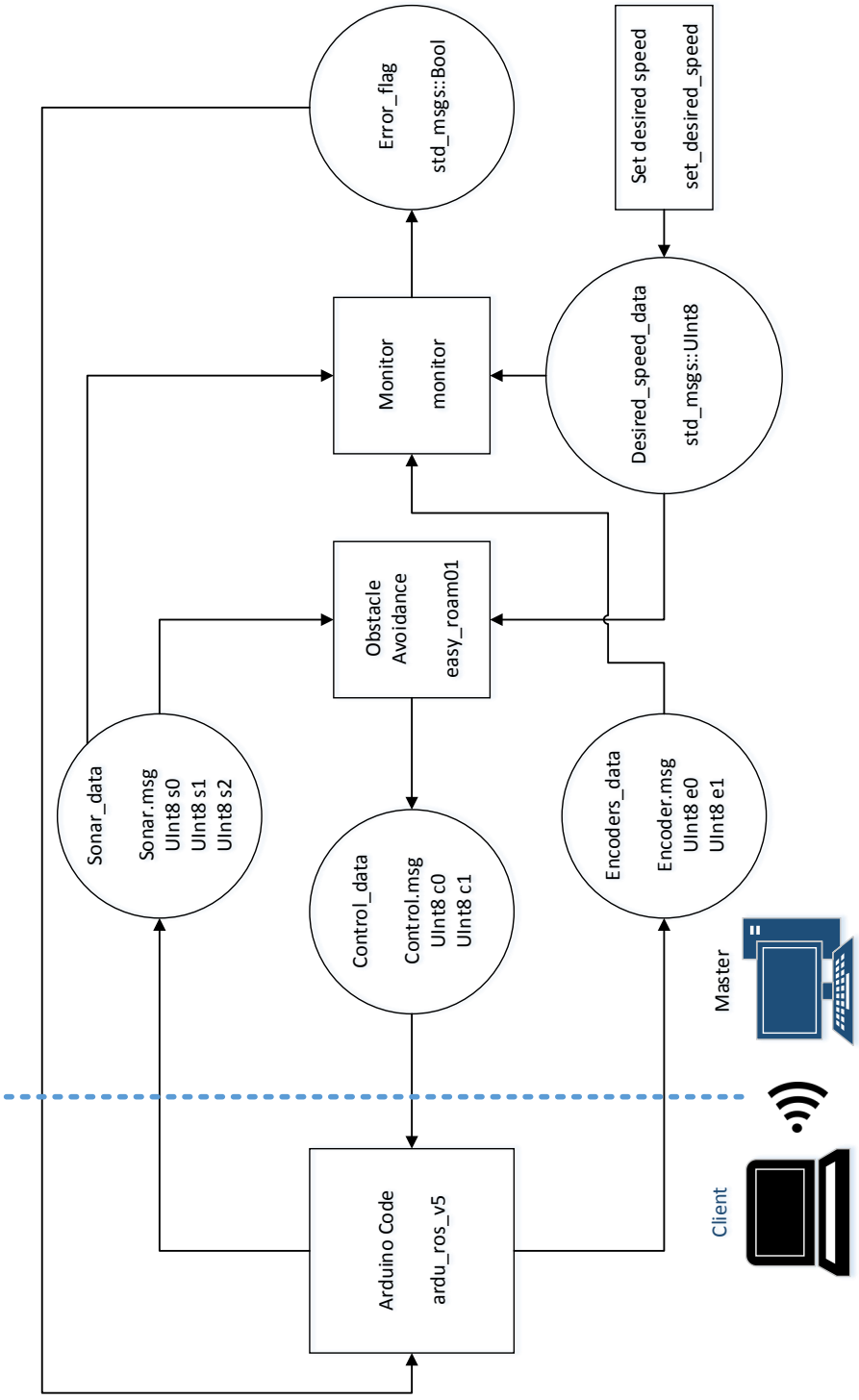


Figure 15: Main ROS nodes, topics and messages

The threshold $k0$ marks the transition between $D = 1$ and $D = 2$. The minimum sensor reading among the three sonar readings is $s\_min = min(\vec{s})$. Since the delay of the overall system is extremely big, no hysteresis has been added to the algorithm.

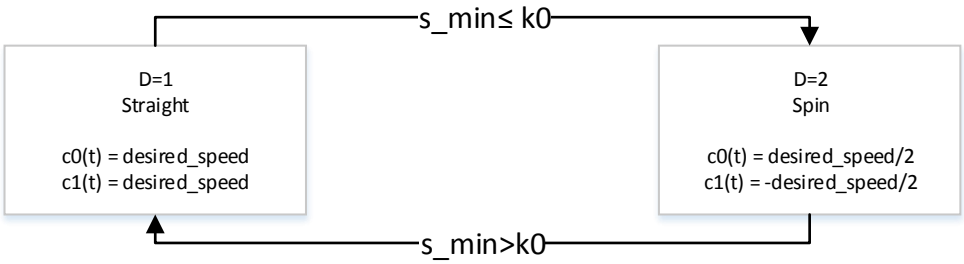


Figure 16: Obstacle avoidance algorithm (basic implementation)

The algorithm is based on the following state equation:

$$\vec{c}_{k+1} = f(\vec{s_k}, ds_k) \tag{4.2}$$

The applied control is a function of the distance seen by the sonar sensors in all three directions and the user selected desired speed ds.

To properly implement the algorithm a new C class `SubscribeAndPublish()` was created in ROS. Once instantiated, it automatically publishes new controls in the shortest time possible only when it receives new readings from Arduino. This solution represented a better alternative

than using the common structure, based on a `while (ros::ok())` (suggested in many online tutorials) that would have created redundant messages and a lower response time of the node.

4.2.2 Other nodes

A number of additional ROS nodes has been used throughout the whole research, let us take a closer look:

- `rosserial_arduino` is a node that allows serial communication with Arduino. In this way, it is possible to properly interface the micro-controller with the laptop mounted on top of the prototype. At present the client does not execute any other node, except this one, acting like a proxy between the Arduino board and the ROS master. Theoretically, it would have been possible to use an Arduino Wi-Fi shield to send data directly from the board to the ROS master, but the high number of ISR routines executed at the same time by the micro-controller would have probably caused high latency and possible missing packets
- `set_desired_speed` allows the user to correctly publish the *desired speed* topic
- `log_data` saves a log of all information subscribed and published to Arduino for $t = \{0, 1, \ldots, 99, 100\}$: $\vec{s_t}$ (sonar sensors readings), $\vec{e_t}$ (encoder sensors readings), $\vec{c_t}$ (control vector) [24] and ds_t (desired speed).
- `input_data` allows the user to manually input sonar, encoder and control data. It was mainly used to correctly debug the monitor and simulate previously recorded sessions (using the node `log_data`)

4.2.3 Scripting

To properly speed up the process of setting up the ROS master for each work session, a Linux script has been developed. Note that to open a new terminal window the command `gnome-terminal` has been employed. The main steps executed are as follows:

- "Source" two `.bash` files: the first one is related to the ROS system, whereas the second one is used just for the current workspace
- Open a new terminal window, execute the ROS node "`set_desired_speed`" and wait 4 seconds so that the user can input the desired speed
- Open a new terminal window and execute the obstacle avoidance node
- Open a new terminal window and execute the monitor node
- Open 5 new terminal windows and execute the `rostopic` command to listen to the following ROS topics: `\Desired_speed_data`, `\Encoder_data`, `\Sonar_data`, `\Control_data`, `\Error_data`

  Before executing the above mentioned script, it is needed to "`ssh`" the master and launch `roscore`.

CHAPTER 5

THE MONITOR

In order to properly monitor the correctness of the system, a monitor node has been developed in ROS. Initially, due to the difficulty of debugging ROS code, a fist implementation has been carried out in Matlab. Once the correct behaviour of the code has been verified, the program is ported in `C++`. At present, the algorithm monitors the state of the system by making a prediction of the next set of observations, using an approximated model of the system (Hidden Markov Model). This has been accomplished employing a particle filter that runs in real-time [25] [26].

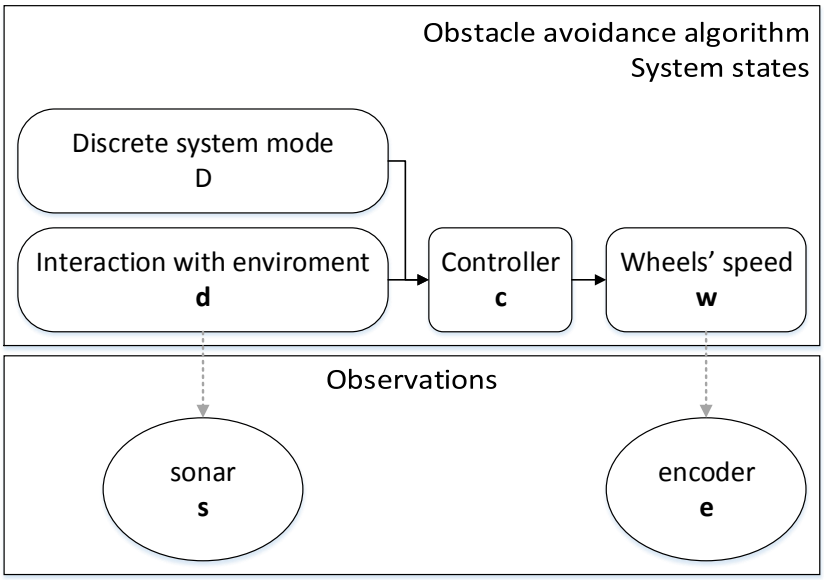


Figure 17: System states and observations

Let us take a closer look at Figure 17. The set of observable outputs is $y = \{\vec{s}, \vec{e}\}$, where $\vec{s}$ is the vector containing the three sonar sensors readings and $\vec{e}$ is the vector containing the two encoder sensors readings. D (system mode), $\vec{d}$ (distance vector), $\vec{c}$ (control vector), $\vec{w}$ (wheel's velocity vector) are the system states $x = \{D, \vec{d}, \vec{c}, \vec{w}\}$.

5.1 System State Model

The system state model inside the monitor is based on the obstacle avoidance algorithm (Figure 16), but it also takes into account a possible failure of the electric motors (stochastic). In addition, note that no complete deterministic model of the system is available: given an arbitrary observation (5-tuple: 3 sonar and 2 encoder sensors) it is impossible to make a unique prediction of the system (e.g. in the spinning mode no sufficiently accurate prediction of sonar sensor readings can be done). For this reason, a stochastic model of the system has also been employed. It has also been assumed that not all arbitrary failures of the system can be detected (e.g. since the prediction of sonar sensors readings can be performed with good accuracy only in the system mode "straight", a possible failure of sonar sensors may not be identified in other modes). Basing on this assumption, for optimization purposes, there is no need inside the particle filter to spread the particles across the whole map, that is the multiplication of all the possible readings (e.g $[0-255] \times [0-255] \times [0-255] \times [0-255] \times [0-255]$). In fact, there is no predetermined model that describes all the possible evolutions given an arbitrary reading with sufficient accuracy. In this framework, only 3 failures have been modelled (both wheels blocked, left wheel blocked and right wheel blocked).

5.1.1 Prediction of $\vec{d}_{k+1}$ and D_{k+1} - Deterministic transitions

Given the system states at time k: $x_k = \{D_k, \vec{d}_k, \vec{c}_k, \vec{w}_k\}$, let us take a closer look at the deterministic prediction of the system mode D_{k+1} and distance vector $\vec{d}_{k+1}$ (Figure 18) [27]:

- $D_k = 1$: "Straight". The predicted distance vector $\vec{d}_{k+1}$ is evaluated by the function $\vec{f}(\vec{w}_k, \vec{d}_k)$. There are two cases that have to be taken into account:
  - Case 1: if the current distances are *all* sufficiently far from the upper bound of the distance interval (e.g. $min(\vec{d}_k) << b,\ d \in [a, b]$), then the function $\vec{f}(\vec{w}_k, \vec{d}_k)$ produces an output that is inversely proportional to the wheels' velocities:

$$\vec{d}_{k+1} = \vec{d}_k - [\lambda_d * avg(\vec{w}_k)] + n_s \tag{5.1}$$

    where λ_d is a gain coefficient and n_s is the average noise of the sonar sensors. This is the only situation where the prediction of the distance vector $\vec{d}_{k+1}$ is generated in a consistent way.
  - Case 2: if the current distances are sufficiently close to the upper bound of the distance interval (e.g. $min(\vec{d}_k) \simeq b,\ d \in [a, b]$), than
    - 50% of the time:

$$\vec{d}_{k+1} = \vec{d}_k + n_s \tag{5.2}$$

      In fact, the algorithm has to take into account the possibility that there are no obstacles in front of the car so the distances at time k will not decrease.

* 50% of the time:

$$\vec{d}_{k+1} = \vec{d}_k - [\lambda_d * avg(\vec{w}_k)] + n_s \tag{5.3}$$

Based on the new predicted distances, a new state D_{k+1} is estimated, that is

$$if\ min(\vec{d}_{k+1}) \geq k0 \qquad \Rightarrow D_{k+1} = 1 \tag{5.4}$$

$$if\ min(\vec{d}_{k+1}) < k0 \qquad \Rightarrow D_{k+1} = 2 \tag{5.5}$$

- $D_k = 2$: "Spin". No precise prediction of the future distances $\vec{d}_{k+1}$ can be done.
  - 50% of the time, the function $\vec{g}(\vec{c_k}, \vec{d_k})$ generates an arbitrary distance vector $\vec{d}_{k+1}$ such that $min(\vec{d}_{k+1}) \geq k0$ ($\Rightarrow D_{k+1} = 1$).
  - 50% of the time, the function $\vec{g}(\vec{c_k}, \vec{d_k})$ generates an arbitrary distance vector $\vec{d}_{k+1}$ such that $min(\vec{d}_{k+1}) < k0$ ($\Rightarrow D_{k+1} = 2$). Note that in this mode as well as in the system modes $D_k = 3$, $D_k = 4$, $D_k = 5$, the distance vector $\vec{d}_{k+1}$ does not give additional information about the system; for this reason it is not taken into account during the re-sampling process.
- $D_k = 3$: "Failure state". No prediction of $\vec{d}_{k+1}$ is performed. $D_{k+1} = 3$
- $D_k = 4$: "Failure state". No prediction of $\vec{d}_{k+1}$ is performed. $D_{k+1} = 4$
- $D_k = 5$: "Failure state". No prediction of $\vec{d}_{k+1}$ is performed. $D_{k+1} = 5$

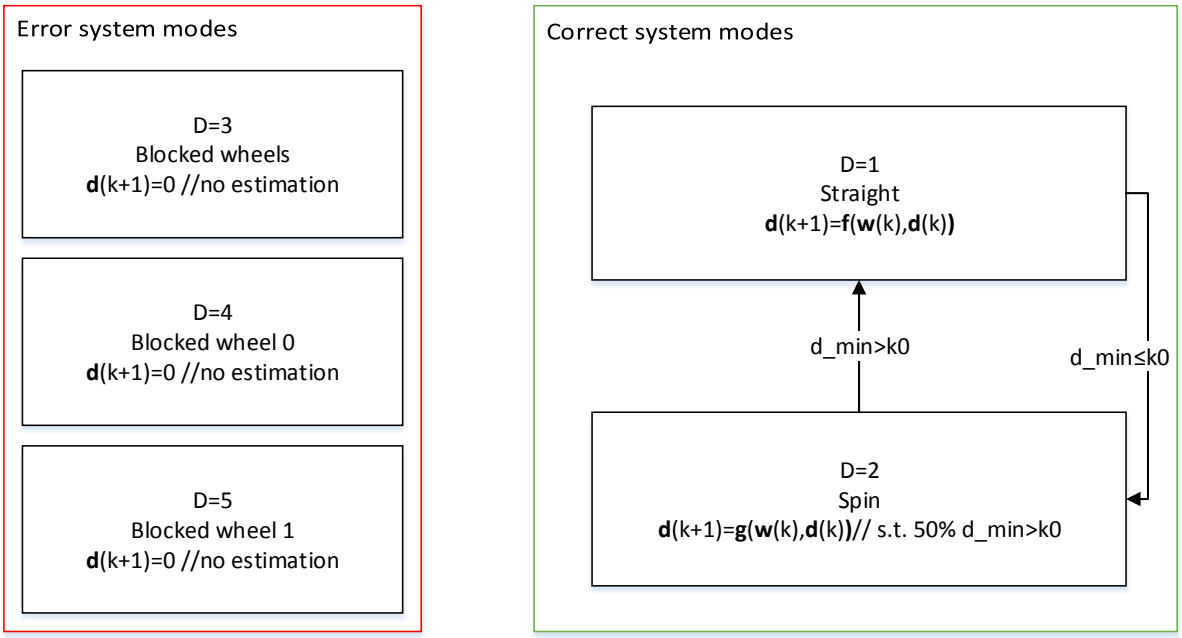


Figure 18: Deterministic transitions inside the system state model of the monitor node

5.1.2 Prediction of $\vec{d}_{k+1}$ and D_{k+1} - Stochastic transitions

The algorithm also has to take into account the stochastic transitions (Figure 19) between the system modes. Note that no transition state has been modelled here. Since the response of the motors is not ideal, the system will probably pass through an Error mode during the transitions $D_k = 1 \rightarrow D_{k+1} = 2$ and $D_k = 2 \rightarrow D_{k+1} = 1$. Since all the particles that go to an Error mode (e.g. $D = 3$, $D = 4$, $D = 5$) do not change mode (i.e. $D_{k+1} = D_k$) it is necessary to "inject" new particles in the system to take into account discrepancies of the model. Let us take a look at the main stochastic transitions:

- $p13$, $p14$, $p15$: the sum of those probabilities is α and represents the possibility of going to an Error mode from mode $D_k = 1$
- $p23$, $p24$, $p25$: the sum of those probabilities is α and represents the possibility of going to an Error mode from mode $D_k = 2$
- $p31$, $p32$, $p34$, $p35$: probability of injected particles from $D_k = 3$
- $p41$, $p42$, $p43$, $p45$: probability of injected particles from $D_k = 4$
- $p51$, $p52$, $p53$, $p54$: probability of injected particles from $D_k = 5$
- All the other probabilities (not shown in Figure 19) are 1 and the related transitions depend only on deterministic conditions.

5.1.3 Evaluation of controls $\vec{c}_{k+1}$ and wheels' velocities $\vec{w}_{k+1}$

According to the estimated system mode D_{k+1} and based on Equation 4.2, a new control is generated, as shown in Figure 20.

Note that in the Correct system modes, the wheels' velocity is $\vec{w}_{k+1} = \vec{c}_{k+1} + n_e$, where n_e is noise related to the encoders' sensors. On the contrary, this is not always valid for the Error system modes, where one or both wheels' velocities may be unrelated to the controls and be equal to 0.

Let us now take a look at the property automaton of the monitor, shown in Figure 21.

5.2 Property Automaton

The Property Automaton [28] is made of three states:

- $Q = 1$: "OK". The system is behaving correctly, no failure has been detected.

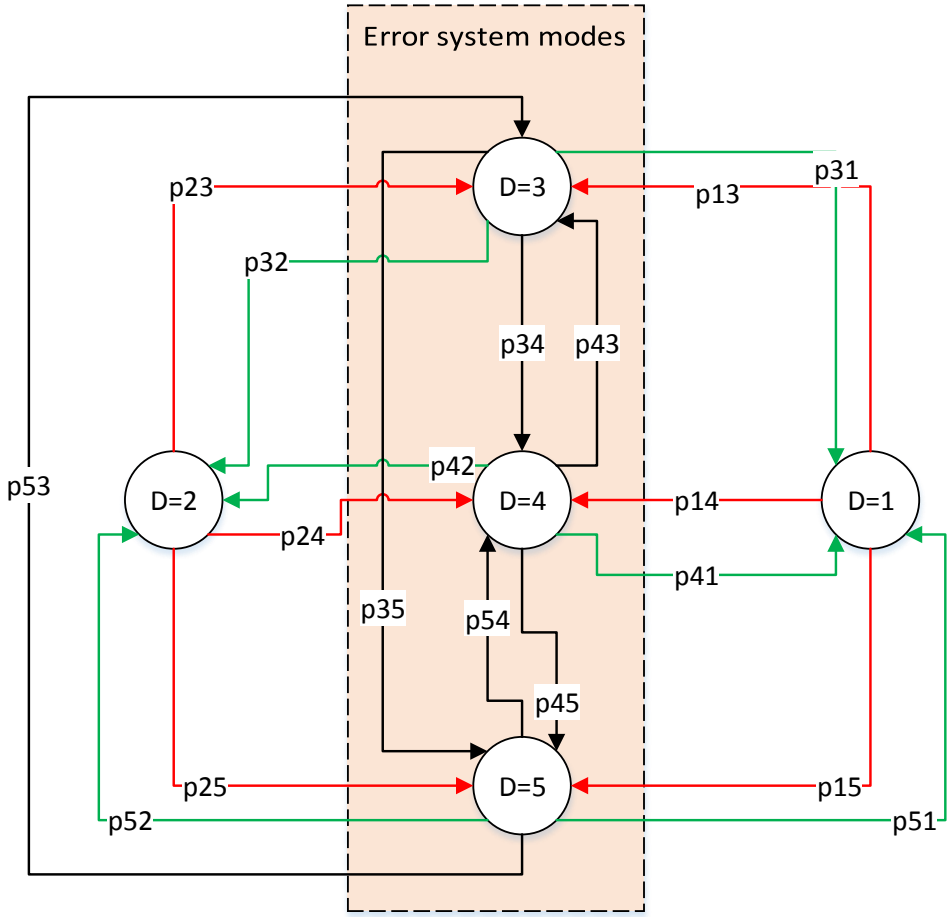


Figure 19: Stochastic transitions inside the system state model of the monitor

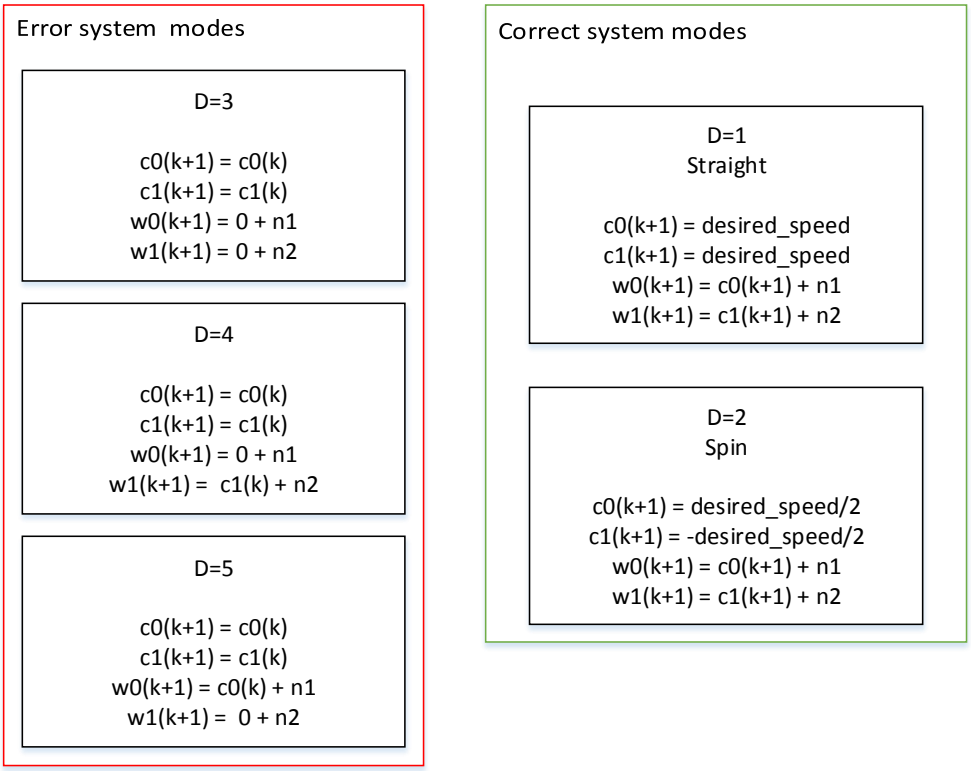


Figure 20: Outputs of the system state model employed inside the monitor

- $Q = 2$: "Counter". An error has been detected and the system went to an intermediate failure mode. If the value of the counter stored in this state reaches a given value, then the Property Automaton goes to $Q = 3$
- $Q = 3$: "Error". A failure has been detected.

The failure condition that the system is now monitoring is:

$$D = 3 \quad or \quad D = 4 \quad or \quad D = 5 \tag{5.6}$$

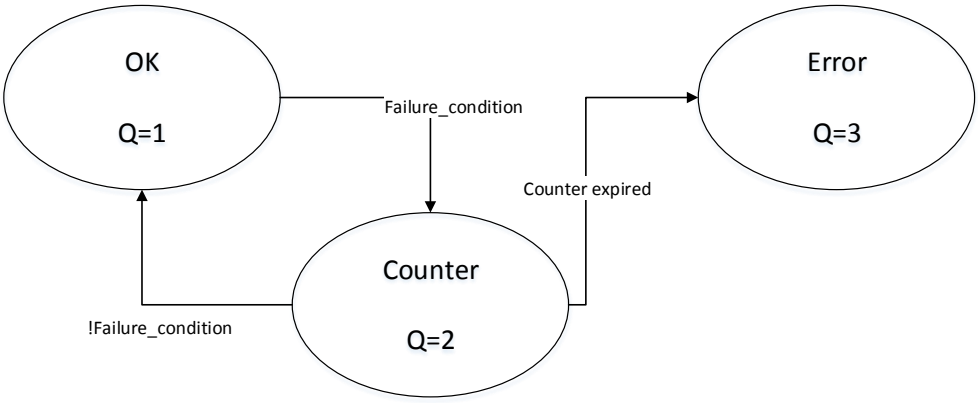


Figure 21: Property automaton (basic implementation)

5.3 Implementation

Two new `C++` classes have been defined to properly store data:

- `ParticleClass`: each particle is of class `ParticleClass`. This class contains all the information needed to reconstruct the state of the system at a specific time instant $t = k$: D_k (system mode), $\vec{d_k}$ (distance vector), $\vec{w}_k$ (wheel's velocity vector) [1], Q_k (property state) and cnt_k (counter value of the property automaton) (Figure 21).
- `InputData`: each sensor reading is saved in an `InputData` object. It allows the storing of observation data received at a generic time instant k (sonar and encoder sensors readings): $y_k = \{\vec{s}_k, \vec{e}_k\}$.

5.3.1 Main Algorithm

As previously explained, the main parts of the particle filter are sampling, prediction,weight evaluation and re-sampling. Each one of those steps plays a key role in the correct execution of the algorithm. Let us take a look at the function `main()` of the monitor ROS node (Figure 22). First, there is an initialization process that includes instantiation of ROS objects (publisher and subscribers), creation of particles (stored in a vector `ParticleClass`) and allocation for all the needed structures. In addition, in this step all the ROS callback functions are linked to the related topics, as shown in Figure 15. Afterwards, an array of class `ParticleClass`, containing all the particles, is allocated. Then, by means of a function `CreateSparseParticles` an initial particle distribution at time $t = t_0$ is created. All allocated particles are spread uniformly

---

[1] note that the vector $\vec{c}_k$ (controls) can be uniquely determined given D_k and $\vec{d_k}$

across the system modes. Coherent values of system states at time $t = t_0$ for each particle i are generated:

$$x_{i,t_0} = \{D_{i,t_0}, \vec{d}_{i,t_0}, \vec{c}_{i,t_0}, \vec{w}_{i,t_0}\} \tag{5.7}$$

Note that all system modes have well-defined outputs for $\vec{w}$ ($\vec{d}$ is not predicted). In all system modes, but $D = 1$, given a low value of sensors' noise, a sufficiently low number of particles can satisfactorily cover the whole domain of interest (the "map" of the particle filter). On the other hand, in system mode $D = 1$ the particle filter needs to also estimate continuous values of the distance vector $\vec{d}$. This is the only state where it is needed to spread the particles to cover the sonar sensor domain in order to approximate the sonar sensors readings with sufficient accuracy. Given N particles, the new generated array, at time $t = t_0$, is called `x_prior`:

$$\texttt{x\_prior} = \texttt{x\_prior}_t = \{x_{0,t}, x_{1,t}, \ldots, x_{i-1,t}, x_{i,t}, x_{i+1,t}, \ldots, x_{N,t}\} \qquad , t = t_0 \tag{5.8}$$

The second step involves the acquisition of data from sensors (i.e. observations [29]). This has been accomplished using flags inside the callback functions that guarantee that the information received are always up-to-date. Only if the monitor has new observations will it then proceed in executing the rest of the code. The next step, described in Figure 23, predicts the next state of all the particles from the previous distribution `x_prior`. The results are then stored in a new vector of particles: $\texttt{x\_posterior} = \texttt{x\_posterior}_t = \texttt{x\_prior}_{t+1}$. If the probability of being in the error state (i.e. the normalized number of particles is $Q = 3$) is greater than the *monitor threshold*, then the monitor publishes the error message and then exits.

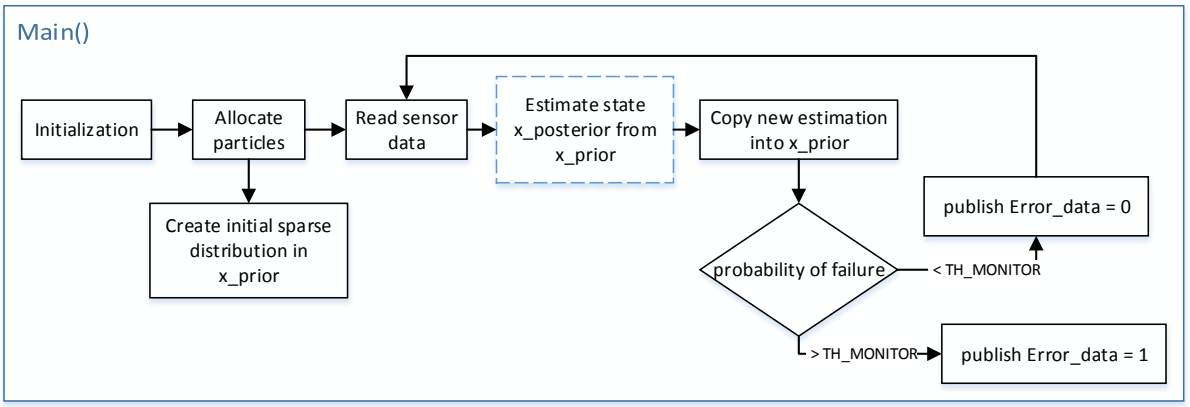


Figure 22: Basic algorithm implemented inside the function `main()` of the monitor ROS node

5.3.2 Estimation of `x_posterior` from `x_prior`

Let us now take a closer look at Figure 23. After initializing all the needed structures, each particle is propagated in time using the function `propagate_state()` (Figure 24). Depending on the particle predictions of system states, a weighting function evaluates how close the predictions are with the current observation (y_{k+1}). Note that since the prediction of sonar sensor readings only gives additional information in the system mode $D = 1$, weight coefficients change depending on the system mode D_{k+1} of the particle. Afterwards, all the weights are renormalized and the particles are re-sampled. In this last process, each particle is drawn with a probability directly proportional to the individual weight of the particle. In a probabilistic framework, a particle with bigger weight is more likely to be re-sampled than a particle with lower weight. At last, a complete distribution of all particles is generated and returned

(`x_prior`). At the same time, a matrix that shows the distribution of particles is printed on screen.

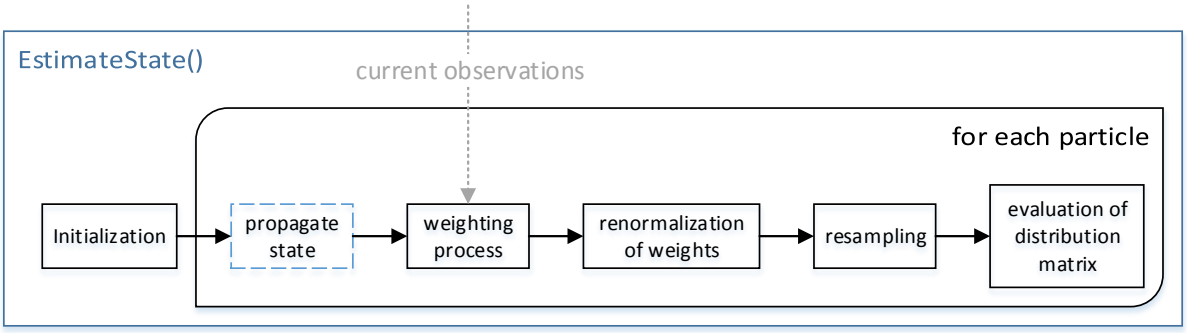


Figure 23: Basic algorithm implemented inside the function `estimate_state()`

5.3.3 Prediction of system states

Figure 24 represents the flowchart of the main steps inside the function `propagate_state()`. First, a prediction of the distance vector $\vec{d}_{k+1}$ and of the system state D_{k+1} is performed. This step is accomplished by taking into account the previous system state x_k, the noise that affects the sensors and the gain coefficient λ_d (coefficient that describes how the speed of the car $\vec{w}$ affects the distance vector $\vec{d}$). Afterwards, an evaluation of the new control vector $\vec{c}_{k+1}$ and then wheels' velocity vector $\vec{w}_{k+1}$ is performed. At last, after determining if the failure condition is verified at time $t = t + 1$, then the property state Q_{k+1} is evaluated and if necessary, the counter cnt_{k+1} (Figure 21) is incremented accordingly.

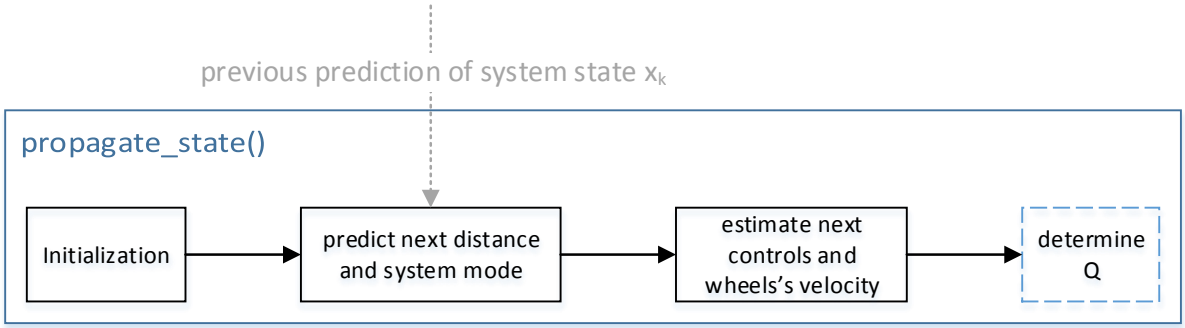


Figure 24: Basic algorithm implemented inside the function `propagate_state()`

CHAPTER 6

FURTHER DEVELOPMENTS

6.1 Improved obstacle avoidance algorithm

An improved obstacle avoidance algorithm has been developed in the last part of the research(Figure 25). There is a big literature related to the the topic of robust control systems [30], but it was decided to implement a better obstacle avoidance that could be still sufficiently easy to model.

For each sonar reading there is a threshold that marks if the obstacle is *far* or *close*. Since there are 3 sonar sensors, there is a total number of 8 cases that have to be taken into account. For each one of those system modes, a fixed wheels' control has been previously computed. Due to time constraints, no monitor that integrated this system model has been developed. This complex obstacle avoidance has been more deeply analysed by the graduate student Eric Serra.

6.2 New Monitor: more accurate Error modes

Let us consider a new model for the system described in Section 4.2.1 that takes into account a more general number of failures of the system. Instead of just detecting a blocked set of wheels, the algorithm models a malfunctioning of the wheels: if a set of wheels' readings is constant in time and it is sufficiently different from the correct controls that are applied to the system then there is a failure. This small change deeply affects the main structure of the monitor. The initial distribution `x_prior` must now take into account a continuous interval of wheels'

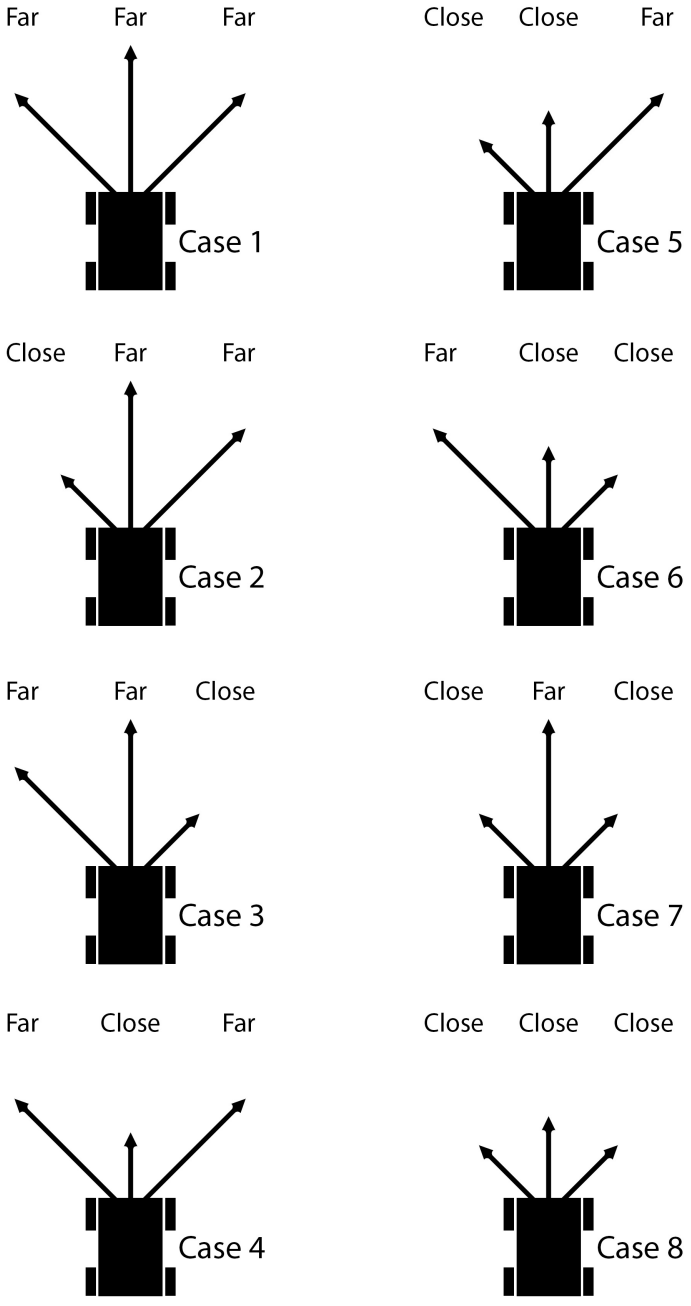


Figure 25: System modes of the new obstacle algorithm

velocities (i.e. a much larger range compared to the previous well-defined values). In fact, all particles that go to one of the three Error modes ($D = 3$, $D = 4$ and $D = 5$), must cover a wider map (i.e. $[0, 255] \times [0, 255]$), implying that a higher number of particles are needed to reach a sufficiently high accuracy of the monitor.

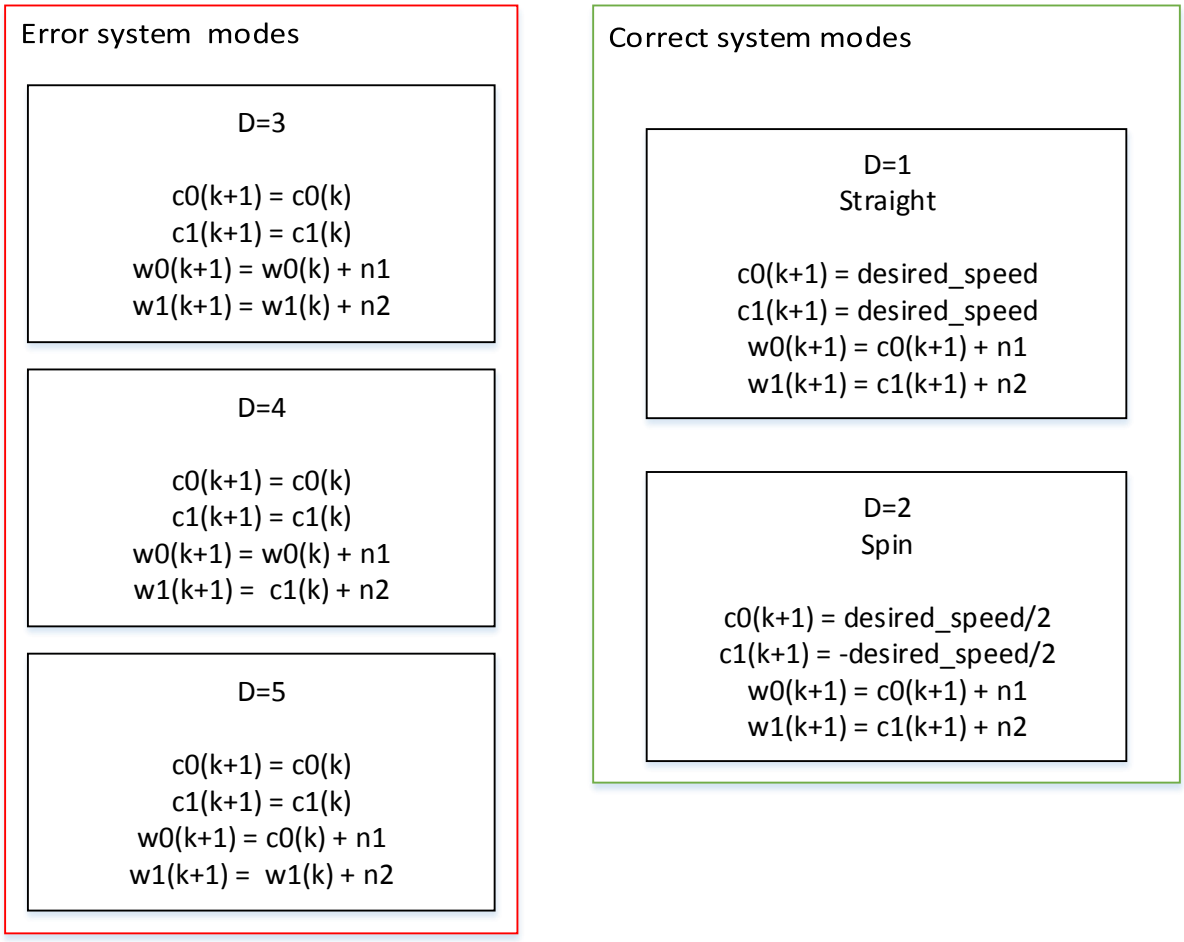


Figure 26: Outputs of the new system State model (only Cases 1), employed inside the monitor

Let us take a look to Figure 26 that describes part of the new system model. If a particle is in an Error mode ($D = 3$, $D = 4$ and $D = 5$), then two possible cases have to be taken into account:

- Error mode 3.
  - Case 1. If $D_{k+1} = 3 = D_k$ then:

$$w0_{k+1} = w0_k + n1 \qquad w1_{k+1} = w1_k + n2 \tag{6.1}$$

    that is, since a malfunctioning of both left and right motors has been detected, the new predicted wheels' velocities are equal to the previous velocities plus some noise
  - Case 2. If $D_{k+1} = 3 \neq D_k$, then $\vec{w}_{k+1}$ must be chosen such that:

$$w0_{k+1} \in [-128, 127] \cap \{c0_k\} \qquad w1_{k+1} \in [-128, 127] \cap \{c1_k\} \tag{6.2}$$

- Error mode 4.
  - Case 1. If $D_{k+1} = 4 = D_k$ then:

$$w0_{k+1} = w0_k + n1 \qquad w1_{k+1} = c1_k + n2 \tag{6.3}$$

that is, since a malfunctioning of the left motors has been detected, the new predicted left wheels' velocities are equal to the previous velocities plus some noise. On the contrary, the right set of wheels is still behaving correctly

- Case 2. If $D_{k+1} = 4 \neq D_k$, then $w0_{k+1}$ must be chosen such that:

$$w0_{k+1} \in [-128, 127] \cap \{c0_k\} \qquad w1_{k+1} = c1_k + n2 \tag{6.4}$$

- Error mode 5.
  - Case 1. If $D_{k+1} = 5 = D_k$ then:

$$w0_{k+1} = c0_k + n1 \qquad w1_{k+1} = w1_k + n2 \tag{6.5}$$

This is the perpendicular situation of the previous case: there is a malfunctioning on the right motors, but the left motors are still behaving correctly.

  - Case 2. If $D_{k+1} = 5 \neq D_k$, then $w1_{k+1}$ must be chosen such that:

$$w0_{k+1} = c0_k + n1 \qquad w1_{k+1} \in [-128, 127] \cap \{c1_k\} \tag{6.6}$$

The reason why "Cases 2" has to be considered is because particles must spread across the whole range of possible wheels' velocities interval, when there is a malfunctioning of the motor(s).

6.3 Introducing a transition state

Currently, the control system of the prototype, as well as the system monitor, do not model the transition phases between $D = 1$ and $D = 2$. In other words, since an on-off controller that abruptly switches between the two modes (”straight” or ”spin”) has been employed, the transition time for the wheels, even if reduced by the presence of a PID control inside the Arduino board, is long enough to be detected by the encoder sensors (refresh rate set at ˜3 Hz). As the model of the system inside the monitor does not take into account those transitions, they could be interpreted as a failure of the system. Let us take a closer look at a log of data taken from the system and from the monitor (Figure 27) (200 particles employed). The control signal changes from mode 1 to mode 2 at $t = 20$. The monitor detects the change and with a fixed delay reduces the probability of being in mode $D = 1$ and goes to 0 at $t = 22$ accordingly. At $t = 23$ the monitor starts to increase the probability of being in mode $D = 2$: the monitor now notices a similarity between the observations and the estimated output of the particles whose mode in $x_{i,k+1}$ is $D_{k+1} = 2$. The probability of being in $D = 2$ (i.e. the normalized number of particles in $D_{k+1} = 2$) goes to 1 at $t = 25$. It is easy to notice that for $t \in \{[23, 25], [44, 46], [51, 53]\}$ the monitor detects a non-zero probability of being in an Error mode, but the car did not experience any failure. The reason why it happened is directly related to a discrepancy inside the model of a transition system of the monitor: the current model did not properly approximate the real behaviour of the system.

Let us now take a closer look at two possible solutions of the problem:

- Add a transition state inside the property automaton of the monitor (Case 1)

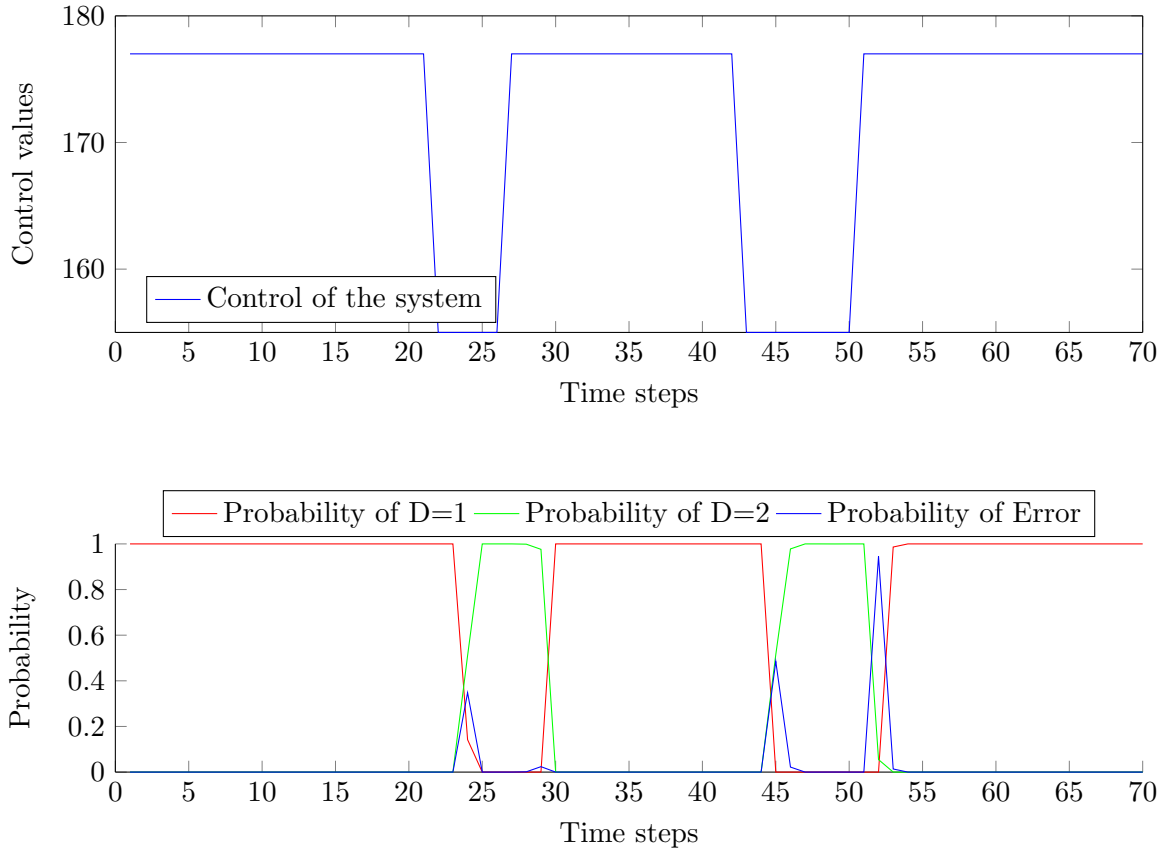


Figure 27: Plot of the probability of being in mode 1, 2 or Error mode versus the real state of the system

- Add two transition states inside the system state of the monitor (Case 2)

6.3.1 Case 1

A new state inside the Property Automaton has been added ($Q = 4$), as shown in Figure 28. The transition condition can be easily implemented by satisfying two conditions:

- There must be a malfunctioning of both set of wheels (all the motors are affected by the transient)
- The estimation of the encoder readings has to be within a predetermined range

If both of those conditions are verified then the Properly Automaton can correctly interpret the behaviour of the car. Note that adding a new state in the Property Automaton implies that the matrix of particle distribution is now 5-by-4 with a total number of states equal to 20.

6.3.2 Case 2

Let us now focus on the problem of deriving a more accurate model of the control system that could take into account the physical limitations of the motors (Figure 29). One of the possible solutions would be to create two different states: $D = 6$ and $D = 7$ depending on which transition state we want to model: from mode 1 to mode 2 or vice-versa. The expected outputs in both modes would be strictly related to the physical performances of the electric motors. Figure 29 shows a flowchart of a partial control system (error states are not present). Additionally, note that this solution refers to a case where a complete model of the system (i.e. can always estimate the evolution of an arbitrary set of observations) is given.

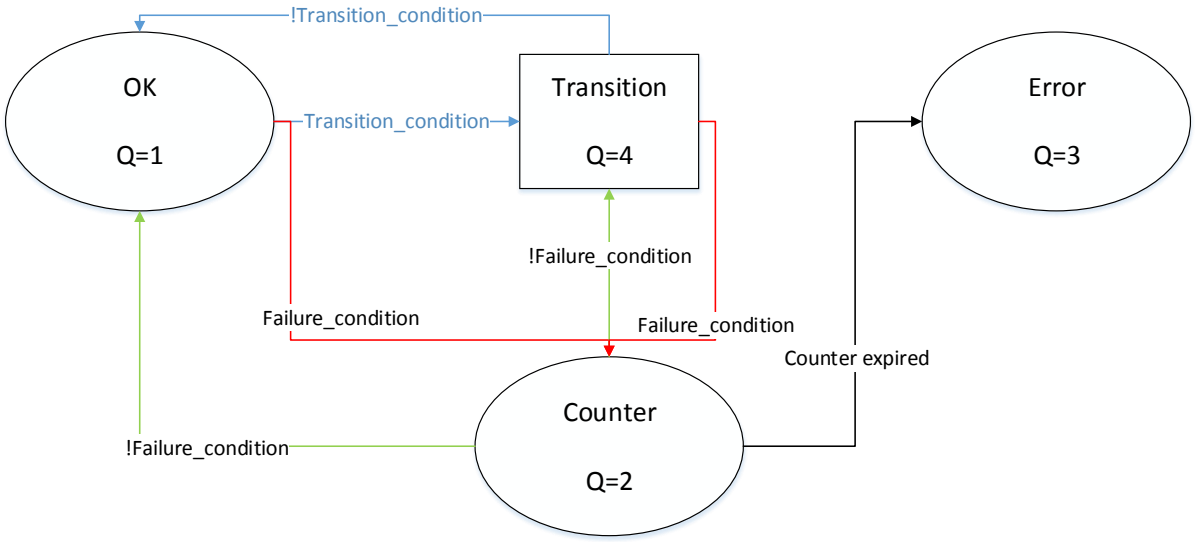


Figure 28: New transition state in the property automaton

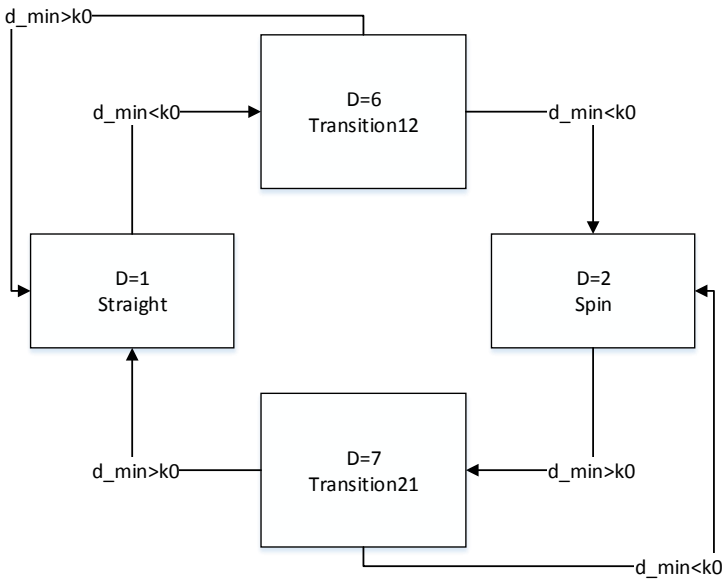


Figure 29: New transition modes in the model of the control system. The figure shows a partial model of the system (no Error modes)

Here, the matrix of particle distribution is 7-by-3 with a total number of states equal to 21. The main downside of this implementation is related to higher number of states that the monitor must take into account. This would definitely have an impact on the number of particles that the monitor needs to compute (to keep a sufficiently good estimation) and finally on the overall performances of the monitored system. On the other hand, this solution has a better system model, allowing a higher accuracy of the monitor.

6.3.3 Implementations

Before giving a more in-depth explanation about the implementation of the two above mentioned solutions, let us take a look at Figure 30.

The complex transient of the system to a drastic change of the controls has been approximated with an arbitrary monotonic function (not shown). This approximation may lead to a lower AA and RA of the monitor. Both Case 1 and Case 2 have been implemented correctly inside ROS. Note that adding a new state inside the Property Automaton is not feasible in the monitor mentioned in Chapter 5, since all wheels' values of the particles are set accordingly to a discrete number of outputs of the system states.

The system model does not take into account the possibility that a particle is in an Error mode. During the transition time, the monitor can not predict the evolution of the system and, thus, its intrinsic accuracy is lower. All particles will be re-sampled with a sufficiently low weight since none of those particles would be capable of sufficiently describing the current state of the system. In this framework, it is necessary to use the complex monitor mentioned in Section 6.2 as it is the only one that creates a sparse uniform distribution of particles. In both imple-

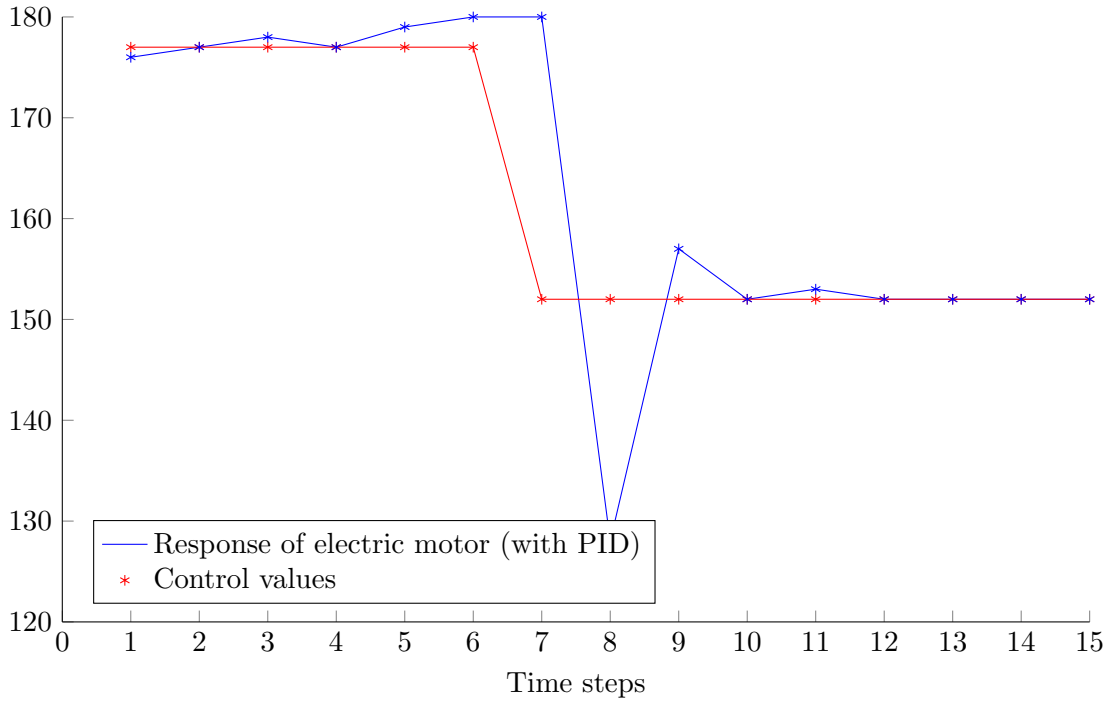


Figure 30: Plot of the response of the system versus sent control

mentations it is easy to notice that detecting a transition state may create more uncertainties about the system mode. For instance, the presence of high noise levels in the observations may be interpreted as a transient.

6.3.3.1 New property state inside the complex monitor

All particles that fall in the error state $D_{k+1} = 3$ are checked. If the following conditions are verified then the particle goes to $Q_{k+1} = 4$ (transition state):

$$w0_{k+1} \in \{c0_k - m, c0_k + m\} \qquad w1_{k+1} \in \{c1_k - m, c1_k + m\} \tag{6.7}$$

where m is a sufficiently small margin. Note that this solution basically changes the property automaton of a bad particle to $Q = 4$ under specific conditions. This may lead to a lower acceptance accuracy and rejection accuracy of the monitor. Additionally, since the response of the PID to a drastic change may have unpredictable transients depending on external conditions, it might be possible that a particle randomly generated temporarily describes the system better and consequently may be weighted more. In Figure 31 are the results of the estimations of the system with this implementation. The number of particles used is 5000, a substantially higher number than the one used in the previous experiment (Figure 27). This results in the system having to cover a much wider range of system states' outputs. All four transitions have been detected correctly.

6.3.3.2 New system states inside the monitor

Let us now take a look about a possible implementation of Case 2 (Subsection 6.3.2): the previous monitor (Chapter 5) has been employed. In Figure 33 the results of the monitor are shown together with the applied controls. Since two additional independent transition modes are present, it is possible to identify a transition $D = 1 \rightarrow D = 2$ as well as $D = 2 \rightarrow D = 1$. Due to the fact that no complete deterministic model is available, to properly implement the algorithm shown in Figure 29, stochastic transitions had to be introduced. Let us take a look at Figure 32. The only deterministic transition is between $D_k = 1 \rightarrow D_{k+1} = 4$, as $D_k = 1$ is the only mode where an accurate prediction of the distance vector $\vec{d}_{k+1}$ can be done. All other transitions ($p31$, $p32$, $p24$, $p42$, $p41$) are currently set to 50%.

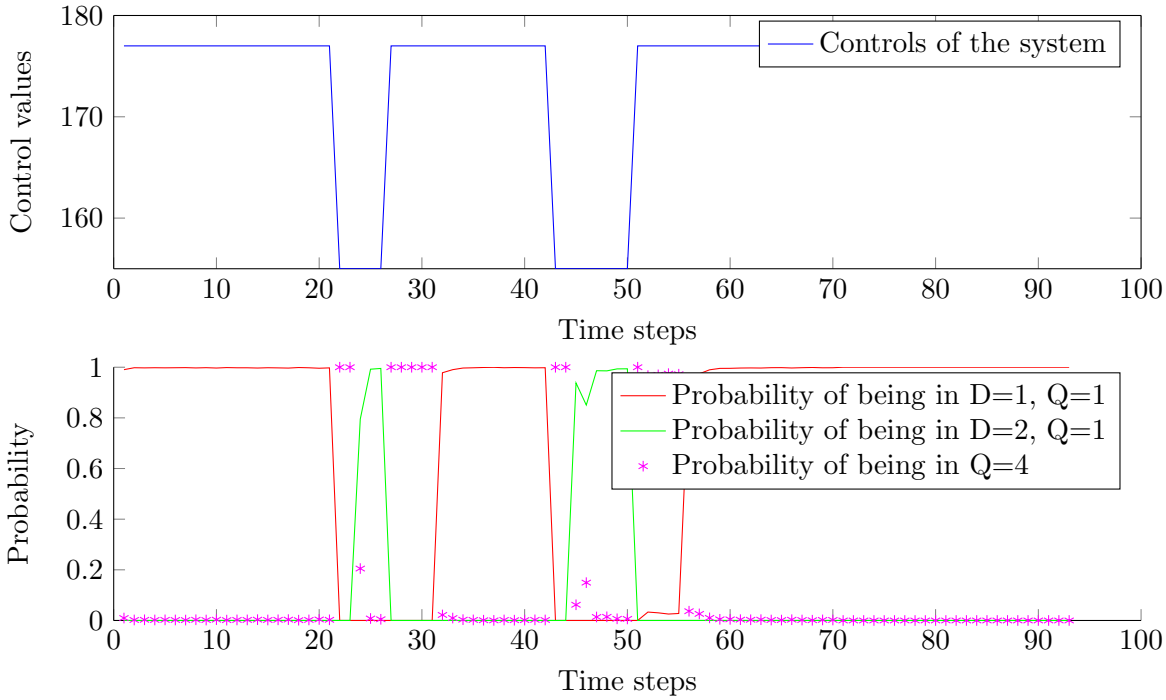


Figure 31: The picture shows a plot of the probability of being in "straight" $(D = 1, Q = 1)$, in "spin" $(D = 2, Q = 1)$ and in "transition" $Q = 4$ versus the applied controls

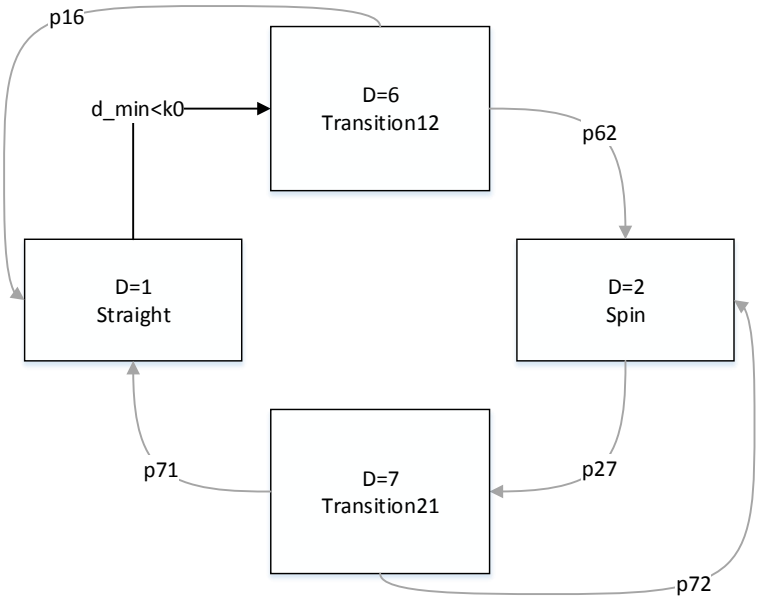


Figure 32: Implemented transition modes in the model of the control system. The figure shows a partial model of the system (no Error modes)

Concerning the outputs of the transition mode, let us focus first on mode $D = 6$, that is transition $D = 1 \rightarrow D = 2$.

If $particle_i : \ D_k = 1 \ \& \ D_{k+1} = 6$ then $\vec{c}_{k+1}$ is an arbitrary value such that:

$$c0_{k+1} \in (ds_k/2, ds_k) \qquad c1_{k+1} \in (-ds_k/2, ds_k) \tag{6.8}$$

If $particle_i : \ D_k = 6 \ \& \ D_{k+1} = 6$ then $\vec{c}_{k+1}$ is an arbitrary value such that:

$$c0_{k+1} \in (ds_k/2, c0_k) \qquad c1_{k+1} \in (-ds_k/2, c1_k) \tag{6.9}$$

In this way the approximated monotonic transient of the PID is estimated correctly. The complement of this algorithm has been implemented for the transient mode $D = 7$. Let us consider the results shown is Figure 33. All transitions have been correctly recognized, but here is it easy to notice that the AA is lower than in the previous case. This is due to the fact that all modes , except $D = 6$ and $D = 7$,(given a low noise level of the sensors) have well-defined predictions $\vec{w}_{k+1}$, that are easy to detect with good precision. On the contrary, a large number of particles are needed to estimate the vector $\vec{w}_{k+1}$ in modes $D = 6$ and $D = 7$ with reliable accuracy.

6.3.4 Final Remarks

The idea of creating a more accurate model of the system is definitely useful to properly detects false errors. On one hand, the complexity of the monitor and its performances can decrease drastically even in medium-complex systems. On the other hand, unless a very accurate model of the system is present (implying a high number of particles), then not all transients of the systems can be recognized properly. Further analysis can be made in systems modelled with more complex Markov chains about what is the right trade-off between accuracy of the model and performance.

6.4 Monitor accuracy

Let us suppose we want to measure the accuracy of the control model inside the monitor: in case the discrepancy of the best prediction and the real observation increases too much it is possible to raise a silent alarm, called "monitor error". In this way it would be taken into account the chance that a specific prediction may not be sufficiently reliable. Let us take a

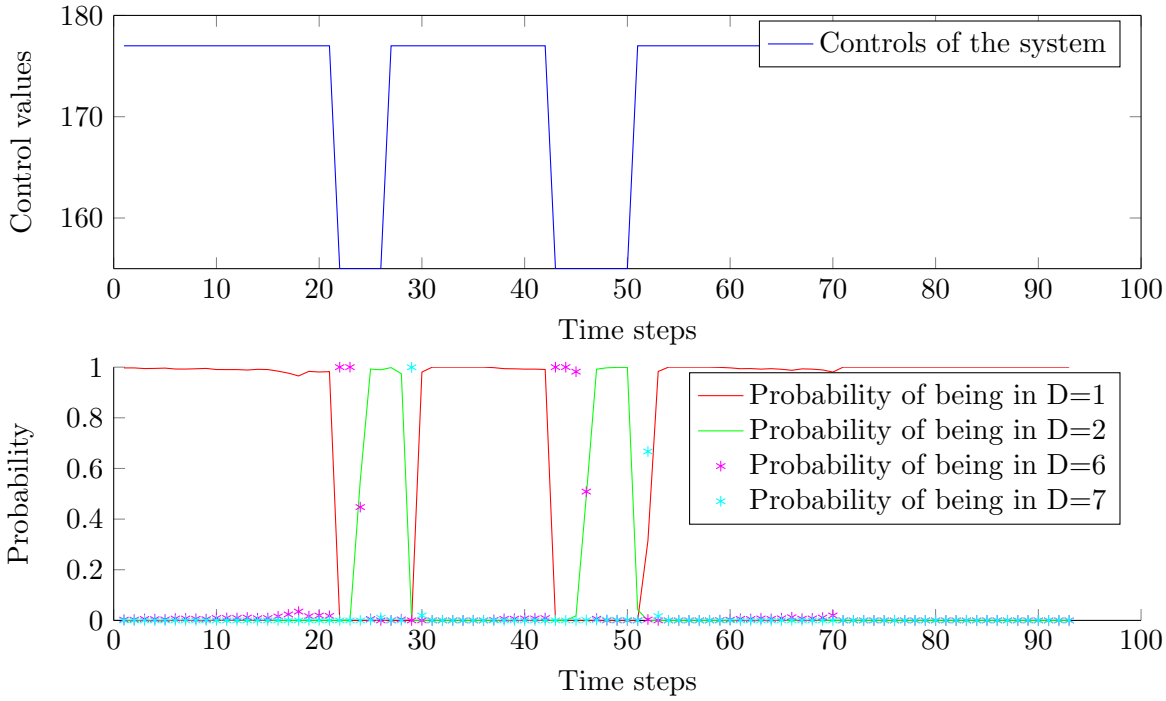


Figure 33: Plot of the probability of being in mode "straight" $(D = 1, Q = 1)$, in mode "spin" $(D = 2, Q = 1)$, in mode "transition $(D = 1 \rightarrow D = 2)$" $D = 6$ and in mode "transition $(D = 2 \rightarrow D = 1)$" $D = 7$ versus applied control

closer look to Figure 34. In this situation the standard monitor has been used (Chapter 5). Note that no transition states are present here. During each iteration of the particle filter, the highest weight w_i (among all of those that have been assigned to the particles, stored in `x_posterior`,) has been printed as a function of time. This value is directly proportional to how well system states $x_{i,k+1}$ of the related particle p_i approximates the current state of the system, given observation y_{k+1}, and it reflects the accuracy of the monitor itself.

It is easy to see that every time there is an edge in the sent control, a drastic decrease in precision is noticeable. The reason is strictly related to a lack of transition modes that model the transients of the electric motors and PIDs.

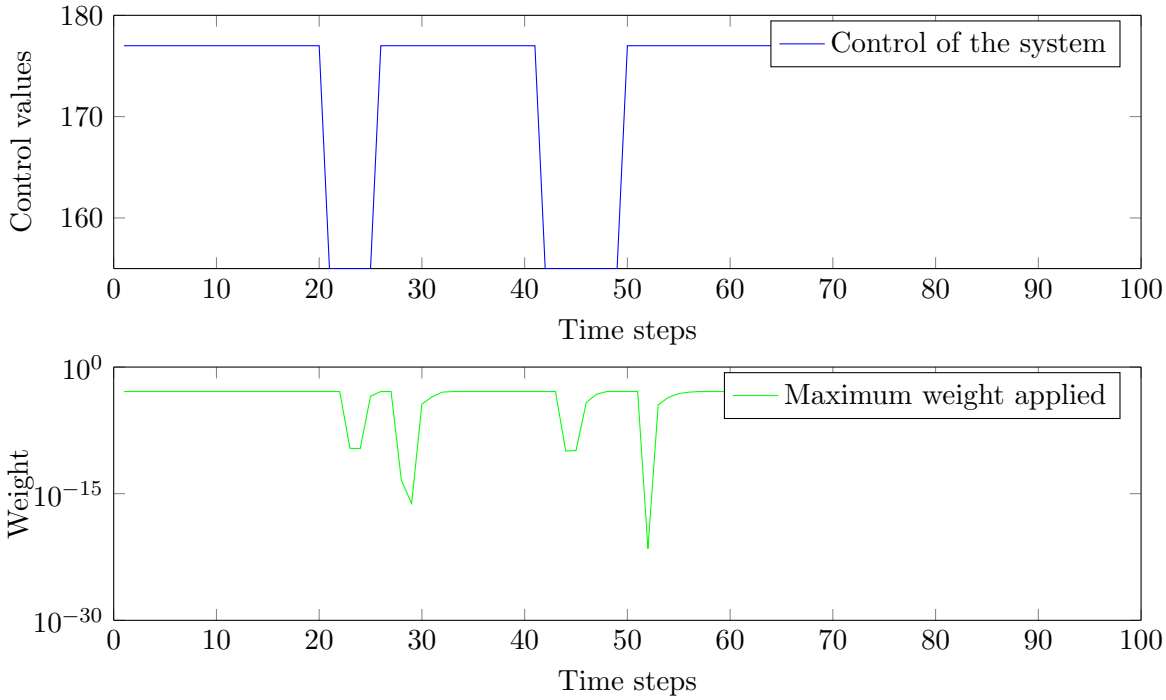


Figure 34: Weights in the standard monitor (without transition modes)

Let us now take a look at Figure 35. In this situation the monitor described in Case 2 has been employed: two modes have been added to the system so that transitions from mode 1 to mode 2 and vice-versa have been taken into account. A noticeable improvement can be noticed compared with the previous case. It can be seen that a small discrepancy of the monitor and the real system is still present, but it is several orders of magnitude smaller than in the previous case.

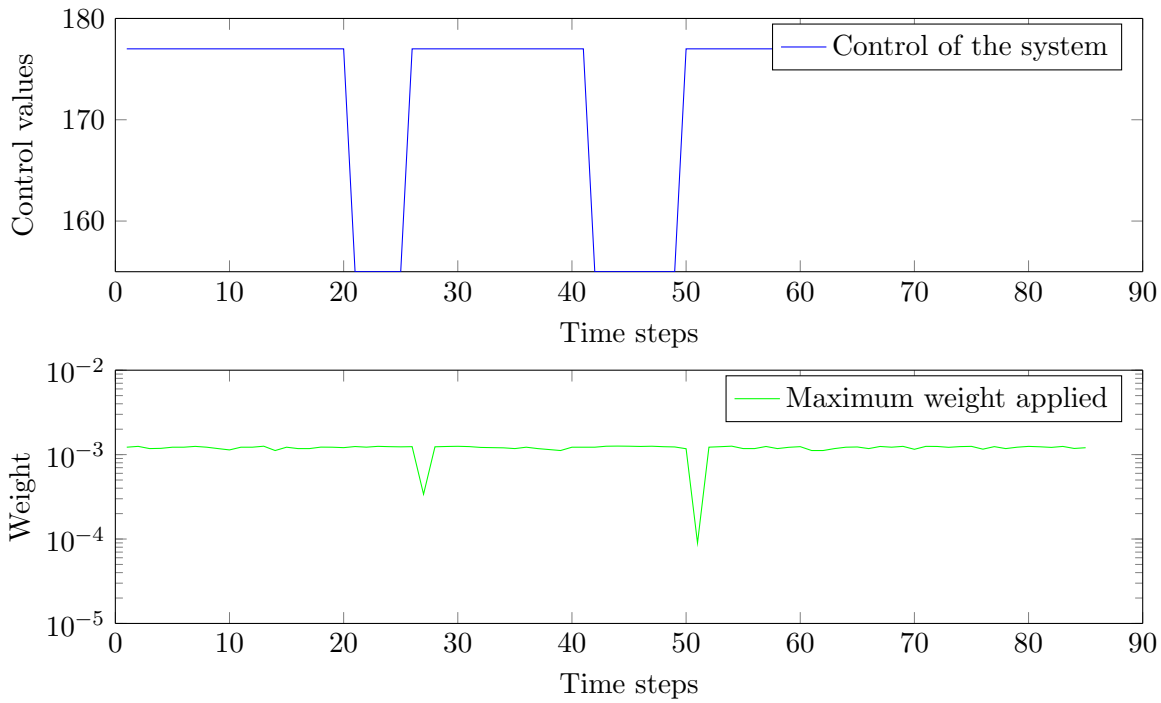


Figure 35: Weights in the monitor with transition modes

If the monitor accuracy drops drastically, then the monitor is no longer reliable. This information can be extremely useful in two different situations:

- If the accuracy is sufficiently low, a "monitor error" alarm could be raised; any other active alarms will be disabled and the monitor node restarts with all particles spread accordingly to the initial distribution. Note that in this situation, the monitor node is practically deactivated for a specific amount of time, since the system has observations that are not coherent with the system model inside the monitor node. This solution can be applicable when an arbitrarily high number of particles is used, but the system model of the monitor is not sufficiently accurate
- Let us consider a sufficiently accurate system, with an arbitrarily high number of system modes. Some of the modes may have well-defined outputs that can be described with a reasonable low number of particles. However, other modes' outputs may be continuous. In this last case a higher number of particles would be necessary to achieve a good precision. For optimization purposes, let us run the particle filter with a low number of particles. Let us suppose that with an arbitrary time instant $t = k + 1$ the accuracy drops, such that:

$$\nexists x_{i,k+1} \in \texttt{x\_posterior} : w_i >> me_{th} \tag{6.10}$$

where me_{th} is the threshold of the model error. Then, new predictions of `x_posterior` for the same time instant may be computed again until the wanted accuracy is reached. Note that a maximum number of iterations must be decided a priori since the model node

is executed in real-time. In this case the computational workload is adaptive: it depends on the contingency of the moment

6.5 Introducing noise in the observations

Let us consider a system where observations are affected by a various levels of noise. Figure 36 shows the control applied to the wheels. At $t = 21$, $t = 26$, $t = 42$ and $t = 50$ are present four transitions, respectively: $D = 1 \rightarrow D = 2$, $D = 2 \rightarrow D = 1$, $D = 1 \rightarrow D = 2$ and $D = 2 \rightarrow D = 1$. To properly test the behaviour of the particle filter, variable noise levels have been added to the observations:

$$\bar{y} = y + \vec{n} = \{\vec{s} + n\vec{1}, \vec{e} + n\vec{2}\} \tag{6.11}$$

In the following plots the monitor without transition states has been employed. Note that the system state of the monitor node has been customized depending on the noise levels of the observations.

Let us take a closer look at Figure 27: it represents a situation where no additional noise has been added to the observation. Let us now add 5 units of noise to $\vec{y}$ (Figure 37). It is easy to notice that in the probability of being in an Error mode increases moderately when there is a transient (not considered the system model of the monitor node). On the contrary, no change can been seen in any other situation.

In the next two plots (Figure 38 and Figure 39), it is even more clearly observed that with higher noise levels an unmodelled situation may give rise to to a higher number of incorrect

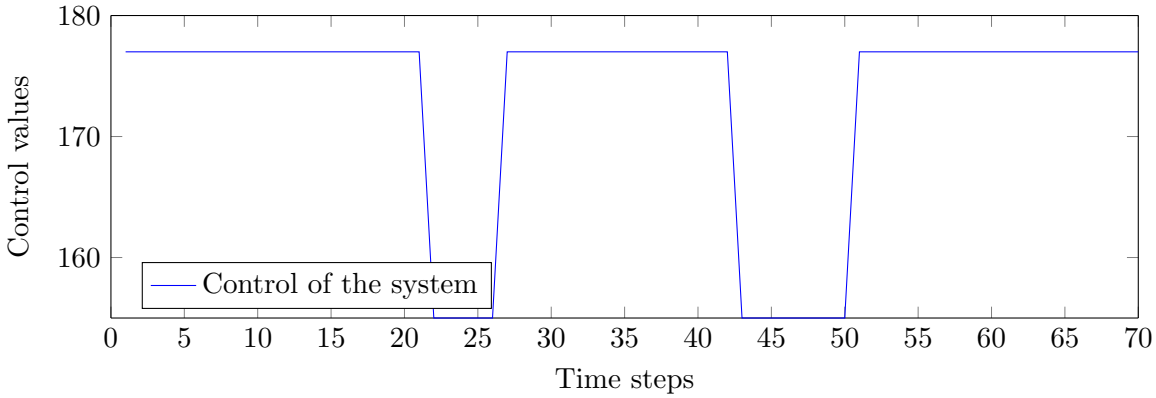


Figure 36: Applied control values

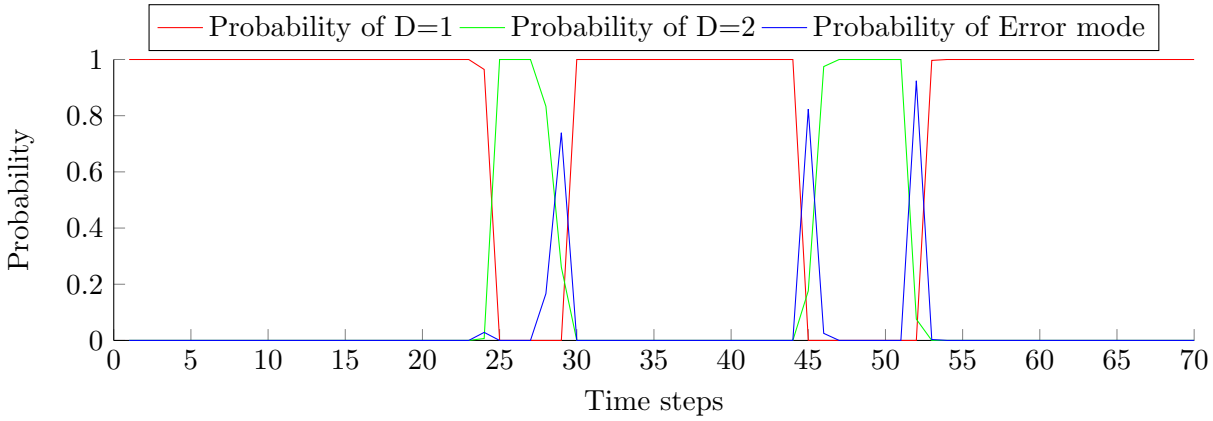


Figure 37: Noise level: 5 units. Plot of the probability of being in mode 1, 2 or Error mode versus the real state of the system

estimations of the particle filter. The current desired speed is $ds = 50$. Note that during the "spin" mode the absolute value of the applied control is half of desired speed. In addition, on top of the manually added noise, you also have to take into account the presence of the intrinsic noise of the real sensors. Under these conditions, a noise level of 20 units represents a threshold after which the monitor node may not immediately detect even well-modelled system modes (e.g. $D = 1$ and $D = 2$)

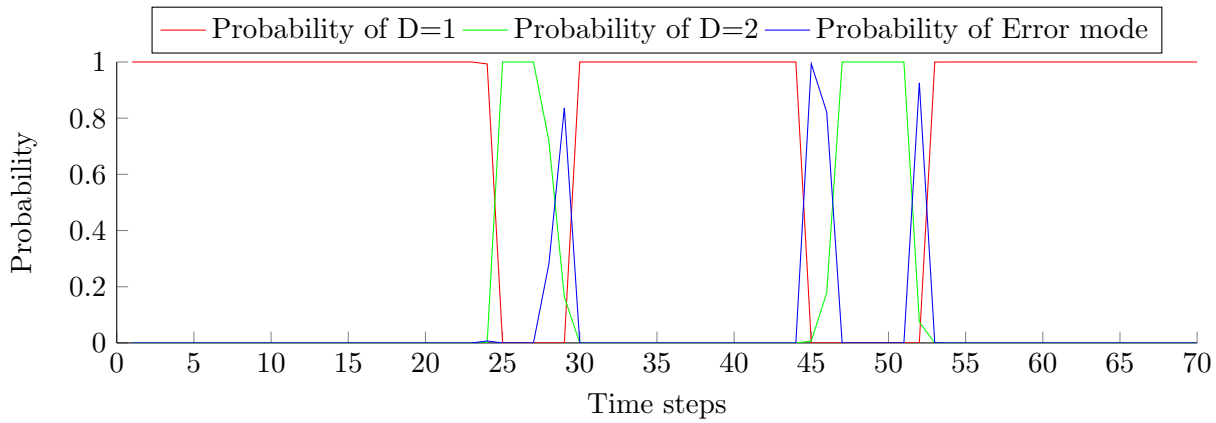


Figure 38: Noise level: 10 units. Plot of the probability of being in mode 1, 2 or Error mode versus the real state of the system

Let us now take a closer look at Figure 40. It is clear that the unmodelled transient at $t = 23$, together with reasonably high levels of noise in the observations, deeply affects the reliability

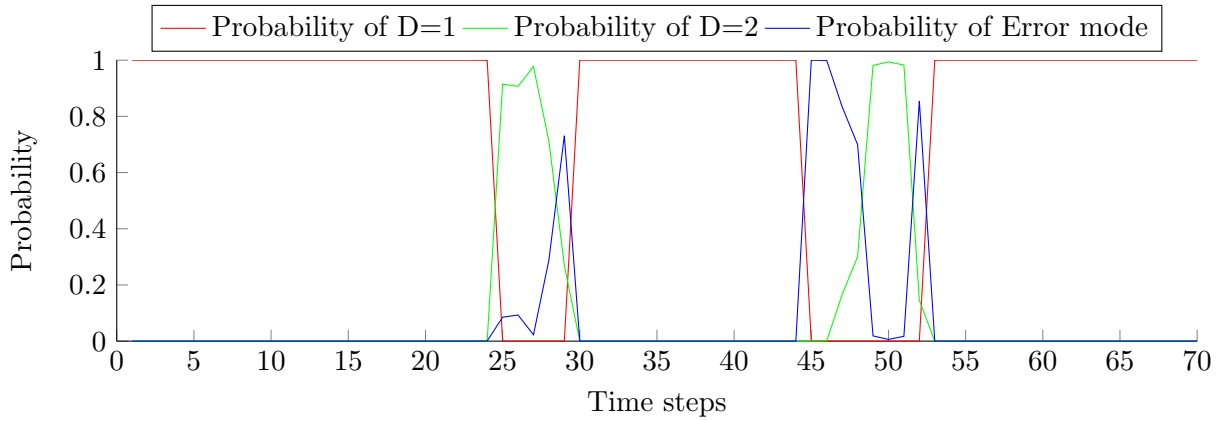


Figure 39: Noise level: 20 units. Plot of the probability of being in mode 1, 2 or Error mode versus the real state of the system

of the monitor node. The probability of being in mode $D = 2$ is critically low compared to the probability of being in an Error mode. On the other hand, since:

$$\sum_{t=0}^{\infty} n_t = 0 \tag{6.12}$$

then before $t = 50$ (new transient), the probability of being in $D = 1$ rapidly increases.

In the last plot (Figure 41), extremely high levels of noise have been added (25 units). The monitor node is no longer able to recover from an unmodelled situation.

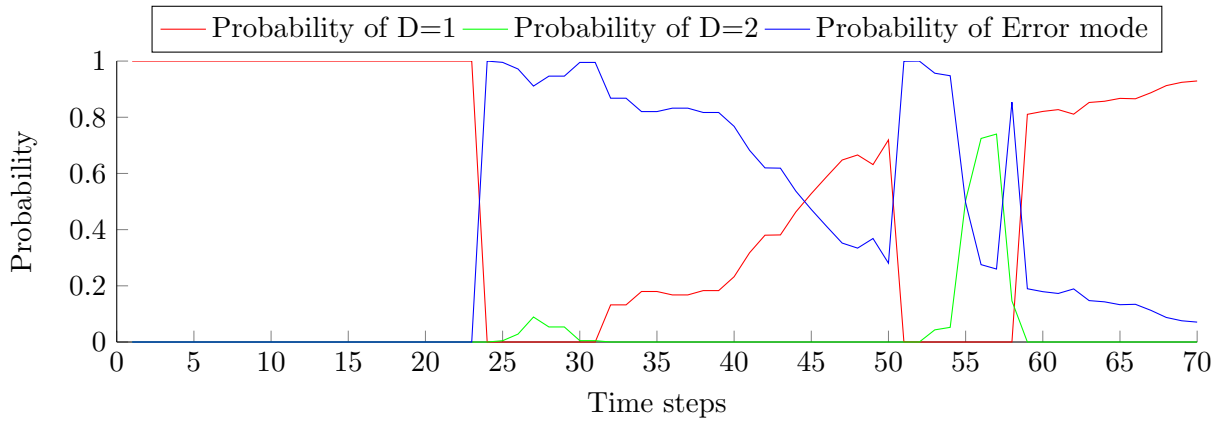


Figure 40: Noise level: 22 units. Plot of the probability of being in mode 1, 2 or Error mode versus the real state of the system

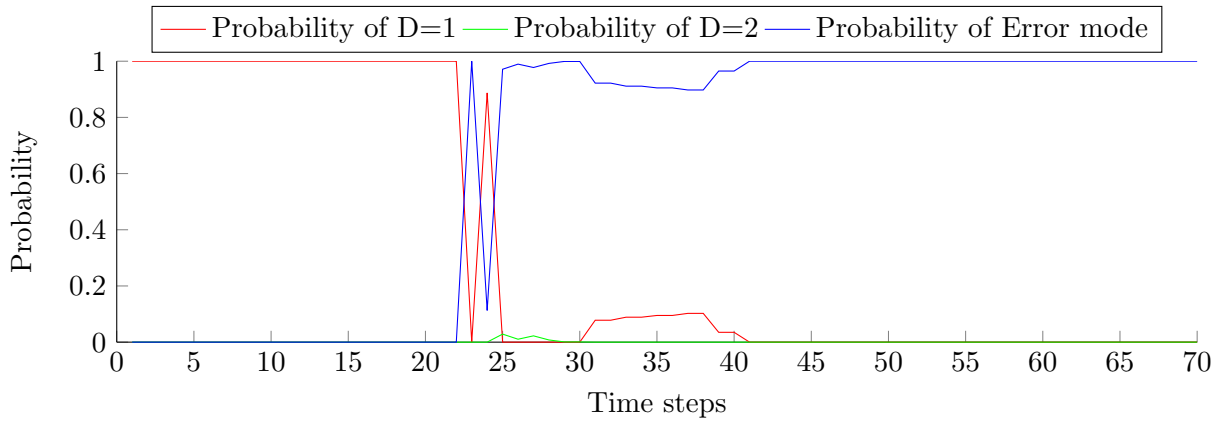


Figure 41: Noise level: 25 units. Plot of the probability of being in mode 1, 2 or Error mode versus the real state of the system

CHAPTER 7

CONCLUSION

Throughout the entire research particular detail has been given to the physical implementation of a number of monitor nodes. The employed monitoring techniques represent one of the most feasible solutions for a real-time and dynamic verification of complex cyber-physical systems. The research touched on some of the most challenging aspects of monitoring theory: real-time internal monitoring and monitor accuracy. Successfully implementing -on a working prototype- the latest monitoring algorithms represents a key step towards the design of better cyber-physical systems. The idea that control systems and monitoring systems can be considered as two well-defined and distinct entities may lay down the foundations of a new era of safer and more reliable automaton systems.

Great effort was paid to engineering a prototype that that could be used as flexible platform for a number of tests. All the stages, from the physical layer up to the application layer have been correctly realized. The test methodology that has been employed throughout the entire research allowed to successfully implement, on a realistic system, the most relevant theories of monitorability. The safety property that was initially designed was later integrated by a system model that, by means of a state-estimation algorithm, allowed to precisely determine the inner state of the system. Several monitors with different characteristics, have been successfully employed on the robot. Experimental results, which corroborate theoretical analysis, of all monitor versions have been shown, explained and compared with each other.

At last, great importance has been given to the influence of discrepancies in the system model. In this framework, it was also taken into account the physical response of several actuators whose models were designed and tested with different levels of precision. The monitor accuracy of several monitor version has been compared. Furthermore, the reader has been introduced to the new concept of "monitor error", presented in the last chapter of the thesis. In the end, the response of the monitor, where observations were affected by a number of noise levels, has been analyzed.

Future developments of the research may be conducted on more complex monitoring systems that integrate advanced obstacle avoidance control systems together with precise PID transient models. Further implementations may also focus on monitors whose reliability may be directly controlled by means of additional entities that, by checking its intrinsic monitor accuracy at each time step, determine how reliable the monitor is - and consequently - how safe the overall system is. In addition, the integration of a camera system mounted on the prototype could allow more precise detection of obstacles and might represent one of the most challenging and interesting areas of future work.

APPENDICES

Appendix A

SKETCHES OF THE PROTOTYPE

Appendix A (continued)

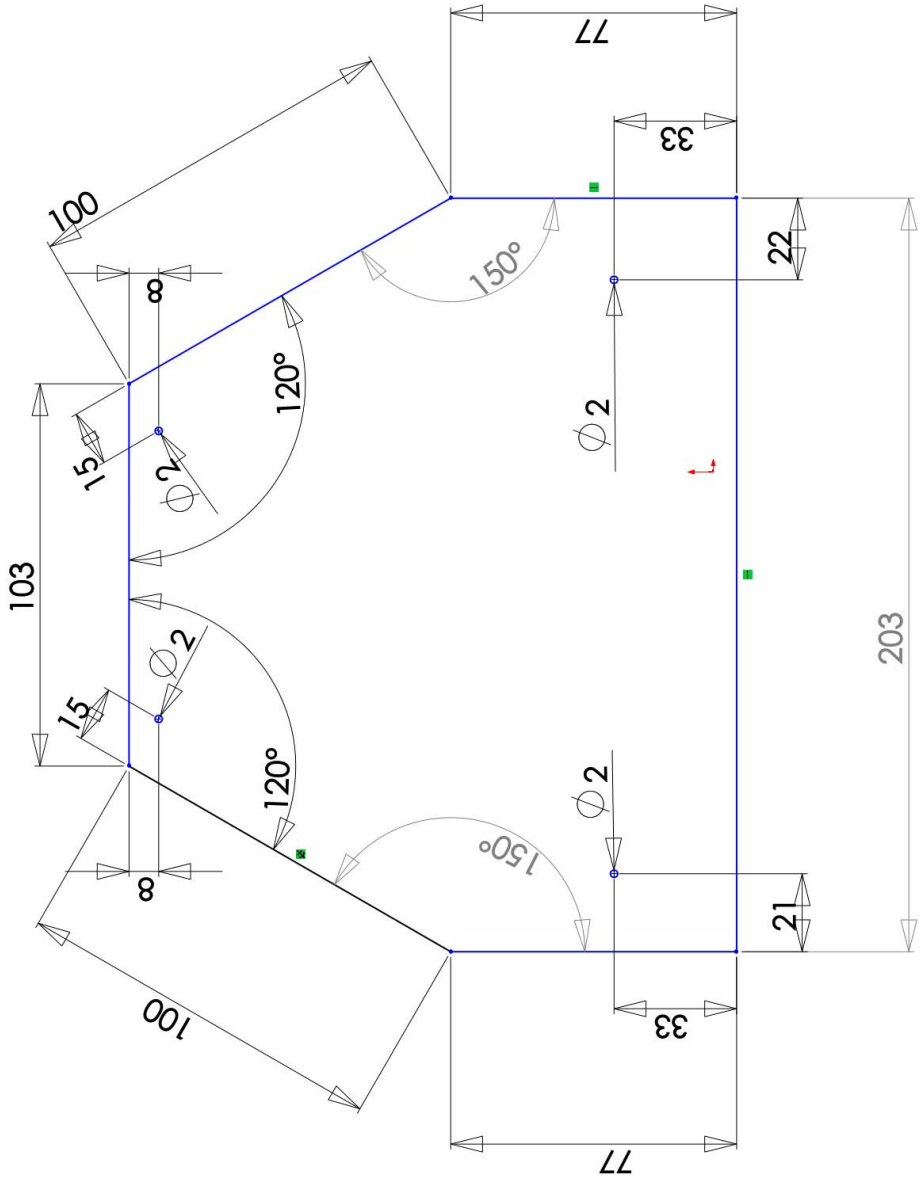


Figure 42: Top sensor case

Appendix A (continued)

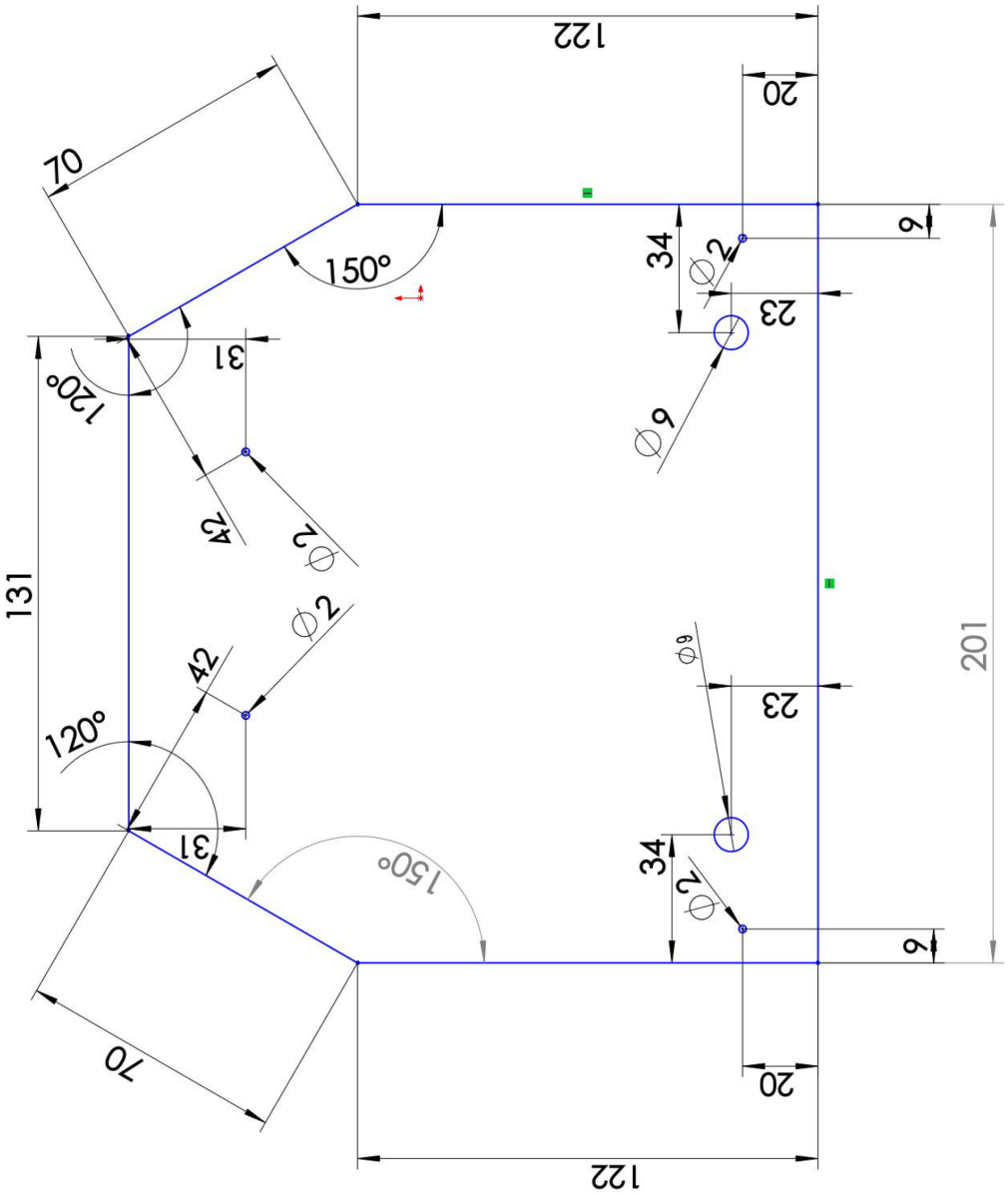


Figure 43: Bottom sensor case

Appendix A (continued)

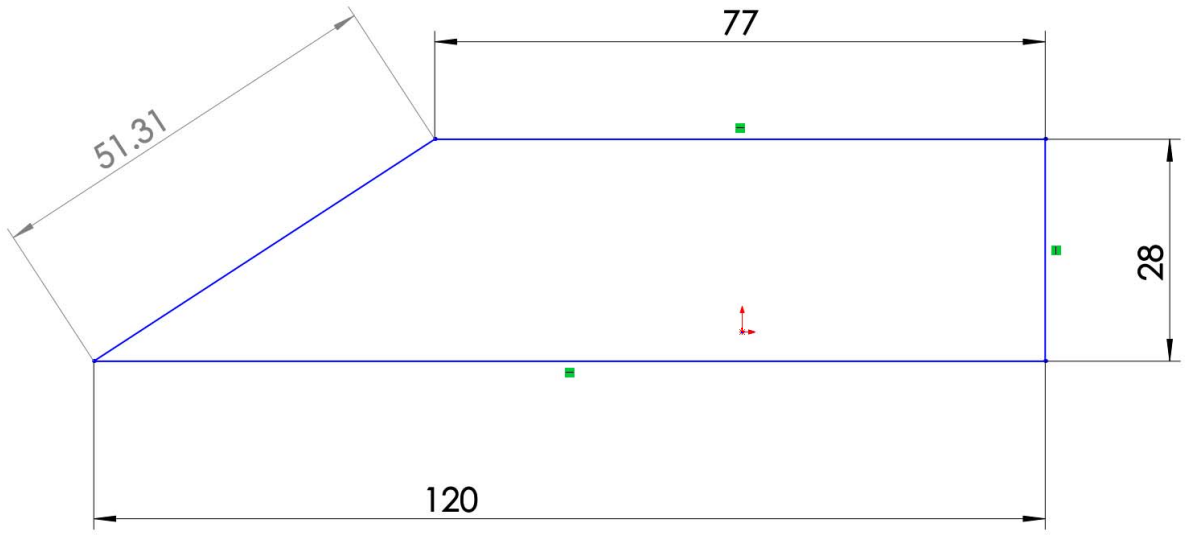


Figure 44: Lateral sensor case, piece 1-5

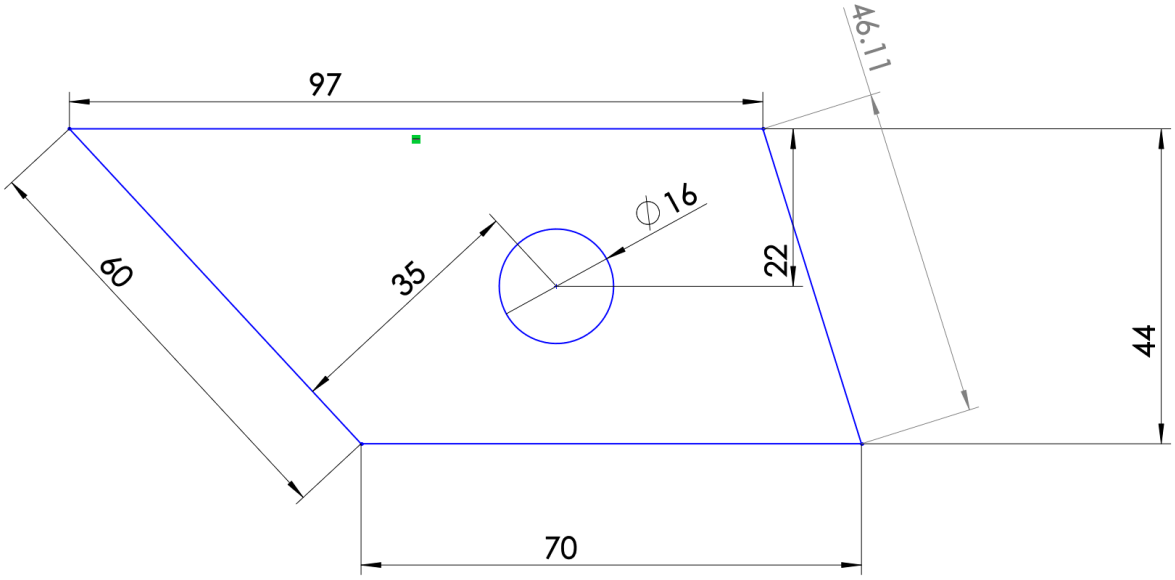


Figure 45: Lateral sensor case, piece 2-4

Appendix A (continued)

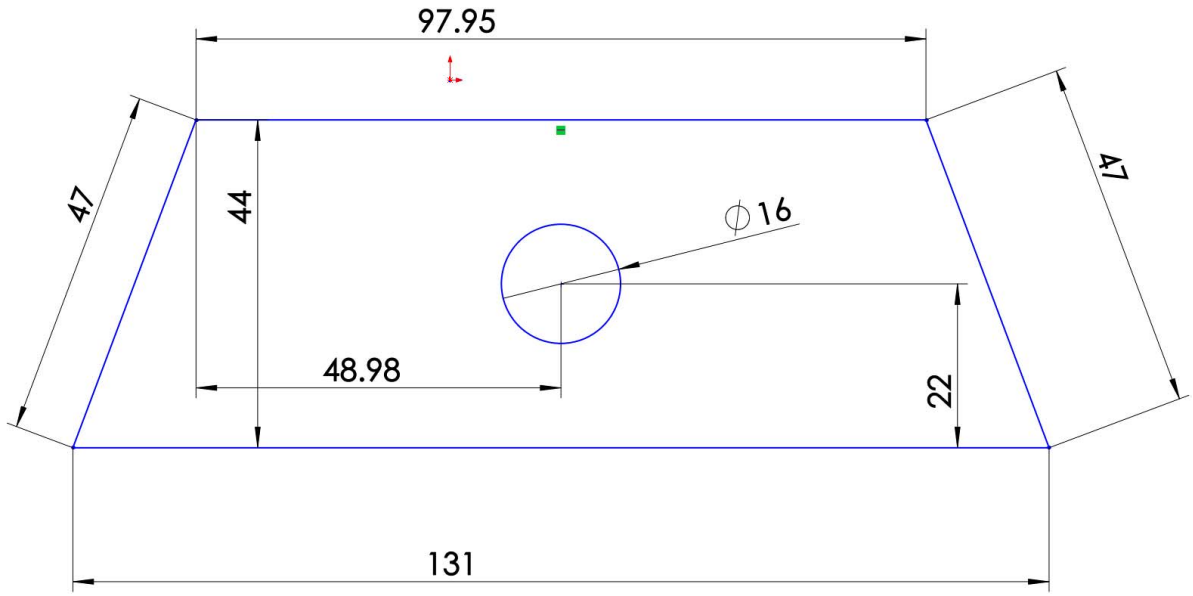


Figure 46: Lateral sensor case, piece 3

Appendix A (continued)

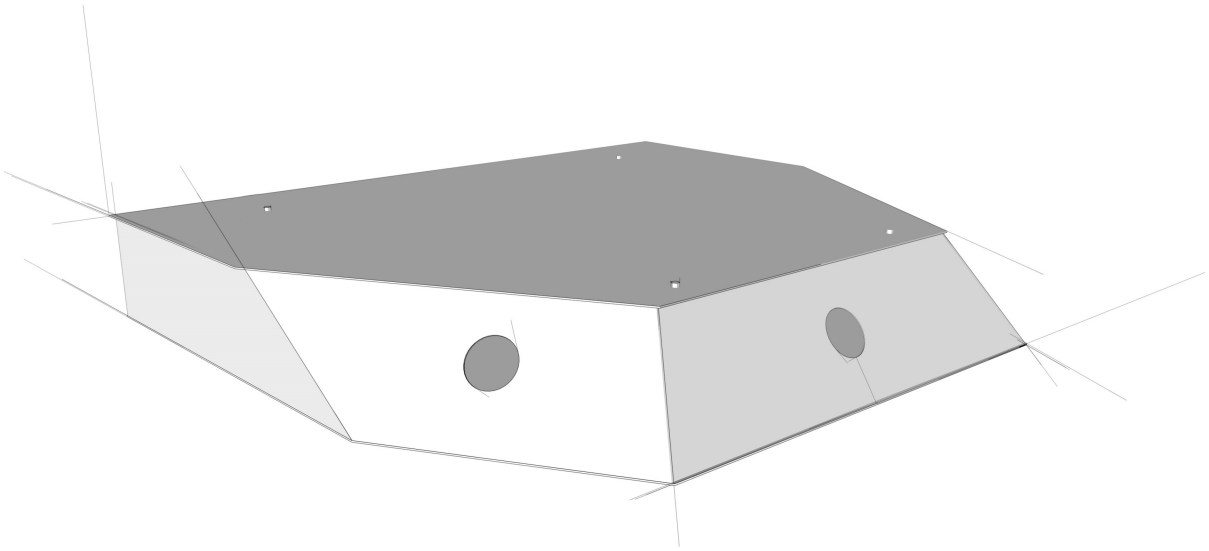


Figure 47: Final render of the sensors case

Appendix B

TEMPORAL LOGIC OPERATOR

The following table (Table III) shows the temporal logic operators that are used throughout this thesis.

TABLE III: MOST COMMON TEMPORAL LOGIC OPERATORS

Operator	**Name**	**Example**
$\cup$	Until	$g \cup h$
$\bigcirc$	Next Time	$\bigcirc g$
$\diamond$	Eventually	$\diamond g$
$\boxdot$	Always	$\boxdot g$
W	Weak Until	GWh
$\ominus$	Last Time	$\ominus h$
S	Since	g S h
$\bowtie$	Sometime in the Past	$\bowtie h$
$\boxminus$	Always in the Past	$\boxminus h$

CITED LITERATURE

1. Thrun, S., Burgard, W., and Fox, D.: Probabilistic robotics. MIT press, 2005.

2. Bauer, A., Küster, J.-C., and Vegliach, G.: The ins and outs of first-order runtime verification. Formal Methods in System Design, pages 1–31, 2015.

3. Fersman, E., Krcal, P., Pettersson, P., and Yi, W.: Task automata: Schedulability, decidability and undecidability. Information and Computation, 205(8):1149–1172, 2007.

4. Leucker, M. and Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming, 78(5):293–303, 2009.

5. Sistla, A. P., Zhou, M., and Zuck, L. D.: Monitoring off-the-shelf components. In Verification, Model Checking, and Abstract Interpretation, pages 222–236. Springer, 2006.

6. dAmorim, M. and Roşu, G.: Efficient monitoring of ω-languages. In Computer Aided Verification, pages 364–378. Springer, 2005.

7. Parzen, E.: Stochastic processes, volume 24. SIAM, 1999.

8. Blunsom, P.: Hidden markov models. Lecture notes, August, 15:18–19, 2004.

9. Cappe, O., Moulines, E., and Riden, T.: Inferencing in hidden markov models, 2005.

10. Papoulis, A. and Pillai, S. U.: Probability, random variables, and stochastic processes. Tata McGraw-Hill Education, 2002.

11. Doucet, A., De Freitas, N., and Gordon, N.: An introduction to sequential Monte Carlo methods. Springer, 2001.

12. Doucet, A., Gordon, N. J., and Krishnamurthy, V.: Particle filters for state estimation of jump markov linear systems. Signal Processing, IEEE Transactions on, 49(3):613–624, 2001.

CITED LITERATURE (continued)

13. Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T.: A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. Signal Processing, IEEE Transactions on, 50(2):174–188, 2002.

14. Sistla, A. P., Žefran, M., and Feng, Y.: Monitorability of stochastic dynamical systems. pages 720–736, 2011.

15. Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P.-H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. Springer, 1993.

16. Sistla, A. P., Žefran, M., and Feng, Y.: Runtime monitoring of stochastic cyber-physical systems with hybrid state. pages 276–293, 2012.

17. Sistla, A. P., Žefran, M., Feng, Y., and Ben, Y.: Timely monitoring of partially observable stochastic systems. pages 61–70, 2014.

18. Khurshid, S. and Sen, K.: Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers. Springer, 2012.

19. Gondi, K., Patel, Y., and Sistla, A. P.: Monitoring the full range of ω-regular properties of stochastic systems. In Verification, Model Checking, and Abstract Interpretation, pages 105–119. Springer, 2009.

20. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y.: Ros: an open-source robot operating system. In ICRA workshop on open source software, volume 3, page 5, 2009.

21. Araújo, A., Portugal, D., Couceiro, M. S., and Rocha, R. P.: Integrating arduino-based educational mobile robots in ros. Journal of Intelligent & Robotic Systems, 77(2):281–298, 2015.

22. Schmidt, M.: Arduino. Pragmatic Bookshelf, 2011.

23. Wei, S.: Smooth path planning and control for mobile robots. 2005.

24. Franklin, G. F., Powell, J. D., and Workman, M. L.: Digital control of dynamic systems, volume 3. Addison-wesley Menlo Park, 1998.

CITED LITERATURE (continued)

25. Sammapun, U., Lee, I., and Sokolsky, O.: Rt-mac: runtime monitoring and checking of quantitative and probabilistic properties. In Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on, pages 147–153. IEEE, 2005.

26. Pnueli, A., Zaks, A., and Zuck, L.: Monitoring interfaces for faults. Electronic Notes in Theoretical Computer Science, 144(4):73–89, 2006.

27. Burguera, A., González, Y., and Oliver, G.: Sonar sensor models and their application to mobile robot localization. Sensors, 9(12):10217–10243, 2009.

28. Valiant, L. G. and Paterson, M. S.: Deterministic one-counter automata. Journal of Computer and System Sciences, 10(3):340–350, 1975.

29. Russell, S. J. and Norvig, P.: Artificial intelligence: a modern approach (international edition). 2002.

30. Brooks, R. A.: A robust layered control system for a mobile robot. Robotics and Automation, IEEE Journal of, 2(1):14–23, 1986.

31. Di Benedetto, M. D. and Sangiovanni-Vincentelli, A.: Hybrid Systems: Computation and Control: 4th International Workshop, HSCC 2001 Rome, Italy, March 28-30, 2001 Proceedings, volume 4. Springer Science & Business Media, 2001.

32. MaxBotix: Xl-maxsonar-ez datasheet. `http://maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf`, 2005-2012. original document from MaxBotix.

33. DimensionEngineering: Sabertooth 2x12. `https://www.dimensionengineering.com/datasheets/Sabertooth2x12.pdf`, 2012. original document from DimensionEngineering.

34. Ros website. `http://www.ros.org/`. Accessed: 2014-09-01.

35. Wikipedia website. `https://www.wikipedia.org/`. Accessed: 2014-09-01.

36. Arduino website. `http://www.arduino.cc/`. Accessed: 2014-09-01.

VITA

NAME	Ruggero Balteri
EDUCATION	
	Master of Science in Electrical and Computer Engineering, University of Illinois at Chicago, August 2015
	Specialization Degree in Devices and Technologies for Microelectronic Circuit and Optoelectronics, Jul 2015, Polytechnic of Turin, Italy
	Bachelor's Degree in Electronic Information Engineering, Jan 2013, Tongji University, China
	Bachelor's Degree in Information Technology Engineering (ITE) - Electronics, Jul 2012, Polytechnic of Turin and Polytechnic of Milan, Italy
WORK EXPERIENCE	
Sep 2015 - Sep 2016	Selected for the industry-oriented student exchange program for EU students "Vulcanus in Japan 2015-2016"
Dec 2014 - May 2015	Research Assistant at UIC, focusing on monitors for cyber physical systems, robotics and control engineering
Sep 2014 - Dec 2014	Research Assistant at UIC, focusing on robotics and artificial intelligence
Sep 2012 Dec 2012	4-month paid internship at the Selerant headquarter in Shanghai, supporting the QC team and the IT team
May 2012	6-month internship at the "Wiicom" company, working on the development of an embedded Wi-Fi module and webserver
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	March 2013 - IELTS examination (8.0/9)
	A.Y. 2014/15 One Year of study abroad as transfer student in Chicago, Illinois
	A.Y. 2013/14. Lessons and exams attended exclusively in English

VITA (continued)

	A.Y. 2010/11. Lessons and exams attended exclusively in English
Chinese	Advanced (5000 words)
	Currently preparing HSK 6, native speaker level (5000 Chinese words studied)
	A.Y. 2010/11 and 2012/13 - Two Years of study abroad in Shanghai
	June 2013 - HSK 5 exam (3000 Chinese words studied)
	April 2013 - Passed HSK 4 exam
	Nov 2012 - Six months language course at Tongji university
	AY 2011/12 - Language courses at the University of Turin (Faculty of Foreign Languages)
	AY 2010/11 at the Tongji University: two language courses (six months each) and two other courses of Chinese economy and culture (three months each)
	AY 2009/10 - Three months Chinese course at the Polytechnic of Turin
Japanese	Elementary
	Fall 2015 - Private tutoring offered by UIC

SCHOLARSHIPS

	Scholarship of the European Union awarded for the training program "Vulcanus in Japan 2015-2016"
	Research Assistantship (RA) position (20 hours/week) with full tuition waiver plus monthly stipend, Spring 2015
	Italian scholarship for final project (thesis) at UIC, Fall 2014
	RA position (10 hours/week) with full tuition waiver plus monthly stipend, Fall 2014
	Italian scholarship for Transfer students, Summer 2014
	Italian scholarship for TOPUIC project, Summer 2014
	Scholarship for PoliTong project, Spring 2010

PERSONAL SKILLS AND COMPETENCES

VITA (continued)

Technical skills	Robotics. Good knowledge of ROS (Robotic Operative System), probabilistic robotics (Bayes filters, particle filters) and 3D simulation tools (e. g. Gazebo). Good familiarity with a wide range of sensors (e.g. sonars, encoders) and actuators. Rapid prototyping with Arduino. Good knowledge on how to use the motion sensing input device Kinect. Experience on human-computer interactions, signal processing, artificial intelligence and haptic devices
	Microelectronic Devices. Good knowledge of modern silicon based microelectronic technologies (esp. short channel MOSFETs), main models (e.g. Pao and Sah model), optoelectronic devices, nanoimprint lithography and design techniques used to represent non-idealities in MOFETs. Ability to use numerical simulation program (CAD tools) for semiconductor devices (e.g. Sentaurus)
	Digital Electronics. Good knowledge of the internal structure of DSP, FPGA, DMA controllers, memory (taxonomy, access protocols and physical models), I/O, interconnection network (transmission lines), power supply, crosstalk, EMC and timings. Experience in designing Finite State Machine (FSM), Algorithm State Machine (ASM) and Data-path. Working experience with microcontrollers, Wi-Fi modules and communication protocols (e.g. TCP/IP)
	Analog Electronics. Good knowledge of main amplifiers stages, conditioning circuit, Op-Amp, ADC and DAC (e.g. differential converters), VCO, power source regulators, radio systems architectures (e.g. ZIF and SW radio), mixers (intermodulation), PLL (design circuit and select integrated devices) and GPS. Knowledge of the working principles of the most commonly used sensors and ability to properly interface them with Arduino
	Microwaves Electronics. Good knowledge of integrated technologies for RF, microwave and mm-wave active and passive devices, S-parameters, Heterodyning, linear and nonlinear models of active microwave transistors, narrowband, wideband ad ultrawideband microwave amplifier solutions. Ability to design computer aided linear high-gain and low-noise microwave amplifiers
Social skills	Good ability to adapt to multicultural environments, gained through study and work experience abroad. Open-minded and flexible to change country
	Team spirit

VITA (continued)

	Good communication skills gained through work experiences in Italy, China and USA
	Extremely curious about different cultures, habits and languages
	Good interpersonal skills

COMPUTER SKILLS	
Electronic Device Simulation	AWR Microwave office 2013, Synopsis Sentaurus 2013 (e.g. MOSFET simulation)
Robotics	ROS (Robotic Operative System), Gazebo, scripting in Ubuntu Linux
HW Descr. Lang.	VHDL (advanced), implemented using Altera Quartus II and ModelSim on FPGAs
Programming	C (advanced), C++ (advanced), C# (average). Microsoft Visual Studio and Team Foundation Server. Experience with microcontroller programming (9S12 Freescale, Arduino with shields), Mathworks MATLAB 2013, Assembler for CPU and micro-controller (average)
Virtualization software	VMware Workstation 10
CAD	Maxon Cinema 4D R16 (advanced), Maya 2014 (average), AutoCAD 2012 (average)
Writing & designing	Suite Microsoft Office, OpenOffice, Adobe Acrobat XI, Adobe InDesign CC
Image processing	Adobe Photoshop CC, Adobe Photoshop Lightroom 5, Adobe Illustrator CC
Video processing	Adobe After Effects CC (and VideoCopilot plugins), Adobe Premiere CC
Audio processing	Adobe Audition CC, Cubase 5, Native Instruments Komplete 8