DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Design, Implementation and Evaluation of a Software Solution for Localization, Navigation and Mission Management Applied to Autonomous Mobile Robots

Monografia submetida à Universidade Federal de Santa Catarina

como requisito para a aprovação da disciplina:

DAS 5511: Projeto de Fim de Curso

Clovis Peruchi Scotti

Florianópolis, Março de 2010

Design, Implementation and Evaluation of a Software Solution for Localization, Navigation and Mission Management Applied to Autonomous Mobile Robots

Clovis Peruchi Scotti

Esta monografia foi julgada no contexto da disciplina
DAS 5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação Industrial

Banca Examinadora:

Sauro Longhi
Orientador Empresa

Endson Roberto de Pieri
Orientador do Curso

Prof. Augusto Humberto Bruciapaglia
Responsável pela disciplina

Prof. Leandro Buss Becker, Avaliador

Leonardo de Campos Taschetto, Debatedor

Daniel de Macedo Possamai, Debatedor

Acknowledgements

This thesis and related project were held at Università Politecnica delle Marche, Italy. I would like to thank all my coworkers at the DIIGA department of UNIVPM for providing such an open and motivated work environment and for coping with the recurrent hallway transit restrictions due to incessant robot testing; Gianluca di Buò, Andrea Cesetti and Mirco Babini for the huge amount of insightful brainstorming sessions on robotics and general technology; my supervisor Sauro Longhi for giving me the opportunity to work in a real, out-of-research, project and at the same time to provide invaluable research freedom.

My sincere thanks also go to the staff at the *spin off* IDEA for being somewhat my temporary family in Ancona and to my good friends that stayed in Italy struggling with Italian food and the cold weather that makes people work instead of sweat. You all helped me maintaining my mind sane by providing me unmeasurable amounts of fun.

Finally, I would like to thank my whole family and close friends for being the ultimate development framework and toolkit for life.

He who makes a beast of himself gets rid of the pain of being a man.

Dr. Johnson

Abstract

The employment of autonomous mobile robots in industrial environments is considered to be a great promise to the automation of these environments and thus to the separation of the human worker from manual, highly repetitive or dangerous operations. The use of these robots is limited due to its usual high costs and lack of flexibility.

This work provides a broad description of the robotics-specific software layer for a flexible and inexpensive autonomous system for use in different sectors of industry. This system includes an extended Kalman Filter localization system that handles features of different natures concurrently, a stable pose controller for navigation and a complete coordination and mission management system, based on a tasks queue, that provides a *web service* interface to the robot. This interface is XML based and is designed to ease its integration with third party systems. Furthermore, the developed system is deployed into a real application, its performance is evaluated and some results are presented.

Resumo Estendido

O uso de Robôs Móveis Autônomos em ambientes industriais e laboratoriais promete dar um gande impulso à automação de certos processos e assim afastar ainda mais o operador humano de tarefas braçais indesejáveis ou perigosas. Tal tecnologia ainda é pouco empregada devido à alta especificidade, ao preço proibitivo das soluções disponíveis e às barreiras tecnologicas para o uso prático de robôs móveis. Em particular, o planejamento de tarefas, a tecnologia de localização e a geração de trajetórias têm se mostrado bastante restritivos à utilização destes em ambientes industriais. Este trabalho oferece uma ampla descrição da camada de software específica à robótica móvel de um sistema industrial autônomo flexível e barato para emprego em diferentes setores da industria.

O software implementado inclui: um sistema completo de localização que utiliza um filtro de Kalman estendido, um sistema de controle de posição usado para navegação e um sistema completo de coordenação de missão baseado em uma fila de tarefas.

Ao longo deste trabalho é apresentada uma extensa descrição dos sistemas de localização e de controle de posição adotados, é proposta uma nova arquitetura orientada a serviços para sistemas dessa natureza com o objetivo de aumentar a flexibilidade e performance do sistema como um todo e posteriormente uma descrição mais orientada a implementação é apresentada para os serviços implementados juntamente com uma alternativa para a solução do problema de tempo-real. Após esta apresentação do sistema, é realizada uma avaliação do mesmo em funcionamento em um laboratório de controle de qualidade de máquinas de lavar roupa. Nestes laboratórios, uma grande quantidade de máquinas de lavar roupa são mantidas em contínuo funcionamento para fins de monitoramento. O robô móvel, por sua vez, é equipado com os equipamentos de medição (e.g. vibrômetro, microfone) e deve visitar periodicamente as máquinas afim de adquirir dados sobre o funcionamento de cada uma delas.

Em suma, as contribuições relevantes deste trabalho são:

- Estudo e descrição da implementação de um sistema de localização baseado em um filtro de Kalman estendido com o uso de características (*features*) de

diferentes naturezas (e.g. linhas, pontos) concorrentemente (seção 2.4.4.3).

- Apresentação de uma arquitetura orientada a serviços (SOA) específica para sistemas desta natureza desenvolvida com o intuito de aprimorar os sistemas de localização, de extração de características e de controle de posição (capítulo 4).
- Uma solução para o problema de tempo-real decorrente da implementação de um sistema de localização robusto ao movimento aleatório do robô (seção 5.6).
- Um breve estudo de caso do uso de tecnologias modernas de tecnologia da informação (e.g. SOA, interface XML, *web services*) na implementação de um robô móvel.

Contents

Tables of Symbols

UNIVPM	Università Politecnica delle Marche
AGV	Autonomous Guided Vehicle
RGB	Red, Green and Blue; Common way to represent images
LRF	Laser Range Finder
WM	Washing Machine
SLAM	Simultaneous Localization and Mapping
GPS	Global Positioning System
MHT	Multi-hypothesis tracking
PF	Particle Filter
EKF	Extended Kalman Filter
SOA	Service Oriented Architecture
MRDS	Microsoft Robotics Developers Studio
CCR	Concurrency and Coordination Runtime
DSS	Decentralized Software Services
DIIGA	Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione
GUI	Graphical User Interface
FIFO	First In First Out
HTTP	Hypertext Transfer Protocol

List of Figures

Chapter 1: Introduction

Mobile robots exhibits a huge potential for industrial and domestic applications but until now not much of this systems have found their way out of research. This work focuses on producing a solution ready for industrial or domestic deployment. Developing such solution involves solving many technological problems taking care with maintaining the outcome practical, cheap and robust.

The current robotics industry has similarities to the early computer industry in many senses; robots (mobile or not) are, like early computers, widely employed in high-end industrial environments but they are expensive and lack flexibility. While computers became extremely popular, flexible and cheap, robots continue in that early stage being expensive and lacking flexibility.

Autonomous Guided Vehicles (AGVs) are specialized mobile robots that operate mainly as autonomous transport systems inside modified environments. AGVs comprises a large and lucrative industry and are widely used throughout distinct industry sectors, from automotive assembly lines to hospitals. The main limitation of the *current* AGV technology is its lack of flexibility and standardization; one company that desires to deploy AGV technology to its operations must be prepared to completely adapt their environment to the use of such technology. This is so expressive that fitting AGVs into a legacy production environment is uncommon, the majority of applications of AGVs considers the use of such systems since the design of the production environment so that it is compatible with the particular adopted AGV technology. This factors impose notable limits to the adoption of AGVs in different industry sectors. Developing more flexible AGV technology can certainly broaden the market for this systems.

It is believed that once mobile robots become flexible to their environment and assigned task, its adoption will increase sharply. Domestic robots for example, are a highly desired household item but until now only a small group of products explores such market. This work focuses in developing a flexible software system for mobile robots that enables the robot to perform generic tasks (e.g. normal AGV operations) in a more flexible or dynamic environment.

The challenges in developing such system are numerous and very interdisciplinary. Locomotion problems are solved with understanding of mechanisms, kinematics, dynamics and control theory. Perceptual systems must certainly employ the use

of signal processing techniques together with specific knowledge on a certain sensor technology (e.g. computer vision, laser scanners). Navigation and Localization capabilities are conceived with the use of computer science, information theory, artificial intelligence etc. Finally, the integration of that system in a production environment will eventually employ modern information technology and more computer science. In fact it is easy to picture even more intricate scenarios with all these knowledge areas interlacing among themselves in between different parts of the system.

Academic research in this area is very wide and many of the independent solutions used on this work are already well established and well understood; this work stands as a case study of the integration of many different technologies into an industrial system. In order to establish a well correlated and grounded work, care is taken to refer to the available literature at its related document sections.

As part of a partnership between *Università Politecnica delle Marche* (UNIVPM) and a private enterprise, the developed solution was deployed as an automated system for Washing Machine inspection in a reliability test laboratory. These laboratories maintain a large number of washing machines fully working for very long periods for many quality control reasons (See Figure 1.1). Common operations carried in such laboratories by human operators include periodically activate some controls in the washing machines' control panel and to carry heavy measurement equipment (e.g. microphones, vibrometers) to each of the test subjects. To automate this application, the robot must navigate through narrow corridors in between rows of washing machines on its way to a specific washing machine and then approach it to take some specific measurements or perform some specific operation. This deployment was very useful to fully evaluate and test the solution to its full extent and receive feedback from third parties.

Figure 1.1: Washing Machine Reliability Test Laboratory

This work was carried in the *Università Politecnica delle Marche* (UNIVPM), Ancona - Italy, from September of 2009 to March of 2010.

1.1: Objectives

The main goal of this work is to establish a software solution for mobile robotics in a ready to market stage. The developed solution must enable a buying costumer to deploy a robot in an already existing environment with very little adaptation. The task management must be simple and should exhibit a simple interface for integration with third party systems (e.g. supervisory systems). Some tasks can be application specific but they should be easily constructed on top of the, already implemented, navigation and feature detection & recognition tasks.

In order to achieve the aforementioned goal, a localization system must work transparently and concurrently with application specific software. Most final users are

not even aware of the depth of the localization problem so solving it stands just as a mean to achieve the desired performance. Navigation and pose control are also very important and in most applications it will be what an hypothetical costumer is explicitly looking for. Additionally a flexible and accessible high-level interface to the system should be provided.

This document aims at describing the relevant parts of such a system with care on bridging state-of-art theoretical knowledge in mobile robotics to their implementation in a final solution.

1.2: Outline

This work is divided into chapters that should provide the logical sequence of a bottom-up design for such system.

Chapter 2 focuses on the problem of Localization and its solutions. In this chapter, a brief discussion on the importance of localization, a review of the most popular localization algorithms and the extensive description of our choice, Extended Kalman Filter, are presented.

Chapter 3 describes the problem of Pose Control together with some details of our implementation.

Chapter 4 describes conceptually the services oriented architecture used in our system. The used architecture is apparently novel in some ways and its design was intimately driven by the design of the localization system.

Chapter 5 gives details regarding the implementation of the general purpose systems as services.

Chapter 6 describes the application in which the system was deployed, the application-specific service added and a brief performance analysis focused on this application.

Chapter 7 provides a discussion about the overall performance of the system during lab tests and in the deployed application.

Chapter 2: Mobile Robots Localization

2.1: Introduction

In mobile robotics, localization is the problem of continuously finding a robot's pose (which comprises its position and orientation) with satisfactory accuracy and precision. The pose of a moving object can be fully described by it's state regarding to each of its degrees of freedom (DOF). In the general case, an object's pose can be expressed by:

$$\xi_{6DOF} = \begin{bmatrix} x \\ y \\ z \\ \psi \\ \theta \\ \phi \end{bmatrix} \tag{2.1}$$

Adding constraints to the system decreases the number of DOF and thus lowers the dimensionality of an object's pose. A differential-drive robot operating on a flat ground, in particular, has only three degrees of freedom due to the constraints implied by the operation in such conditions. Therefore, in our context, determining the robot pose consists in estimating:

$$\xi_{3DOF} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.2}$$

For the unfamiliar reader, this problem may seem trivial since in most common autonomous systems information regarding the system's state can commonly be attained by the use of the correct set of sensors. In the particular case of a mobile robot moving freely, a definitive solution does not exist, in the sense that there is no universal sensor that working alone measures one's pose in relation to some known reference frame.

2.2: Taxonomy of Localization Problems

There are many variations of this problem depending of the application requisites, the available sensors, environment entropy, environment behavior and so on. This section describes the differences between many situations where localization systems are employed with regard to the most relevant factors.

2.2.1: On confidence and availability of previous knowledge

The confidence on the previous knowledge sustained by the robot regarding its own pose changes widely between possible applications; in short there are three possible situations shown here following an increasing degree of difficulty.

Pose Tracking It is assumed that the robot is given its initial pose, or an approximation of it, and should then maintain, or even enhance, the accuracy and certainty of that pose belief throughout its operation in that given environment. The localization algorithm can be confident that the robot is not arbitrarily *teleported* to another location by any other external system and thus the robot pose only changes due to its own, modestly noisy, actions.

Pose Retrieving This is an extension of *pose tracking* variant with the slight difference that there is no previous knowledge regarding the robot's pose. In this case the robot should initially retrieve its pose out of complete uncertainty about it. After a confident pose belief is obtained, this problem becomes that of *pose tracking*. In this variant, like in *pose tracking*, it is assumed that the robot's pose only changes due to its own actions.

Pose Recovering – Kidnapped Robot Problem This is the less restringing of the variants and can be seen as an extension of the *pose retrieving* one. In this case, no total confidence on previous knowledge is ever achieved; this assumption enables the robot to recover from catastrophic localization errors or even a *kidnapping* situation, thus the common *Kidnapped Robot Problem* name, where the robot is arbitrarily moved to another, unknown, position by a third party.

2.2.2: On Environment Behavior and Entropy

The environment in which the robot performs its operations obviously plays a major role on the characterization of the problem to be solved; its properties can come in aid to the localization system (e.g. structured walls and floor, known landmarks, known geometry) lowering its entropy or they can impose problems like freely moving objects and general unpredictability. This deep correlation between environment nature (e.g. predictability) and localization system performance stems from the inherent necessity, from the robot, to acquire some kind of spatial information that correlates the robot's pose and some feature of the environment.

Static Environment In some applications the environment where the robot operates can be maintained strictly static in order to aid in the localization operations. In this situation, the only continuously changing variables are with regard to the robot itself. Additionally, some previous knowledge of that static environment can be provided to the robot in order to leverage the localization system. For example, since the environment is static, the fixed position of some key landmarks can be provided to the robot to aid localization and to *lock* a global reference frame to the one used by the robot.

Dynamic Environment More generally, environments posses moving objects other than the robot itself. This can be the result of a plethora of events including opening doors, passing people and even other mobile robots. The growing levels of dynamism are treated by localization systems in different ways; low dynamism can simply be treated as sensor noise, intermediary and sparse dynamism can be handled heuristically by credibility-tracking mapping and finally high dynamism can be handled by adding the dynamic entities to the state vector resulting in more modeling and computational complexity.

2.2.3: On Robot Behavior

Active Localization In some applications, the robot behavior can come to aid the localization system. For example, the robot movement and behavior can be used to help it localizing itself. This *active localization* approach adds the significant problem of figuring out which behavior (e.g. which movements) would effectively help. This pattern is usually employed on systems of *pose retrieving* or *pose recovering* when searching

for a reasonable pose belief.

Passive Localization *Passive Localization* assumes that the decisions about robot movement are not accessible and can behave in a random manner. This is probably the most common situation since different systems will drive the robot according to its assigned task.

2.3: Deterministic Techniques

2.3.1: Trilateration

The simplest solution to localization problems is the use of a deterministic global positioning system. Such systems generally uses trilateration of a group of simultaneous distance measurements to known points in a global reference frame to compute the deterministic position of the measurement device. To estimate the orientation additional information can be used (e.g. compass) or it can be inferred indirectly by, for example, the current motion vector direction together with some knowledge on the motion constraints of the moving body. Following, a brief description of some of this systems is presented.

2.3.1.1: Global Position System – GPS

The GPS system, which is maintained by the government of the United States of America, is generally provided in a *black box* device that provides detailed information on its global position, velocity, altitude etc. This device works by inferring distance measurements d_i for at least three different satellites orbiting the earth[1]; each of this distance measurement implies that the receiver's position lays on the surface of an sphere of radius d_i. Computing the intersection of such spheres gives the *exact* position of the receiver. Mobile robots can take advantage of such signal as it's position source and solve part of the problem of localization easily. Unfortunately the use of this alternative is very limited for a number of reasons: the system relies on external low power radio signal sources and thus the necessity for constant and concurrent availability of many signals from different satellites makes its use in indoor or underground

[1] In order to compute the distance measurements to three satellites, the receiver needs the signal from at least four satellites. This is due to the fact that the receiver does not have an atomic clock to infer the correct time.

environments difficult, the accuracy of this system is dependent on, generally unpredictable, weather conditions since different atmospheric conditions produce different noise in the final distance readings and finally its precision is very far from sufficient for industrial applications[2].

2.3.1.2: Beacon Positioning System

Another alternative is to replicate an analogue of a GPS system locally. Many commercial systems use sets of optical, ultrasound or radio frequency emitters in the target environment in order to produce a local positioning system without the availability drawback inherent of the earth-wide systems. This solution is generally very satisfactory since it provides high precision and accuracy localization. The most expressive drawbacks of such technology are the lack of flexibility imposed by necessary environment changes (addition of these emitters) and high cost.

Furthermore, even in applications in which such approach could be employed, the use of more sophisticated ones greatly increases the performance of the system.

2.4: Kalman Filter Based

One of the most popular and studied techniques for mobile robots localization employs one specific type of State Observer, the Extended Kalman Filter (EKF). Following we present briefly the logical progression from state observers to the EKF and its implementation.

2.4.1: State Observers

The localization problem characterizes the common situation where state estimators are used: the estimation of a state vector of which no direct measurement alternative is available. In this case, the state being the robot's pose. In Control Theory, state observers use a mathematical model together with the controls applied to it and its available measurements to estimate the desired, internal, state variables of that system.

Figure 2.1 gives a fast idea on the role of a state observer. Note that the state

[2]A normal GPS with very good signal reception has an average error of a couple of meters which is far more than the acceptable for industrial indoor applications

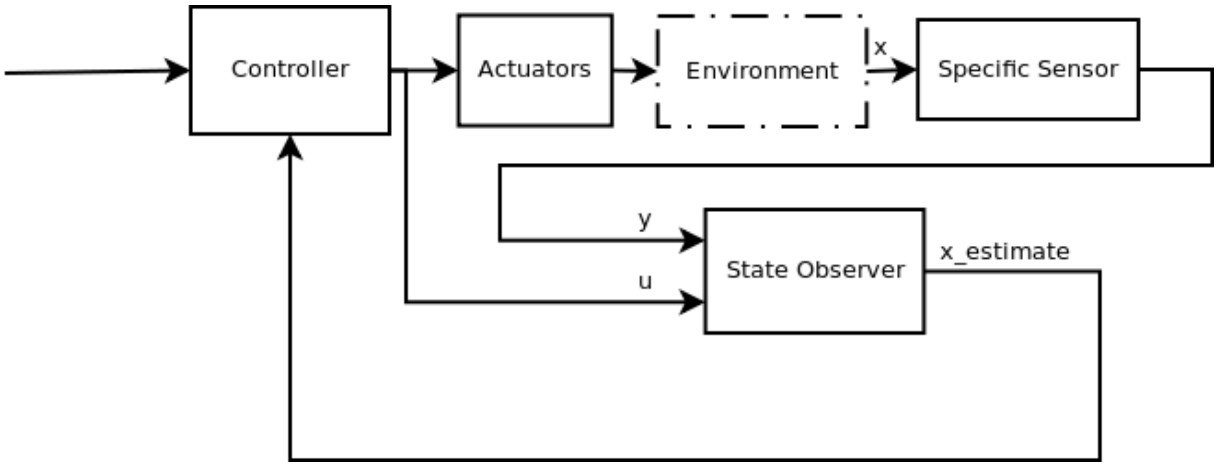


Figure 2.1: Basic State Observer Setup

vector x, although shown in the diagram, is not accessible and thus the controller uses $x_{estimate}$ (x_f) as its feedback source.

In other words, given a multivariate liner, time invariant, discrete-time system on standard representation:

$$x_{k+1} = Ax_k + B_u u_k \tag{2.3}$$

$$y_k = Cx_k \tag{2.4}$$

where x_k can't be directly measured due to:

$$rank(C) < dim(X)$$

the state observer is a dynamic system that produces an estimate x_f:

$$x_{f,k+1} = f(x_{f,k}, u_k, y_k) \tag{2.5}$$

being u_k the current input to the system, y_k the current available output and $x_{f,k}$ the last estimate.

The state observer is in itself a multivariate system and its dynamic is governed by the matrix:

$$A_{obs} = A - KC \tag{2.6}$$

therefore, the project of the state observer consists in finding K such that the eigenvalues of A_{obs} lay inside the stability circle:

$$abs(\lambda) < 1 \tag{2.7}$$

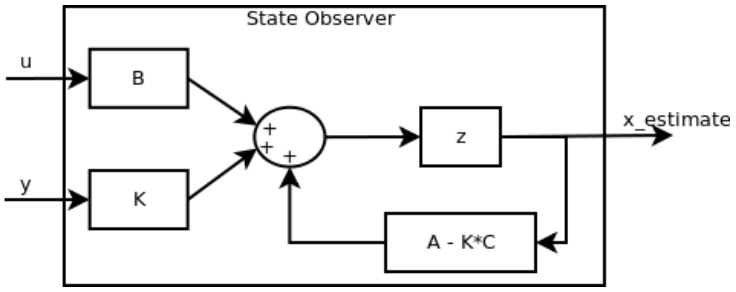


Figure 2.2: State Observer Internal Structure

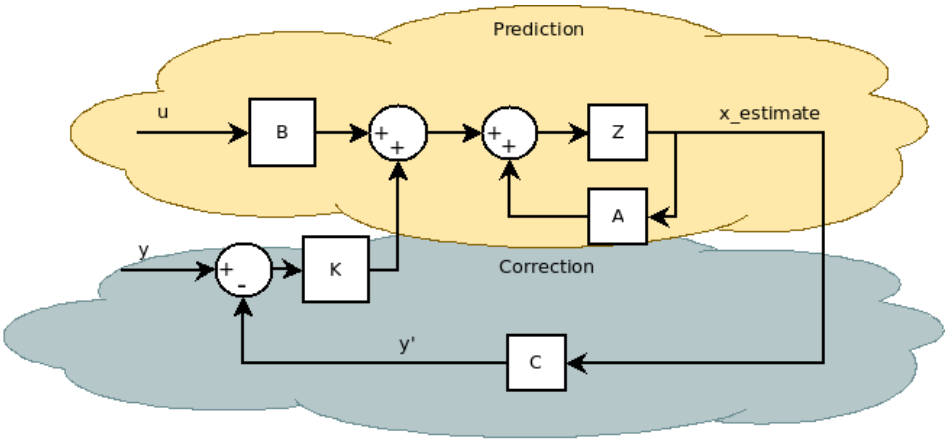


Figure 2.3: Predict and Update functional parts separated.

If designed properly (i.e. the internal model is sufficiently accurate), the output estimate x_f should converge to the real state vector given enough time steps k:

$$\lim_{k \to \infty} (x_{f,k} - x_k) = 0 \tag{2.8}$$

The dynamic of the estimation error $e_k = x_k - x_{f,k}$ does not depend on the system's input or output, depending only on the error present at the initial condition e_0. If the state observer complies with this two requisites, it is known as a Luenberger Observer. Figure 2.2 is a common way to depict the internal components of the observer.

Figure 2.3 shows another representation of the internal structure of a state observer. This representation can easily be achieved by some block manipulations to the diagram from Figure 2.2 and is used to show that state observers consists of a *predictor* part, which simulates the process evolution from its inputs, and a *corrector*[3] part that uses the difference between the system's output and the output from the predicted

[3]A big portion of the literature uses the term Update instead of Correction. Throughout this work, both terms are used refering to the same part or step.

model to *Update* the state estimate.

State observers theory deviates from the context of localization and therefore is not presented here. More information on this topic can be found on [14].

2.4.2: Kalman Filter

A Kalman Filter, initially proposed in 1960 [15], is a special state observer in which the mean of the squared error for the observations of a stochastic system is minimized. That is achieved by using information regarding the uncertainty of each signal when designing the state observer gain K, denominated kalman gain. Given that the system can be represented by:

$$x_{k+1} = Ax_k + B_u u_k + w_k \tag{2.9}$$

$$y_k = Cx_k + v_k \tag{2.10}$$

with $w(k)$ being the process noise (or control noise) and $v(k)$ being the measurement noise and both being independent, zero mean, white noise, unimodal gaussian distributions:

$$p(w) \sim N(0, R)$$
$$p(v) \sim N(0, Q)$$

with R and Q being the covariances of those distributions, at a time-step k, the filter gives an unimodal belief estimate in the form of a mean vector $\mu(k)$ and its covariance matrix $\Sigma(k)$.

In the *prediction* part of the filter, along with the normal *prediction* of state transition computed from the input $u(k)$, Σ is updated, essentially increased, due to the process uncertainty:

$$\bar{x}_k = Ax_{k-1} + B_u u_k \tag{2.11}$$

$$\bar{\Sigma}_k = A\Sigma_{k-1}A^T + R \tag{2.12}$$

The *correction* part uses a *linear* relation between the difference $y_k - C\bar{x}_k$ and the state estimate to correct the latter:

$$x_k = \bar{x}_k + K_k(y_k - C\bar{x}_k) \tag{2.13}$$

$$\Sigma_k = (I - KC)\bar{\Sigma}_k \tag{2.14}$$

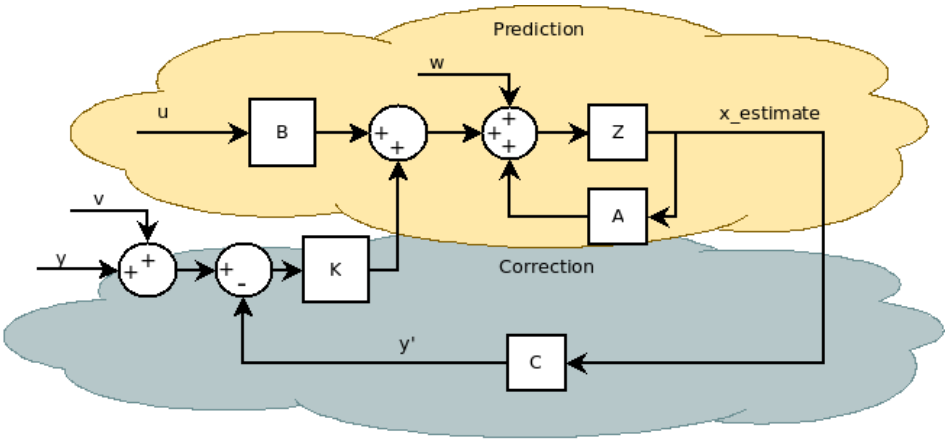


Figure 2.4: Predictor and Corrector parts of the Kalman Filter with modeled noise inputs

Differently from normal state observers, the observer gain K_k is weighted *accordingly* to the current ratio of uncertainty between the predicted signal and the measured one:

$$K_k = \bar{\Sigma}_k C^T (C\bar{\Sigma}_k C^T + Q)^{-1} \tag{2.15}$$

Note that in this discrete form, the state estimate $\bar{x}_k$ and the covariance matrix $\bar{\Sigma}_k$ used on the *correction* part *can* be the ones generated by the *prediction*. This, sequential approach gives the notion that, in this discrete implementation, one filter iteration can be divided into two *steps*:

- *Prediction Step* – Uses the control input during the last time-step to generate updated state estimate $\bar{x}_k$ and increases covariance $\bar{\Sigma}_k$.
- *Correction Step* – Uses the difference between measurement y_k and prediction $C\bar{x}_k$ to generate a correction signal and decreases the covariance Σ_k.

This separation is very popular in robotics applications and will be extensively used in the following sections.

The covariances R and Q are very important filter design parameters and should reflect the expected behavior of the process and the used measurement devices. In applications where the noise present is accurately described by *multivariate normal distributions*, good estimates can be inferred from extensive experimentation or from intricate process knowledge; in other applications, the filter can still be used given

some *quantitative* reasoning compatible with the specific application. In this latter applications, the optimal performance of the filter is obviously degraded but sometimes good results can still be achieved.

Another possible approach is to estimate this matrices *on the fly*; for example, in some applications where $A = I$ (discrete system with no dynamic) it is reasonable to assume that for $u(k) = 0$ the overall uncertainty remains the same, $R \to 0$, and analogously a proportionality relation can be used:

$$R_k \propto u_k \tag{2.16}$$

Also, it may be useful to observe the one-dimension equivalent of Equation 2.15 with $C = 1$, Equation 2.17 where the ratio of covariances becomes obvious:

$$K_{k,1D} = \frac{\bar{\sigma}_k^2}{\bar{\sigma}_k^2 + \sigma_Q^2} \tag{2.17}$$

with σ_Q^2 being the variance of the measurement and $\bar{\sigma}_k^2$ the variance of the state variable. An illustration of the **1D** case is shown in Figure 2.5.

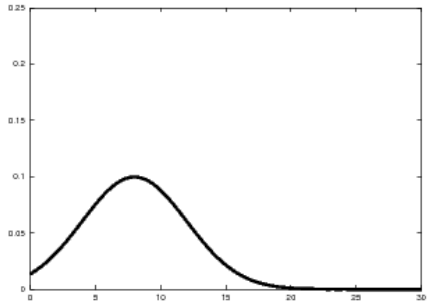
The main reason why the Kalman Filter alone is not suitable for mobile robots localization is its assumptions towards the *linearity* of the control (B_u) and measurement (C) models which are essentially *nonlinear* in that application. To illustrate this limitation, picture a robot moving with constant tangential and angular velocities, $u = \begin{bmatrix} \nu_i & \omega_i \end{bmatrix}^T$, which produces a circular trajectory and the state vector being the common representation for pose, $\xi = \begin{bmatrix} X & Y & \theta \end{bmatrix}^T$; the resulting real pose ξ will indefinitely lay in a limited circumference showing that a linear B_u is not suitable.

2.4.3: Extended Kalman Filter

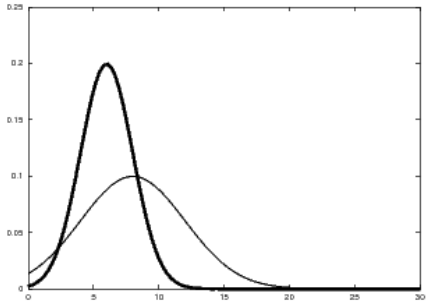
The EKF was designed to address the *linearity* limitation of the Kalman Filter. Being both the state transition, g and the measurement, h, *nonlinear* functions such that:

$$x_{k+1} = g(u_k, x_k) + w_k \tag{2.18}$$

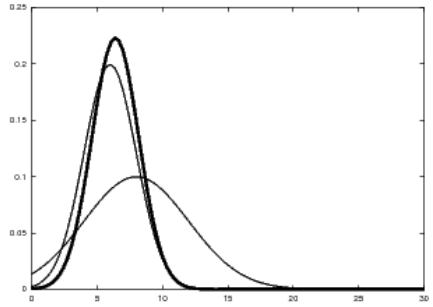
$$y_k = h(x_k) + v_k \tag{2.19}$$



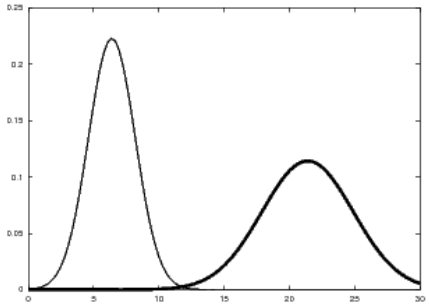
(a) Uncertain belief



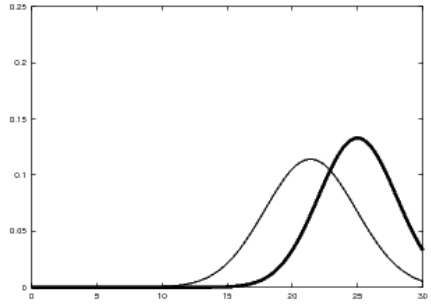
(b) First measurement



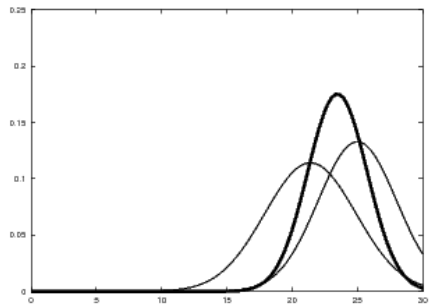
(c) Correction produces a new belief with smaller variance



(d) Prediction produces a new belief with larger variance



(e) New measurement



(f) Second Correction and newer belief

Figure 2.5: Kalman Filter belief through time. New data in every step is presented in bold

with g replacing the A and B_u linear transformations from Equation 2.3 and h replacing C from Equation 2.4, the extended Kalman Filter uses a *first order Taylor expansion* to linearize those functions around the current belief at each iteration of the filter. This partial *linearizations* produces temporary *jacobians* equivalents to A,B_u and C for every filter iteration which are then known as G, V and H respectively, which are the evaluations of:

$$G_k = \left.\frac{\partial g(u,\mu)}{\partial \mu}\right|_{u=u_k,\mu=x_k} \tag{2.20}$$

$$V_k = \left.\frac{\partial g(u,\mu)}{\partial u}\right|_{u=u_k,\mu=x_k} \tag{2.21}$$

$$H_k = \left.\frac{\partial h(\mu)}{\partial y}\right|_{\mu=x_k} \tag{2.22}$$

at the present belief. Substituting the $g(u,\mu)$ on Equation 2.11 and G and V on Equation 2.12 gives the *prediction* step:

$$\bar{x}_k = g(x_{k-1}, u_k) \tag{2.23}$$

$$\bar{\Sigma}_k = G\Sigma_{k-1}G^T + VMV^T \tag{2.24}$$

Note that on equation 2.24, instead of R (from equation 2.12), M is used and V proportionally maps the uncertainty in M to Σ.

Substituting H on Equations 2.15 and 2.14, the function $h(\mu)$ on Equation 2.13 and 2.14 gives the *correction* step:

$$K_k = \bar{\Sigma}_k H^T (H\bar{\Sigma}_k H^T + Q)^{-1} \tag{2.25}$$

$$x_k = \bar{x}_k + K_k(y_k - h(\bar{x}_k)) \tag{2.26}$$

$$\Sigma_k = (I - KH)\bar{\Sigma}_k \tag{2.27}$$

2.4.4: EKF applied to Mobile Robot's Localization

2.4.4.1: Problem Definition

Given a *differential-drive* wheeled mobile robot with encoders in both wheels and a with an virtual sensor (see section 2.4.4.3) that provides measurements in polar coordinates to *reflector beacon features* and measurements, also in polar coordinates, to *line features*:

$$u_{encoder} = \begin{bmatrix} \Delta s_r \\ \Delta s_l \end{bmatrix} \tag{2.28}$$

$$y_{beacons} = \begin{bmatrix} \rho \\ \alpha \end{bmatrix} \tag{2.29}$$

$$y_{lines} = \begin{bmatrix} r \\ \beta \end{bmatrix} \tag{2.30}$$

with polar coordinates for $y_{beacons}$ and y_{lines} in the robot reference frame, $u_{encoder}$ expressed in terms of the incremental traveled distance of the right, Δs_r and left, Δs_l wheels and given a *map* that contains a list of *beacon features*:

$$m_{beacons} = \begin{bmatrix} m_x \\ m_y \end{bmatrix} \tag{2.31}$$

with Cartesian coordinates in the global reference frame, and with *line features*:

$$m_{lines} = \begin{bmatrix} m_r \\ m_{beta} \end{bmatrix} \tag{2.32}$$

also with polar coordinates in the global reference frame. Finally, assuming that such robot must **Passively** perform **Pose Tracking** in a **Static Environment**, a complete EKF localization system should include the following:

1. Prediction Step
2. Feature Extraction
3. Feature Matching
4. Correction Step

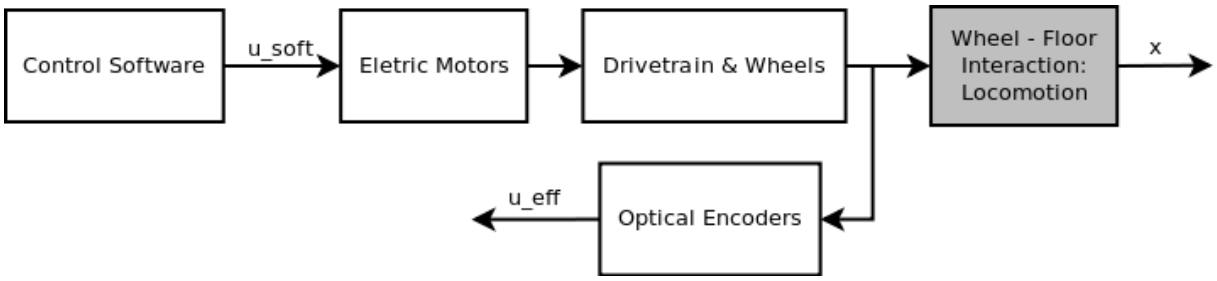


Figure 2.6: Odometers role as control signal measurement

This parts are detailed in the following sections.

2.4.4.2: Prediction Step

The *prediction step* is responsible for predicting the robot position from information regarding the applied control signals. Although not very intuitive, the encoders' readings, $u_{encoder}$ are used as the control vector since it effectively represents the control applied by each wheel on the floor. Figure 2.6 shows the role of the odometers on the localization control system; notice that the gray block is in fact the process which output localization aims at estimating. By using this approach, the electromechanical part of the actuator system and its noise/uncertainty are bypassed.

The incremental travel distances (i.e. the state transition function, Equation 2.23) from encoder signals to state transition, for this kind of robot, is:

$$g(u,\mu) = g(\Delta s_r, \Delta s_l, x, y, \theta) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin(\theta + \frac{\Delta s_r - \Delta s_l}{2b}) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix} \tag{2.33}$$

where b is the distance between the two wheels of the differential-drive robot. The *jacobians* G_k and V_k (Equations 2.20,2.21) of $g(u,\mu)$ are:

$$G_k = \begin{bmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s \sin(\theta + \Delta\theta/2) \\ 0 & 1 & \Delta s \cos(\theta + \Delta\theta/2) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

$$
\begin{aligned}
V_k &= \begin{bmatrix} \frac{\partial g}{\partial \Delta s_r} & \frac{\partial g}{\partial \Delta s_l} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{2}\cos(\theta + \frac{\Delta\theta}{2}) - \frac{\Delta s}{2b}\sin(\theta + \frac{\Delta\theta}{2}) & \frac{1}{2}\cos(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2b}\sin(\theta + \frac{\Delta\theta}{2}) \\ \frac{1}{2}\sin(\theta + \frac{\Delta\theta}{2}) + \frac{\Delta s}{2b}\cos(\theta + \frac{\Delta\theta}{2}) & \frac{1}{2}\sin(\theta + \frac{\Delta\theta}{2}) - \frac{\Delta s}{2b}\cos(\theta + \frac{\Delta\theta}{2}) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix}
\end{aligned} \tag{2.35}
$$

with:

$$
\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{2.36}
$$

$$
\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{2.37}
$$

An error model M_k for the *action* of each wheel is:

$$
\begin{aligned}
M_k &= \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_l^2 \end{bmatrix} \\
&= \begin{bmatrix} k_r|\Delta s_r| & 0 \\ 0 & k_l|\Delta s_l| \end{bmatrix}
\end{aligned} \tag{2.38}
$$

where k_r and k_l are tuned to model uncertainty differences between the two wheels. Applying the above function $g(u, \mu)$, the *jacobians* and the error model on Equations 2.23 and 2.24 results in the *prediction step*.

Figure 2.7 illustrates the integration of uncertainty produced by the *prediction step* during: a straight drive, a parabolic turn and another straight drive. The ellipses are drawn from the first two eigenvectors of Σ depicting only the uncertainty on x and y; uncertainty in θ can indirectly be observed on the uncertainty increase on the direction perpendicular to that of the movement. In this figure the ellipses dimensions, but not the rotation, are greatly exaggerated to ease visualization.

2.4.4.3: Feature Extraction

In this context, feature extraction consists in extracting useful information from large, noisy and high dimensional datasets. This operation is application specific, sometimes unecessary or can be embedded in the sensor itself. One of the most popular sensors used in mobile robotics, and used in this work, is the *Laser Range Finder – LRF*, this sensor provides a raw array of distance measurements forming a planar distance profile, generally covering angles near $\gamma_{range} \approx \pi$. Figure 2.8 shows

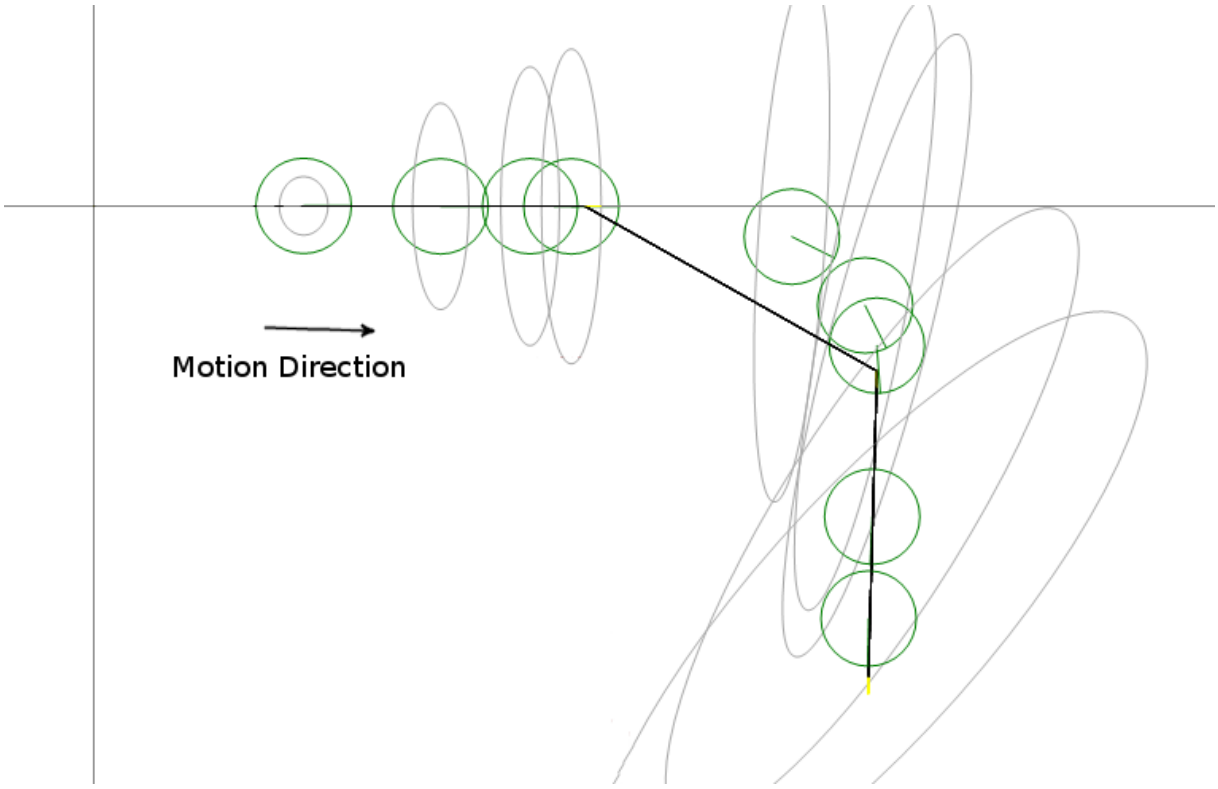


Figure 2.7: Illustration of successive prediction steps without any correction steps showing the integration of uncertainty represented by the gray ellipses

graphically the output of that sensor, note that although apparently connected, the output is just a discrete set of readings.

This type of sensor produces large amounts of data and using its raw readings without any preprocessing is highly inefficient. Together with LRFs in this group are, among others, video cameras and stereo cameras. For example, when using a camera image, each pixel on the image is in itself an independent dimension or in a *RGB* color image, three dimensions (one for each color channel). This situation is more noticeable when using vision cameras, Equations 2.39 and 2.41 give an idea of the dimensionality of the output of those sensors; a typical color image can have:

$$n_{dim,CAM} = width \times height \times n_{channels} \tag{2.39}$$

$$\begin{aligned} n_{dim,CAM} &= 640 \times 480 \times 3 \\ &= 921600 \end{aligned} \tag{2.40}$$

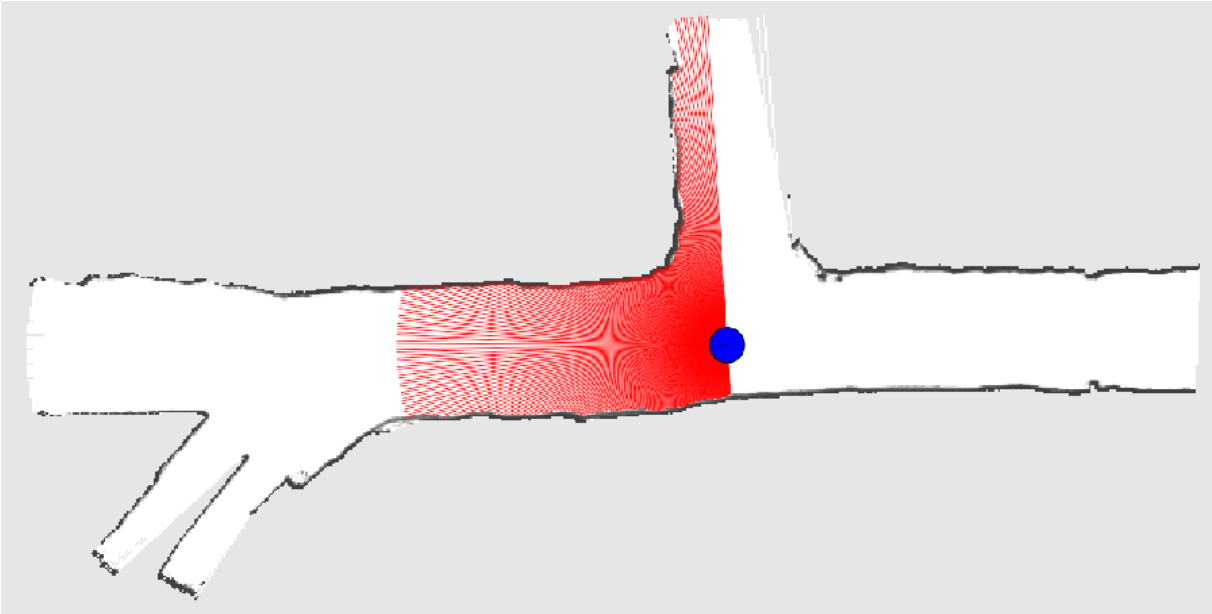


Figure 2.8: Raw Laser Range Finder output in red from [27].

dimensions. In the case of the LRF, that could be:

$$n_{dim,LRF} = \frac{range}{resolution} \tag{2.41}$$

$$\begin{aligned} n_{dim,LRF} &= \frac{180}{0.5} \\ &= 360 \end{aligned} \tag{2.42}$$

This *problem*, can be solved using intermediate computer algorithms that run through this large datasets (e.g. image, echoes array) *extracting* smaller sets of *features* with higher level of abstraction (i.e. with less dimensions). This algorithms are usually encapsulated in a independent subsystem, in hardware or in different software modules so. This separation is attractive because it gives the idea that the feature extraction's output is in fact an *intelligent* sensor output.

Many common vision application aim at extracting the position $\begin{bmatrix} x & y \end{bmatrix}^T$ of a visible object in the image; in this application the *feature extraction* algorithm produces a bi-dimensional output from a high-dimensionality input dataset. LRF data is commonly used to extract *line features* expressed by Equation 2.30 or *reflector beacon feature*, expressed by Equation 2.29. Figure 2.9 shows the *role* of this algorithms. This approach greatly simplifies the rest of the system and in some cases greatly attenuates common noise. For example, a *line feature* extraction algorithm aims at finding the best line that fits a certain, sometimes large, group of points and from information theory it

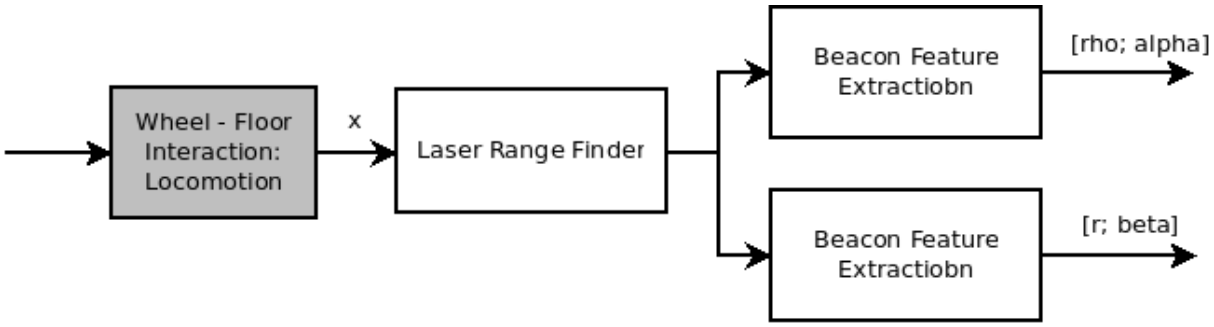


Figure 2.9: Disposition of the LRF and the feature extraction routines.

is reasonable to expect that the accuracy of the extracted line's parameters is better than the the one from the raw points data, thus attenuating noise. This stands true if the present noise is zero-mean and those effects should be more noticeable with the growth in the number of points.

2.4.4.4: Feature Matching

Given a multi-feature environment, feature extraction algorithms will produce a list of features for each sensor measurement. This features alone are of no use for localization; a matching feature from the map, whose position in the global reference frame is known, is mandatory to produce a *difference signal* so that the execution of the *correction step* is possible[4]. Matching a detected feature z to its map counterpart $\hat{z}$ is an interesting problem. In short, there is a sequence $s = \begin{bmatrix} i_1 & i_2 & \dots & i_N \end{bmatrix}$ that correctly correlates the indexes of extracted N features to the index of map features M:

$$\{(z_1, \hat{z}_{i_1}), (z_2, \hat{z}_{i_2}), \dots, (z_N, \hat{z}_{i_N})\}, i \in [1, M]$$

The solution to the matching problem is to find the *maximum likelihood correspondence*:

$$s^{\star} = \arg\max_{s} \sum_{f=1}^{N} \frac{1}{(2\pi) \mid \Sigma_f \mid^{1/2}} \exp\big(-\frac{1}{2}(z_f - \hat{z}_{s(f)})^T \Sigma_f^{-1} (z_f - \hat{z}_{s(f)})\big) \tag{2.43}$$

[4]In part of the literature, the Feature Matching problem is also known as the Data Association problem since it consists in finding the correct association between two data sets; the map and the measurements.

Note that generally the distance vector will have the form:

$$(z_f - \hat{z}_{s(f)}) = \begin{bmatrix} z_{f,x} - \hat{z}_{s(f),x} \\ z_{f,y} - \hat{z}_{s(f),y} \end{bmatrix} \tag{2.44}$$

Many particular heuristics can be added here for enhanced credibility, for example establishing safety thresholds for minimum acceptable likelihood. Note that equation 2.43 maximizes the *Probability Density Function*.

Finally, it is mandatory that Σ_f is in the same space than the features. One way to achieve this is to use, in this feature matching stage, the Cartesian representation for the features. In that case the Σ_f is taken from the present belief Σ with optional addition of uncertainty from the measuring process.

2.4.4.5: Correction Step

In this last step, the difference between pairs of matched features is used to correct the current belief and lower its uncertainty. Assuming that there is a matching $\hat{z}_{s(f)}$ feature for the one being observed z_f, $h(\mu)$ must predict the measurement for that feature:

$$\hat{z_f} = h(\mu, \hat{z}_{s(f)})$$

Equation 2.26 should then be specific to each observed-matching feature pair f:

$$x_k = \bar{x}_k + K_k\big(z_f - h(\bar{x}_k, \hat{z}_{s(f)})\big) \tag{2.45}$$

Furthermore, the H and Q matrices are dependent on feature's nature and representation. In this application, since two different kinds of features are used, the computation of the *correction step* selects those matrices according to the nature of the feature. The derivation of this specific matrices are shown together with some discussion about the effect of a feature nature on the final localization update.

Reflector Beacon Feature As shown in Equation 2.29, beacon readings are represented by their polar coordinates centered on the robot and map landmarks are represented by their Cartesian position. The choice of using the Cartesian representation for map landmarks is to ease map input, output and visualization during the development process. In depth, this choice is not of great impact since conversions between the two representations are simple and fast. In this work, some care is taken to maintain the

same representations used in most of the bibliography [27].

For punctual landmarks, $h(\mu)$ is simply the transformation from its Cartesian representation to the polar representation with its origin on the robot. Given a landmark $m_i = \begin{bmatrix} m_{i,x} & m_{i,y} \end{bmatrix}^T$ and a current pose belief $\xi = \begin{bmatrix} x & y & \theta \end{bmatrix}$, the measurement *prediction* is:

$$h(\xi, m_i) = \hat{z}_i = \begin{bmatrix} \sqrt{(m_{i,x} - x)^2 + (m_{i,y} - y)^2} \\ atan2(m_{i,y} - y, m_{i,x} - x) - \theta \end{bmatrix} \tag{2.46}$$

which is in fact the measurement model for that kind of landmark.

The Jacobian of the measurement model, defined in Equation 2.22, computed using 2.46, is

$$H = \begin{bmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial y} & \frac{\partial \rho}{\partial \theta} \\ \frac{\partial \alpha}{\partial x} & \frac{\partial \alpha}{\partial y} & \frac{\partial \alpha}{\partial \theta} \end{bmatrix} \tag{2.47}$$

$$= \begin{bmatrix} \frac{-(m_{i,x}-x)}{\sqrt{q}} & \frac{-(m_{i,y}-y)}{\sqrt{q}} & 0 \\ \frac{(m_{i,y}-y)}{q} & \frac{-(m_{i,x}-x)}{q} & -1 \end{bmatrix} \tag{2.48}$$

with, for simplicity:

$$q = (m_{i,x} - x)^2 + (m_{i,y} - y)^2$$

The last specific entity is the measurement noise:

$$Q = \begin{bmatrix} \sigma_\rho^2 & \sigma_\rho \sigma_\alpha \\ \sigma_\rho \sigma_\alpha & \sigma_\alpha^2 \end{bmatrix} \tag{2.49}$$

It is interesting to note that adopting ρ and α as independent (Q is diagonal) gives good practical results. In practical terms, it is very time consuming and sometimes complex to estimate Q accurately. In order to address this limitation, one can stipulate lower bounds for σ_ρ^2 and σ_α^2 because although using values higher than the actual degrades the performance of the system, it constitutes a, sometimes desired, conservative approach.

In the application hereby described, what accounts for the largest error source is the movement of the robot itself because, even though care is taken to accurately consider each measurement's *timestamp*, the laser range finder measurement takes considerable time and performing them while moving inevitably produces drifted data.

Equation 2.50 shows our empirical model for computing Q; ν being the tangential and ω the angular current velocities.

$$Q = \begin{bmatrix} (0.1 + |\nu|0.6)^2 & 0 \\ 0 & (0.017 + |\omega|0.6)^2 \end{bmatrix} \tag{2.50}$$

In order to illustrate the effects of correction steps using punctual landmarks, Figure 2.10 shows a robot driving (in reverse) from a very uncertain pose belief (Fig. 2.10a) until it encounters a landmark (Fig. 2.10b). After this detection, the new uncertainty is reduced in the direction of the detected landmark but not in that of the dashed yellow circle. After some more belief updates using only that first landmark, the uncertainty continues to shrink in the direction of that landmark but not on the other directions (Fig. 2.10c). In order to reduce the uncertainty even further, the robot starts to spin (Fig. 2.10d and 2.10e) in search for another landmark (note the *active* localization behavior used in this demonstration). When the detection of the second landmark occurs, the uncertainty matrix (and thus the drawn *2D only* ellipsis) shrinks even further until reaching the artificial (i.e. added by software) saturation (Fig. 2.10f). On the map, white dots represent known landmarks present in the map, blue ones represent a match between a detected and predicted landmark.

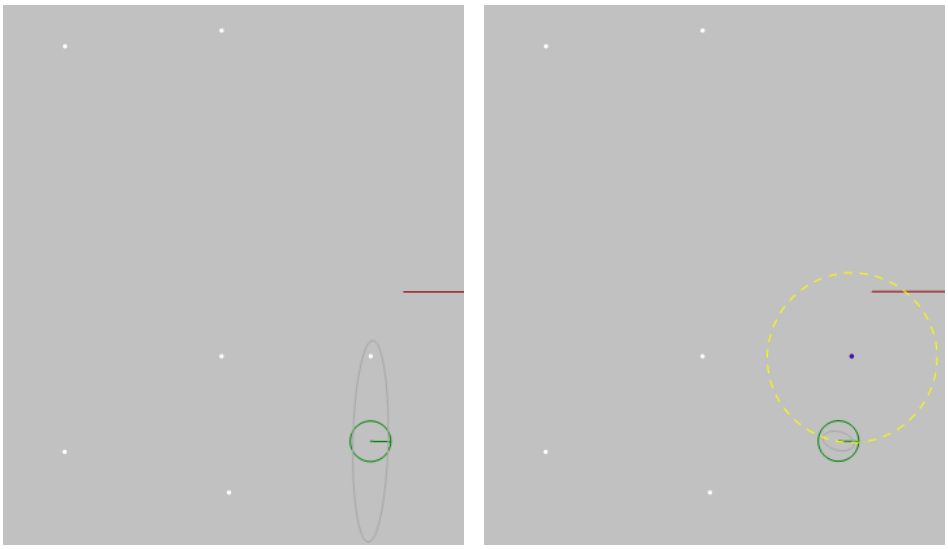
Line Feature Line features are represented by the polar representation of an infinite line (see Equation 2.30) with the origin coinciding with the global reference frame's one. This representation for lines, although not very intuitive, is useful in this application; normal line representations may cause numerical problems for certain lines.

For line features, $h(\mu)$ is: for ρ) the distance function between a $2D$ point on the robot and a line; for θ the difference between the robots bearing and the line's angle in its polar representation. Given a line $l_{map} = \begin{bmatrix} \rho_{map} & \beta_{map} \end{bmatrix}^T$ in the global reference frame and a current pose belief $\xi = \begin{bmatrix} x & y & \theta \end{bmatrix}$, the measurement *prediction* (i.e. measurement model) is:

$$h(\xi, l_i) = \hat{z}_i = \begin{bmatrix} |\varphi| \\ \beta \end{bmatrix} \tag{2.51}$$

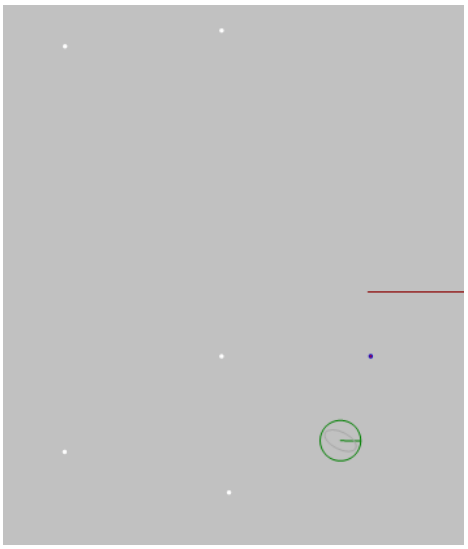
with:

$$\beta = \begin{cases} \beta_{map} + \pi & \text{if } \varphi < 0 \\ \beta_{map} & \text{otherwise} \end{cases} \tag{2.52}$$

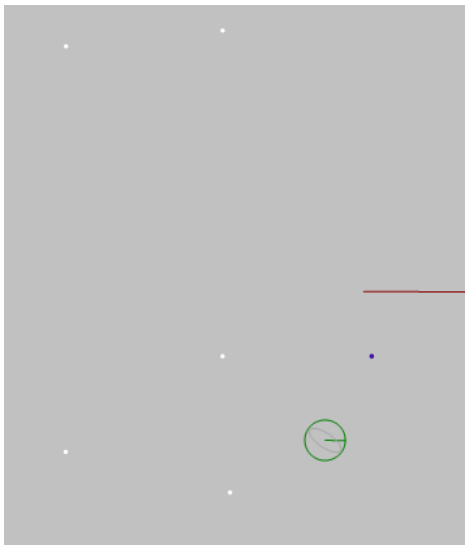


(a) Uncertain belief

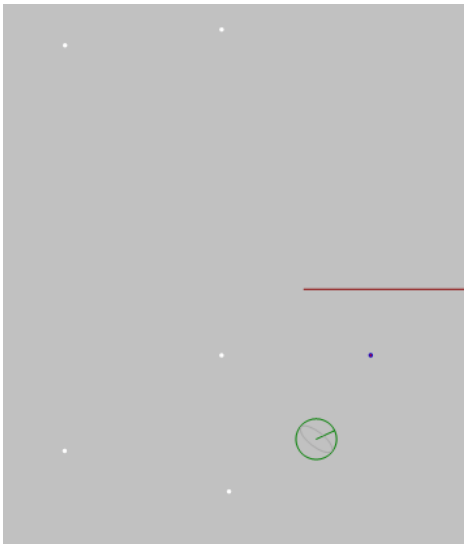
(b) First detection (blue dot)



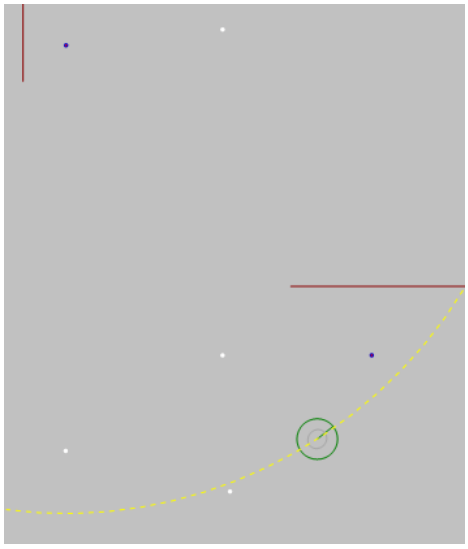
(c) Other updates using the same landmark



(d) Other updates using the same landmark



(e) Spinning to search other landmark



(f) Detection of the second landmark in the upper left corner

Figure 2.10: EKF Localization performance using punctual landmarks through time

and:

$$\varphi = \rho_{map} - \sqrt{x^2 + y^2}\cos\psi \tag{2.53}$$

$$\psi = \beta_{map} - atan2(y, x) \tag{2.54}$$

The Jacobian of the measurement model, defined in Equation 2.22, computed using 2.51, is

$$H = \begin{bmatrix} \frac{\partial\rho}{\partial x} & \frac{\partial\rho}{\partial y} & \frac{\partial\rho}{\partial\theta} \\ \frac{\partial\beta}{\partial x} & \frac{\partial\beta}{\partial y} & \frac{\partial\beta}{\partial\theta} \end{bmatrix} \tag{2.55}$$

$$= \begin{bmatrix} -\cos\bar{\beta} & -\sin\bar{\beta} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{2.56}$$

The last specific vector is the measurement noise, Q, is shown at Eq. 2.57.

$$Q = \begin{bmatrix} \sigma_\rho^2 & \sigma_\rho\sigma_\beta \\ \sigma_\rho\sigma_\beta & \sigma_\beta^2 \end{bmatrix} \tag{2.57}$$

This matrix is estimated similarly to the one for punctual landmarks (See Eq. 2.50).

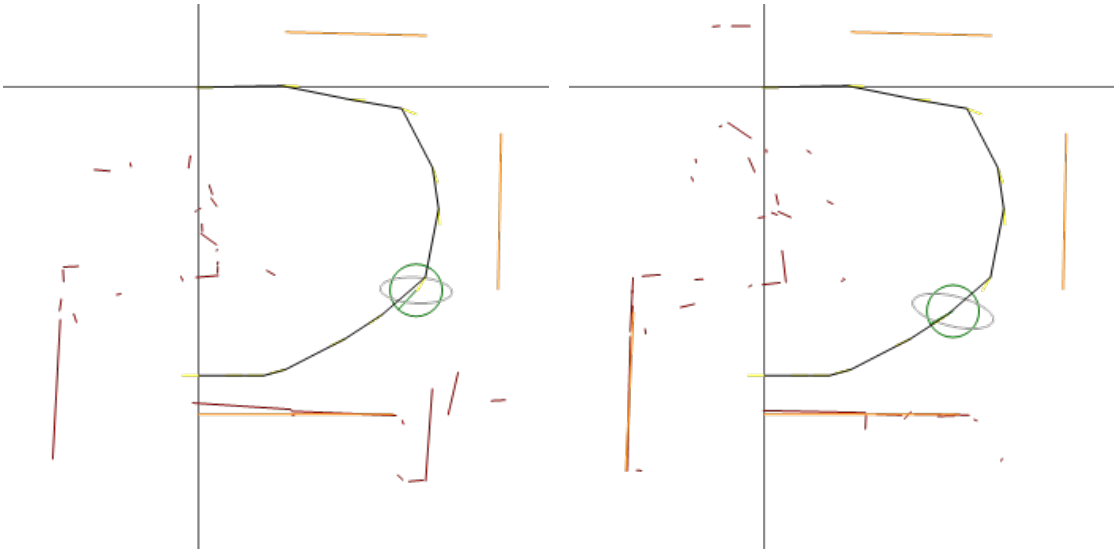
2.4.4.6: Asynchronous Decoupled Implementation

In most applications, the signals from encoders and sensors are produced in different rates or even in sparse rates; the execution time for some feature extraction algorithms is dependent on the input data and thus varies *randomly* between lower and higher known extremes. Additionally, in some situations the feature extraction algorithm won't produce any results at all (due to noise or to the actual absence of features).

This heterogeneous nature of the inputs to the EKF gives rise to an interesting problem: at which rate should the EKF run?

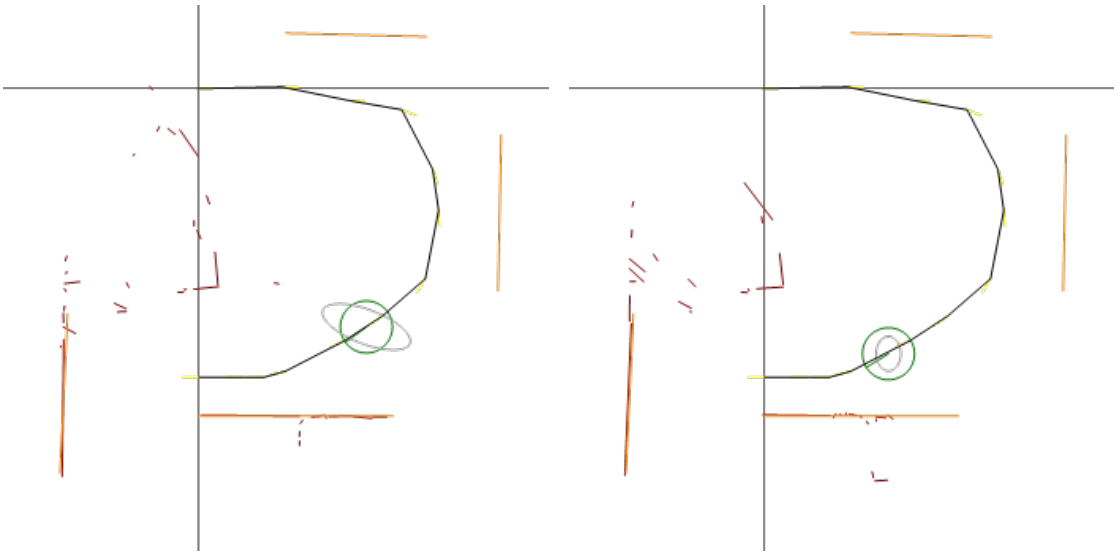
The most trivial solution consists in using the rate from the slowest of the input signals; in this way the data of the inputs with fastest rates are accumulated in between the two iterations to be used in the next iteration which occurs when data from the slowest input source arrives. Another way is to use Asynchronous Holds to extrapolate the input signal of the inputs with lower rates. This last approach has been employed by [2].

A third and more simple approach is to asynchronously execute the prediction



(a) Uncertain belief, orientation error visible

(b) Uncertainty shrinks only in vertical direction due to line in the bottom of the image



(c) Prediction drifts the uncertainty ellipse

(d) Uncertainty shrinks in horizontal direction due to verical line at the left side of the image

Figure 2.11: EKF Localization performance using line features through time in a real experiment. Dark red lines are line features currently being observed, dark orange ones are stored in the map and the black path is the one being followed by the robot.

and correction steps, executing each one when its input data is available: the prediction step whenever odometry data *arrives* and thus in a higher rate and the correction step whenever abstract features, extracted from the LRF's data, *arrives*. This design choice enhances the system performance in numerous ways. It is easy to realize that even when no belief update is possible (absence of extracted features), the *prediction step* itself is useful as it provides a reasonable position belief together with it's estimated uncertainty. Other advantages of this approach include:

- Implicitly solves the problem of multi-rate sensor's data;
- If desired, addition of Multiple Hypothesis Tracking (MHT) in the future is straight forward
- Robot's state estimation (which is used intensively by other services) is decoupled from intensive computation services. This provides great performance improvement when in multi-core platforms.

Additionally, section 5.6 gives a solution to the problem of localization in real-time in the context of the solution we implemented. This solution improves feature extraction accuracy and handles "old" localization corrections by a technique similar to refiltering.

2.4.4.7: Further Reading

Interesting readings on this topic and into using the extended Kalman filter also for mapping include [28], [19], [18] and [23].

2.5: Particle Filter Based

Another modern approach to the problems of localization and mapping for mobile robots uses particle filters (PF) or hybrid approaches. These algorithms are capable of handling non-linear environment treating them as such (not as noise) and are capable of representing multi-modal non-gaussian probability distributions. PFs use a finite set of *particles* to express the current estimate.

This approach relies on the characteristic that given a robot (represented by a particle) in a certain pose in a certain map, its sensors' values can be predicted

and thus compared against the actual values generating a *distance*[5] measurement in a multi-dimensional space, particles with *small* distances (i.e. *good particles*) are preserved while the others are removed. Furthermore, continuous re-sampling is done around good particles to increase accuracy and an equivalent of EKF's prediction step is applied to each particle in between two measurements.

This approach is considered to be the most promising technology in this field; it outperforms all the previous technologies in most metrics. Its most notable pitfalls are its high computation costs and memory usage.

One of the most popular techniques using this approach is called *Montecarlo Localization*. More information on these techniques can be found in [27], [10], [9] and [22].

[5]Not necessarily the euclidean distance; in most cases, statistical distances are used to thread different dimensions adequately. One of the most popular distance measures employed in this case is the Mahalanobis Distance

Chapter 3: Pose Control

Another important component of a mobile robot's software system is its pose control system. Although some *ad hoc* alternatives can easily be implemented, a general feedback controller that drives a robot between any two poses is very desirable.

In short, this can be described as a feedback control system that drives the robot from one pose $\xi_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^T$ to any other $\xi_g = \begin{bmatrix} x_g & y_g & \theta_g \end{bmatrix}^T$ by acting with the control vector $U_c = \begin{bmatrix} \nu & \omega \end{bmatrix}^T$.

The problem of designing such control system is to establish, first, an error signal $e(t)$ that represents the distance between the current and target poses and then a controller:

$$U_c = f_{con}(e(t)) \tag{3.1}$$

that produces the input U_c to the system so that the error $e(t)$ goes to zero in permanent regime:

$$\lim_{t \to \infty} e(t) = 0 \tag{3.2}$$

In this chapter, such general pose controller for differential drive mobile robots is presented together with a brief kinematic modeling of this class of robots.

3.1: Differential Drive Robot Model

3.1.1: Justification

For lightweight mobile robots, it is common to neglect the system's dynamic in favor of a simple kinematic model. This is reasonable by a number of reasons:

- **Fast Dynamic:** in normal operation, the robot's dynamic is very fast due to it's small inertial momentum and can thus be treated as noise.
- **Absence of external forces:** in general it is assumed that no external forces are imposing work on the robot. For example, for a stable robot operating in a plain floor, gravity is constrained by the robot-floor interaction and thus producing no effects at all, apart from enabling the robot movement through friction.

3.1.2: Control Signal Representation

One of the most popular representations for the control signal for differential drive robots uses its tangential ν and angular ω velocities relative to the robot's reference frame. This representation simplifies the system's model and is also very intuitive for debugging and monitoring. The mapping to this representation from the velocity of each wheel (each motor in most applications) is presented here for completeness. Given that:

- φ_r is the angular speed of the right wheel and φ_l of the left one
- r_w is the radius of the driving wheels
- l_w is the displacement between the wheels and the center of the robot (supposedly the same for both wheels)

the direct transform $T_{w\to p}$ can be understood as the sum of the motion contribution of each wheel to the movement of the robot first in the tangential and then in the angular directions:

$$\begin{bmatrix} \nu \\ \omega \end{bmatrix} = \underbrace{r_w \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2l} & -\frac{1}{2l} \end{bmatrix}}_{T_{w\to p}} \begin{bmatrix} \varphi_r \\ \varphi_l \end{bmatrix} \tag{3.3}$$

Finally, the inverse transformation $T_{p\to w}$ is equal to $T_{w\to p}^{-1}$ and is always defined for $l \neq 0$.

3.1.3: Kinematic Model

In the global reference frame and with Cartesian coordinates, the robot velocity resulting from its relative velocity is:

$$\underbrace{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}}_{\dot{\xi}_\perp} = \underbrace{\begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}}_{R} \begin{bmatrix} \nu \\ \omega \end{bmatrix} \tag{3.4}$$

with $\dot{\xi}_\perp$ is the robot's velocity in Cartesian coordinates.

Another way to represent the robot's pose is to use polar coordinates. Figure 3.1 shows these two representations. In this application it is useful to take the target

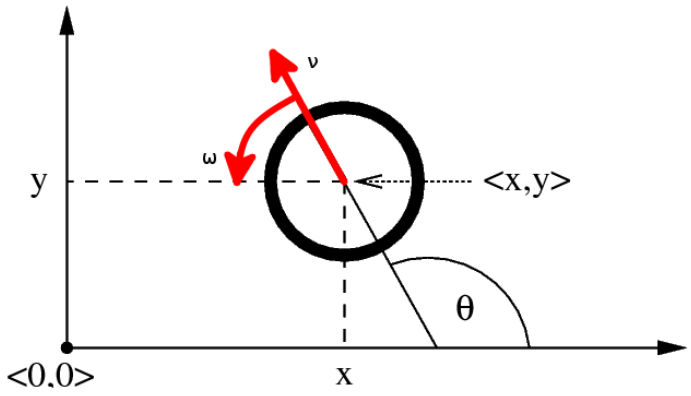


Figure 3.1: Robot's velocity in cartesian and polar representations.

pose (including bearing) coincident to the origin of such polar coordinates system; in that case the current position of the robot can be expressed by:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{3.5}$$

$$\alpha = -\theta + atan2(\Delta y, \Delta x) \tag{3.6}$$

$$\beta = -\theta - \alpha \tag{3.7}$$

which can be interpreted as:

- α is the difference in the robot's orientation and the line between current and target pose (say line ψ)
- ρ is the planar distance between current and target pose's *positions* (length of ψ)
- β is the angle difference from line ψ and the target pose

Figure 3.2 shows graphically the three values for one typical situation. Furthermore, the system can now be described in this new variables:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & -1 \\ -\frac{\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} \nu \\ \omega \end{bmatrix} \tag{3.8}$$

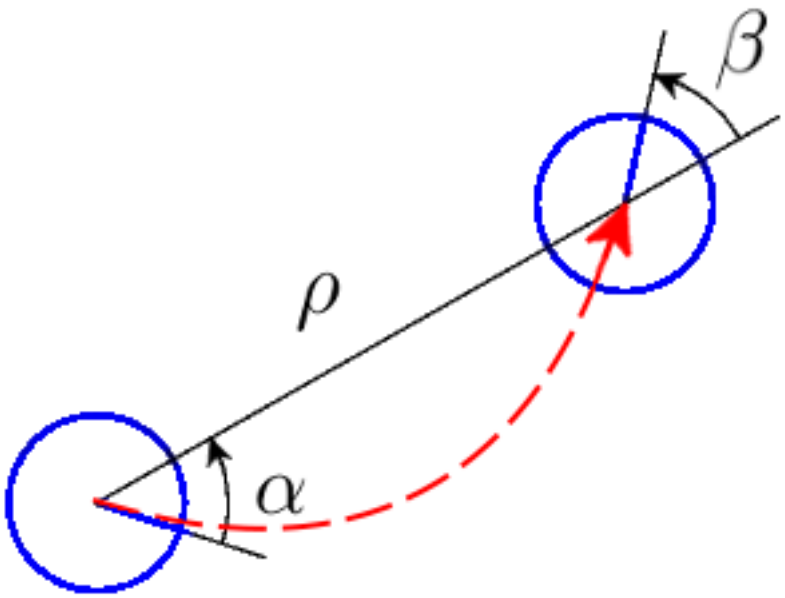


Figure 3.2: Pose controller errors definition. The current and target poses are shown in blue, the red dashed arrow depicts one possible trajectory to the target pose and both angles (α and β) orientations are denoted by the black arrows.

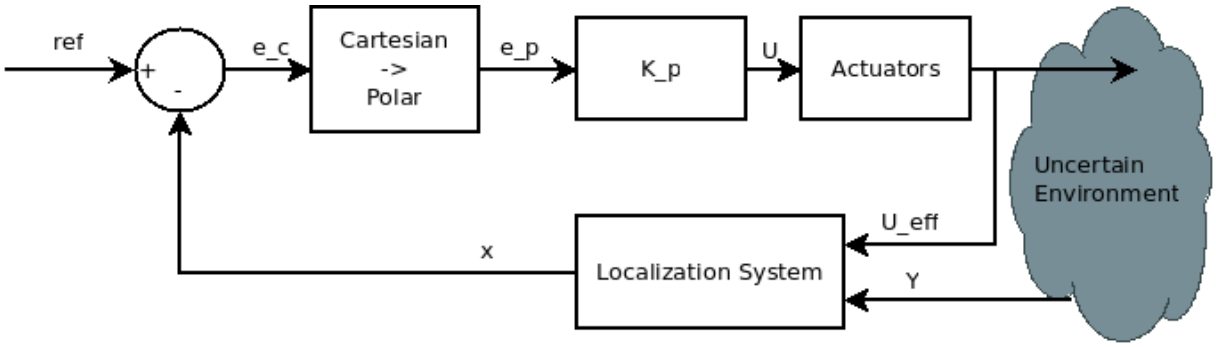


Figure 3.3: Pose Control Scheme

3.2: Control Law

Figure 3.3 shows a typical scheme of pose control for mobile robots using concurrent localization systems. The coordinates transformation of the error signal is presented to emphasize that this choice provides many advantages for this application.

The control law design consists in finding K_P that drives the error to zero as stated in equation 3.2.

In traditional linear multivariate feedback systems, K_P is usually designed by pole placement techniques so that the closed-loop poles of the system are stable. This

approach is of no avail in this case due to the *nonlinear* nature of the system's model. In this case, the approach adopted is to empirically suggest one control scheme and then evaluate its stability.

One fairly intuitive, if given the previously mentioned polar representation of the system, solution is presented in [26]:

$$\underbrace{\begin{bmatrix} \nu \\ \omega \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} k_\rho & 0 & 0 \\ 0 & k_\alpha & k_\beta \end{bmatrix}}_{K_P} \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} \tag{3.9}$$

From some superficial analysis of the structure choice for K_P, it can be noted that:

- the tangential velocity ν is computed solely from the planar distance to the target position ρ
- the angular velocity ω is computed from from both α and β
- the tangential velocity never changes direction and is always positive for $k_\rho > 0$ since ρ is always positive

3.3: Stability Discussion

The closed-loop system is obtained from plugging the equation 3.9 on equation 3.8 producing:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos\alpha \\ k_\rho \sin\alpha - k_\alpha - k_\beta \beta \\ -k_\rho \sin\alpha \end{bmatrix} \tag{3.10}$$

which has only the one desired equilibrium point at $(\rho, \alpha, \beta) = (0, 0, 0)$ and no singularity at $\rho = 0$ (the open-loop system had a singularity for this situation). In order to take advantage of the Grobman-Hartman[1] theorem to assure the stability of this system, the following must be true:

Condition 1 *All the eigenvalues of the closed-loop system linearized at the equilibrium point have nonzero real parts*

[1]Due to historical reasons and not to naming conventions, this theorem is also known as the Hartman-Grobman theorem

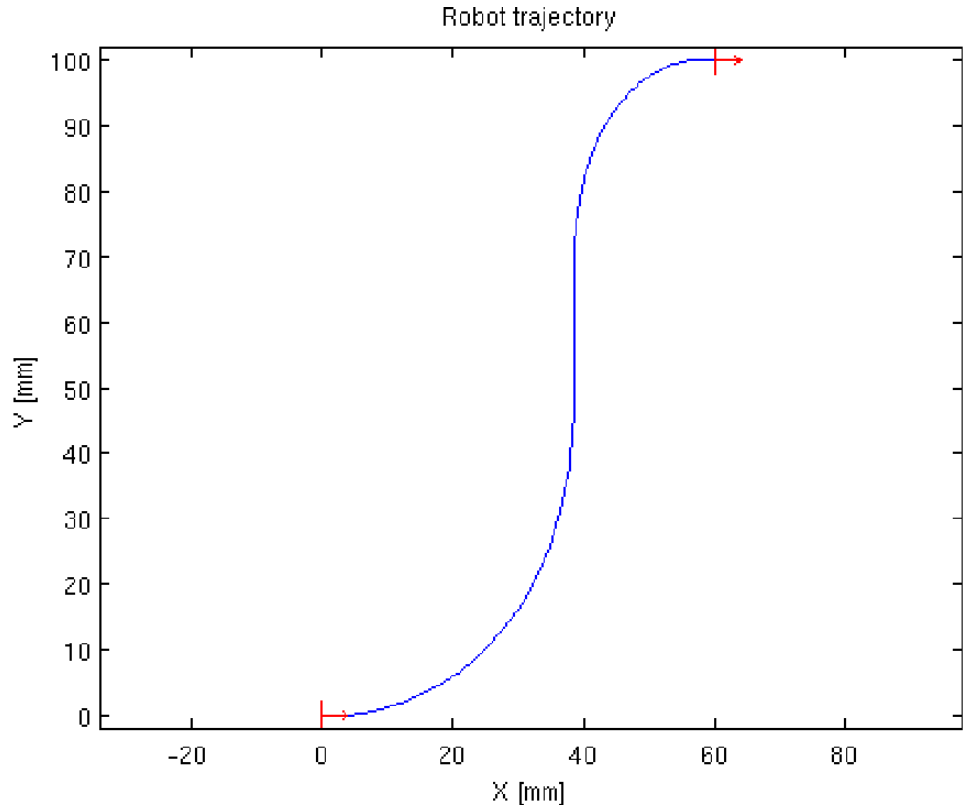


Figure 3.4: MATLAB® Simulation of pose control from the origin to $\big(60, 100\big)$ with gains $k_\rho = 3$, $k_\alpha = 8$ and $k_\beta = -3.5$.

Using the small-angle approximation:

$$\text{for } x \approx 0 \begin{cases} \cos x \approx 1 \\ \sin x \approx x \end{cases}$$

the linearization of equation 3.10 at the equilibrium point gives:

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \underbrace{\begin{bmatrix} -k_\rho & 0 & 0 \\ 0 & -(k_\alpha - k_\rho) & -k_\beta \\ 0 & -k_\rho & 0 \end{bmatrix}}_{A_{eq}} \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} \tag{3.11}$$

which eigenvalues must satisfy condition 1. That can be evaluated using the characteristic polynomial for A_{eq} and using the Routh-Hurwitz stability criterion for:

$$\begin{aligned} 0 &= (\lambda + k_\rho)(\lambda^2 + \lambda(k_\alpha - k_\rho) - k_\rho k_\beta) \\ &\quad \lambda^3 + k_\alpha \lambda^2 + k_\rho(k_\alpha - k_\rho - k_\beta)\lambda - k_\rho^2 k_\beta \end{aligned} \tag{3.12}$$

which gives:

Table 3.1: Table for evaluation using the Routh-Hurwitz stability criterion

1	$k_\rho(k_\alpha - k_\rho - k_\beta)$
k_α	$-k_\rho^2 k_\beta$
$\frac{-k_\alpha k_\rho k_\beta + k_\alpha k_\rho (k_\alpha - k_\rho)}{k_\alpha}$	0
$-k_\rho^2 k_\beta$	0

Stability is granted if there are no sign changes on the first column; this can be accomplished by making:

$$\begin{aligned} k_\alpha &> 0 \\ k_\beta &< 0 \\ k_\rho &> 0 \\ k_\alpha - k_\rho &> 0 \end{aligned} \tag{3.13}$$

3.4: Reverse Driving

So far, it was not considered that the robot can in some applications[2] drive in reverse suggesting that the controller could switch its structure to favor the shortest path; driving forward or reverse. In short, reverse driving can be achieved simply by virtually flipping the heading, $\theta_{rev} = \theta + \pi$, for both the robot's current and goal poses and making $\nu_{rev} = -\nu$

The decision between forward or reverse driving is also simple and can take into account the application itself. This reasoning could be:

$$sign(\nu) = \begin{cases} +1 & \text{if } |\alpha| \geq \phi \\ -1 & \text{if } |\alpha| < \phi \end{cases}$$

With $\phi = \frac{\pi}{2}$ to perform the shortest path from current to goal poses. Other values of ϕ can be adopted to favor reverse or forward driving since in both ways the system is equally stable.

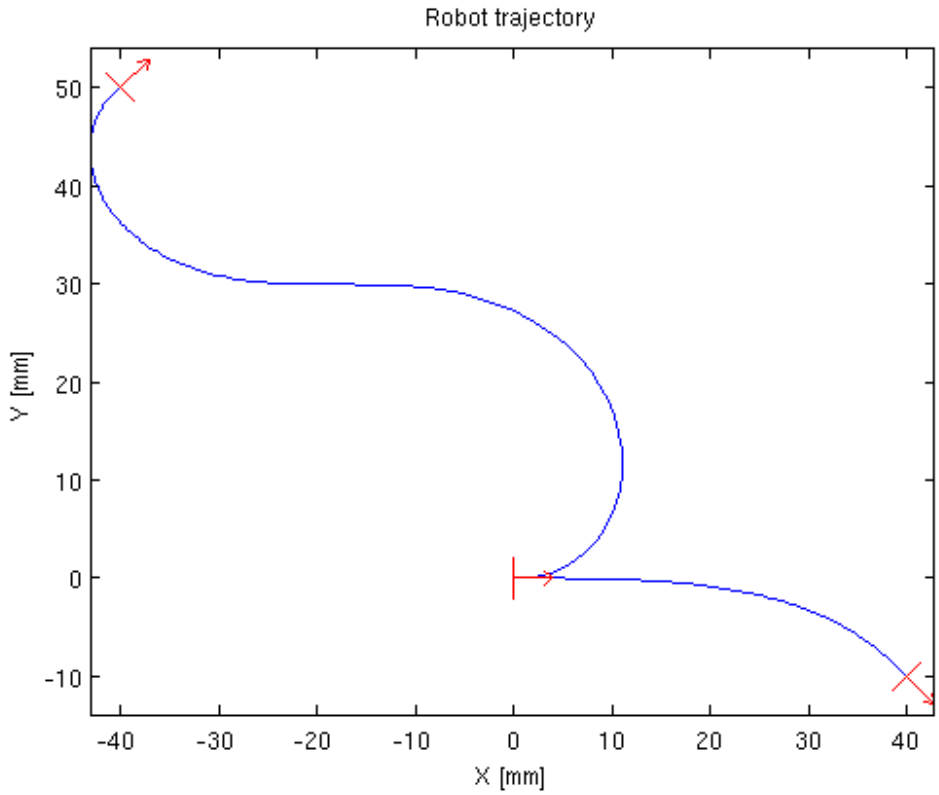
Figure 3.4 depicts the resulting trajectories. Note the difference between figures 3.5a and 3.5b when driving to $\big(-40\ 50\big)$.

3.5: Final Considerations

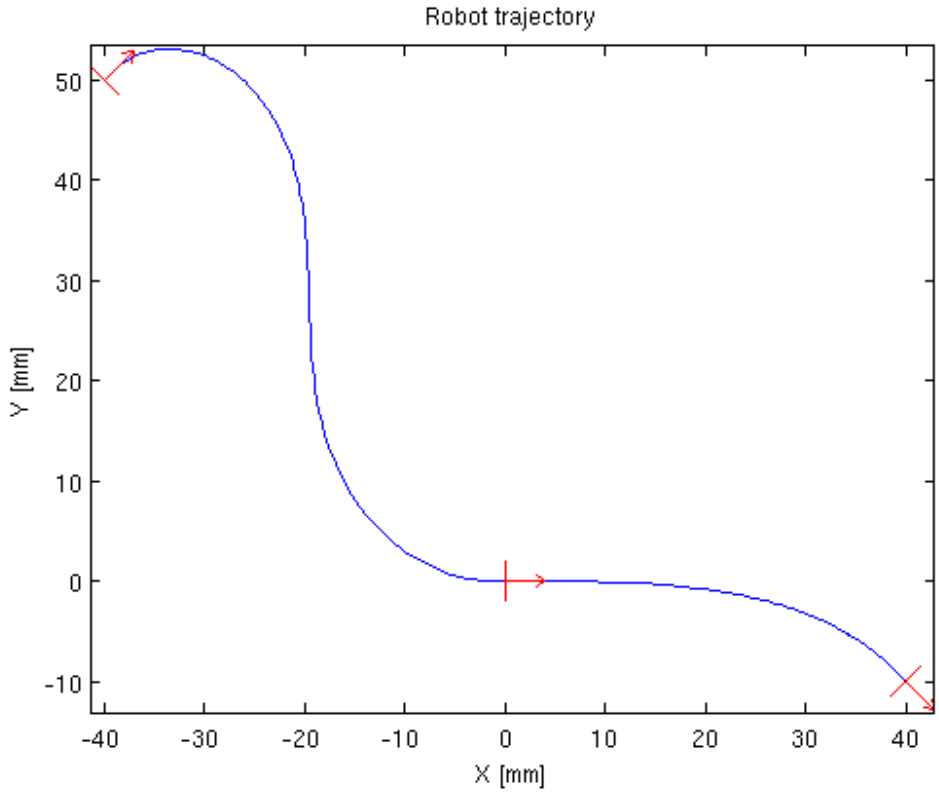
One final relevant remark is that the pose feedback signal is retrieved from the current belief of the concurrent SLAM service, thus generating the trajectory in an "on line" manner corrected at each step. This leads to some interesting stability issues since the pose feedback signal is not smooth; belief updates generate "jumps" when recovering from high pose uncertainty situations. For example, if the controller's choice (see section 3.4) is carried only one time, a little overshoot would make the robot perform complex sharp turns forming an "***S***" when the same result could be achieve by simply switching the motion direction.

To overcome this, the choice between forward and reverse driving can be carried at every time step ensuring that the *best* direction is chosen but this raises another problem: the robot can get trapped if $|\alpha| = \phi$, switching indefinitely but not moving towards the goal pose. To addressing this last problem a hysteresis based protec-

[2]Safety instruments for collision detection are sometimes only available at the robot's front and thus reverse driving is undesired



(a) Without reverse drive support



(b) With reverse drive support

Figure 3.5: MATLAB® Simulation of the robot starting at $(0\ 0)$ performing two maneuvers, to $(-40\ 50)$ and $(40\ -10)$ without reverse drive support (3.5a) and then with (3.5b)

tion mechanism is added to prevent such consecutive switches not comprising stability since the controller rapidly drives $|\alpha|$ away from ϕ always shortening the error signal.

Simillar controllers can be found in [4] and [17] and other approaches and improvements can be found in [8], [21].

Chapter 4: Services Oriented Architecture for Mobile Robots

Services Oriented Architecture (SOA, also knwon as Service Oriented Computing) is a new programming paradigm that aims at systems in which there is loose coupling between different software modules, called services. It is believed that SOA software systems will exhibit a higher level of flexibility and easier maintenance than their non-SOA counterparts. This systems are characterized by well defined standardized generic contracts (i.e. interfaces) for each class of offered service. In this systems, modules are encapsulated so that each module provides a *service*, thus its name, specified in one or more of those contracts and utilizes the *services* from other modules also specified in those contracts. Optionally, a *services directory* service is added to the "network"[1] and its address is broadcast to all services.

In normal operation, a service proceeds to perform a certain operation, maybe requested by another service, using its own resources (e.g. database, hardware, computation power, algorithm etc) and other services. The loose coupling of services stems from the fact that the coupling is done at runtime and mandated by contracts only.

The communication medium and data structure used by this systems are generally of high abstraction levels and are intended to ease integration with as many different programming languages and platforms as possible. For example, the most popular data structure is the eXtensible Markup Language (XML) which is human readable, simple and portable contributing even more to the system's flexibility and expandability.

In cases where a *services directory* is present, services generally notify the *services directory* of what contracts (e.g. services) they provide and query the *services directory* for a given contract to retrieve the address of a service that provides that desired contract.

In this kind of software, as in most modern software, the choice of the system's architecture (i.e. which contracts are provided by who) accounts for much of the system's success or eventual failure.

[1] Not necessarily a computer network but a mesh of loosely coupled components that share a communication medium

4.1: Justification

The employment of SOA technology for mobile robots is relatively recent since the abstract, high level layers necessary for SOA's loose coupling adds significant overhead that could only be coped with recently due to the advance of the embedded computing power.

Another recent development that leveraged the use of SOA for robotics software was the introduction of the Microsoft Robotics Developer Studio® (MRDS)[2] by Microsoft ® which provides a large framework and set of tools for developing this type of applications; this framework includes the Concurrency and Coordination Runtime (CCR) and the Decentralized Software Services (DSS) which together consist in a software platform to construct applications that demand concurrent and decentralized, loosely coupled, operation.

Common robotics software is responsible for handling multiple sensors' input, perform complex computations from a plethora of sensor data and generate outputs to control multiple actuators; all this concurrently. Robotics instrumentation is continuously evolving and offers a very wide range of devices each demanding a specific software module to interact with. This heterogeneity suggests that a decentralized, loosely coupled, environment is necessary so that control and coordination softwares can be written independent of the underneath instrumentation technology. The MSRS was conceived to address those issues; with the DSS providing a complete housing for decentralized software using the SOA paradigm.

4.2: Related Works

During a preliminary search for related works using both SOA and MRDS little work on formalizing a standard architecture that supports more complex robotics applications (e.g localization, pose control) were found:

- [6] and [16] address the use of SOA and MRDS for simple applications but do not focus on more complex issues.
- [13] comprises a more complex application but focuses more on the simulation environment and does not address the localization issue.

[2] Until 2008 it was called Microsoft Robotics Studio - MSRS

- [7] comprises the localization problem but in a different environment, using different techniques.

After this analysis we decided to design an architecture scheme from scratch to meet our specific requisites but paying attention to keep the services generic enough so that it could be employed in different applications. Differently from what was found in the literature, in our system the SOA approach plays a significant role on the localization algorithm implementation.

4.3: Design

In this context, the design of the system architecture consists in establishing which functionalities should be tightly coupled and which should be loosely, in other words to decide how granular the modules separation should be in between a big monolithic solution and a large granular mesh of elementary function providers. In designing such systems there is a trade-off between flexibility and computation overhead among other interesting factors.

Maybe one of the main contributions of this work is to establish a satisfactory, services oriented, architecture that would ease the implementation of similar robotics software.

Given the importance of the localization system (better discussed in Chapter 2), the developed architecture takes into account many inside particularities of the used localization systems and some previous knowledge of localization algorithms for mobile robotics may be necessary to comprehend in full extent the choices made in this stage.

As a first step, the most desired properties of the system should be summarized so that any design iterations can be checked against this desired properties. Defining this properties itself is part of the architecture design since it establishes the first architectural constraints. The following list shows this desired properties:

Property 1: Many systems should have access to the best localization estimate at all times.

Property 2: The Localization system should work concurrently and transparently to other services.

Property 3: The Localization system should be independent of hardware and sensor nature.

Property 4: Environment related data should always be feature-based and represented in a global reference frame.

Property 5: The Localization's prediction and correction steps should be decoupled to support asynchronous sensor data (see section 2.4.4.6).

Property 6: Robot and Environment states should be logically separated.

Property 7: The environment's state should be easily shared among more than one robot.

In order to comply with the aforementioned properties, we incrementally present sketches of services and their interactions so that each of the above is addressed but still paying attention to the trade-offs between simplicity, flexibility and computation overhead.

As discussed in chapter 2, there is no way to directly measure the robot's pose, any feedback system that needs that pose as feedback will use the localization system's output instead. Also, most sensors produce data in local reference frames and the robot's full pose is necessary to correctly transform this sensor data to the global reference frame. These uses for the localization system's output suggests that such information should be readily and directly available to any other services otherwise it would beat the purpose of having a localization algorithm in the first place.

One way to achieve this is to use a central *hub* that provides that information across all interested services. It is also important to note that localization estimate includes both the complete pose of the robot its estimated covariance matrix. Figure 4.1 shows conceptually the interaction of a variety of services that use such information and addresses **Property 1**. Note that in fact the blocks in these diagrams depict contracts and not services themselves; in this way, as far as one isolated service is concerned, the other services could be written in any programming language or implemented in any different forms as long as these services continued to obey those contracts.

To addresses **Property 2**, the localization system can be set to run in parallel as an independent service that updates the belief in the localization *hub* thus achieving concurrency and transparency. Figure 4.2 shows this separation. Following, figure 4.3

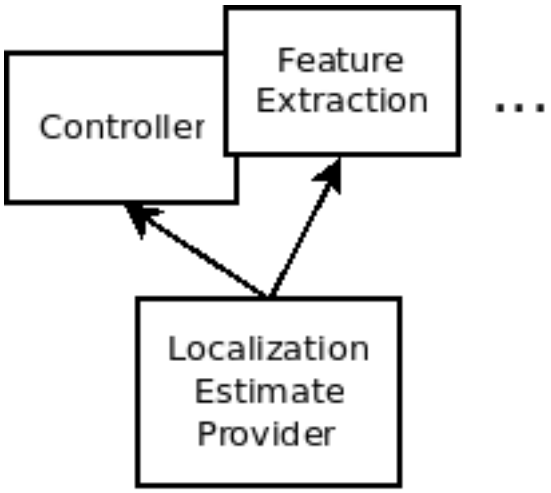


Figure 4.1: Property 1: Localization *hub*

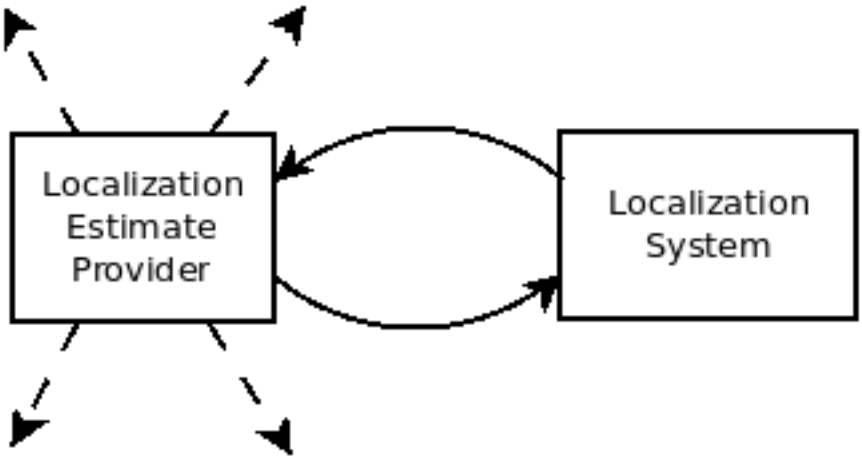


Figure 4.2: Property 2: Concurrent and transparent to other services.

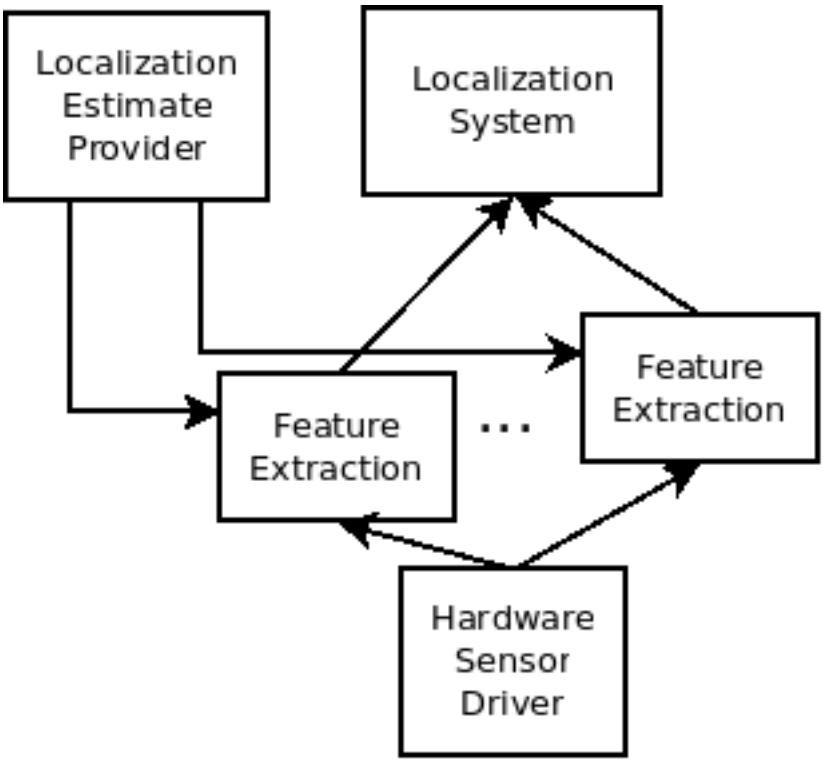


Figure 4.3: Property 3 and 4: Sensor's hardware independence due to the abstraction layer

addresses **Properties 3 and 4** and underlines the additional abstraction layer added by the feature extraction services. In this way higher level services can be developed without regard to the underlying hardware or sensor nature. A Laser Range Finder, for example, produces at the lower logical level an array of distance echoes and feature extraction services can be used to gather line features from that data. Cameras produce a large amount of raw data that needs a considerable commonly non-trivial computer vision system that extracts the useful information from it. See section 2.4.4.3 for more information on the usefulness of this approach.

As extensively discussed in chapter 2, there is a big advantage on separating the *prediction* and *correction* steps to address the problem os multi-rate sensors. This approach leaves room to continuous development of new *correction* step implementations. One could even use more than one of those services concurrently to address particular situations like a kidnapped robot. This separation, shown in figure 4.4 addresses **Properties 5, 6 and 7**. In is interesting to note that the belief regarding the environment is not necessary to perform the *prediction* step and that the *correction* step can be seen as a functional, state-less, transformation that takes three inputs:

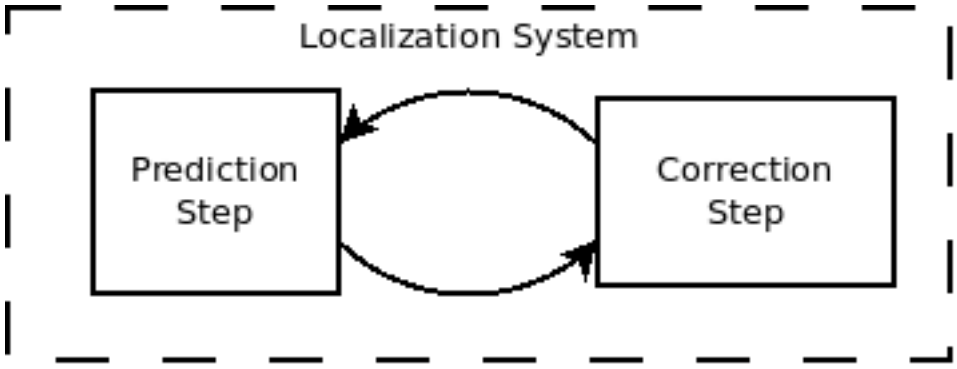


Figure 4.4: Criteria for props 5,6,7

- Robot pose belief
- Environment Map
- Measurement

to produce one new pose belief for that robot. This concept suggests that if separated, the service that performs the *correction* step can easily be shared among more than one robot operating in the same environment and with coincident global reference frames[3].

Figure 4.5 puts the aforementioned services together giving the impression of a first iteration of the system's architecture and its interconnections. An analysis of this first iteration in respect to the computation complexity assigned to each of those components and the amount of overhead in normal operation reveals that, although coined in by the aforementioned desired properties, one part of that architecture is inefficient and adds unnecessary complexity.

The choice of a dedicated "Localization Estimate Provider" service is justified by its *broadcasting* role and ideally high baud-rate but seems unnecessary for its *bridge only* character only adding overhead. The "Prediction Step" service also runs at the same high baud-rate (which is in fact the rate at which the localization belief is updated), and its operations are very lightweight since the *prediction* step itself is very simple. This scenario suggests that the "Prediction Step" itself can serve as a centralized, lightweight localization hub. Using the concept of figure 4.6 and adding additional necessary services, the system's architecture becomes the one at figure 4.7.

[3]The capability of sharing one service among different network nodes, possibly through the Internet, is native to the MRDS software.

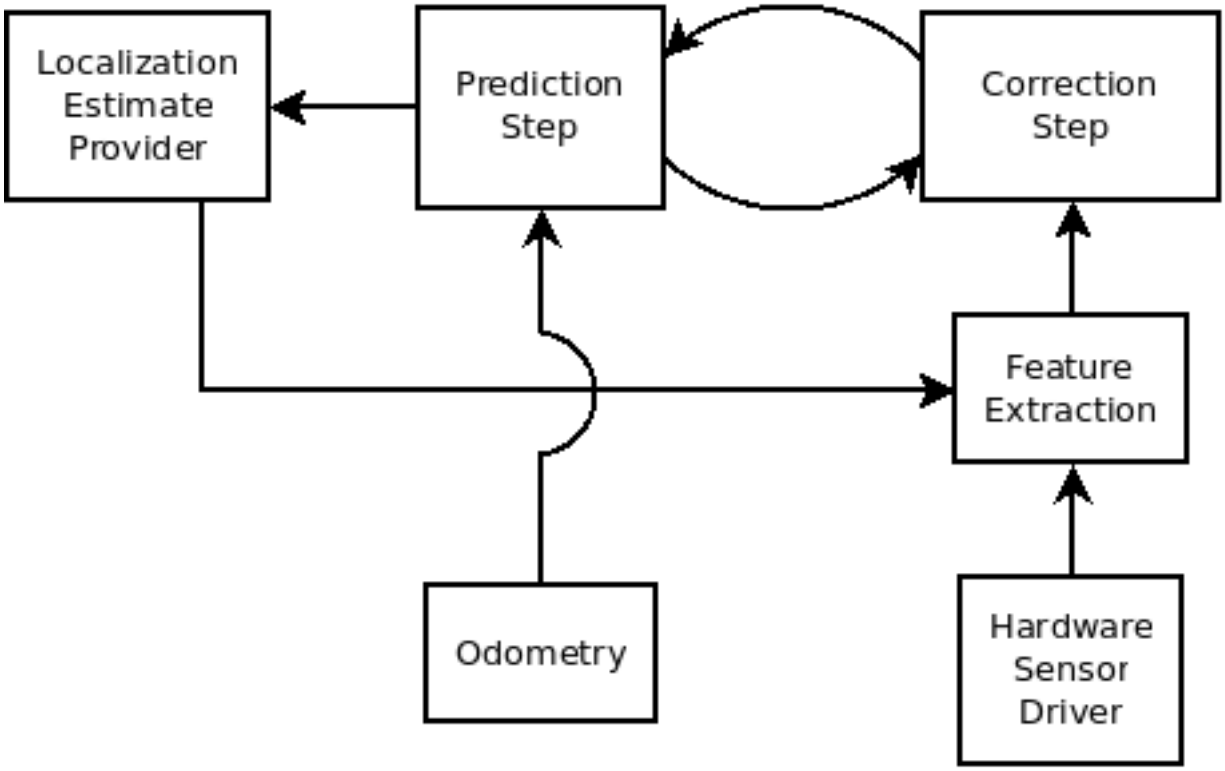


Figure 4.5: Intermediate Architecture proposal

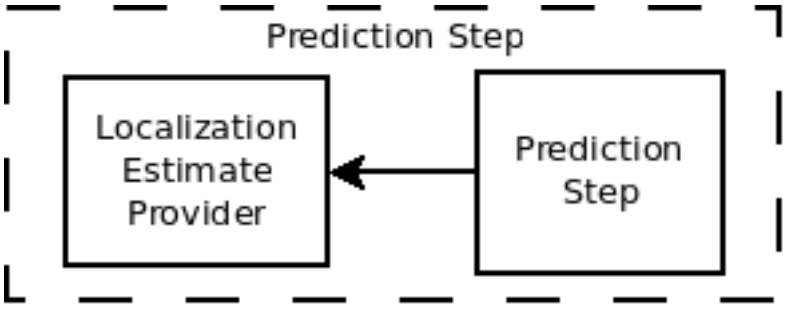


Figure 4.6: Merging the two services

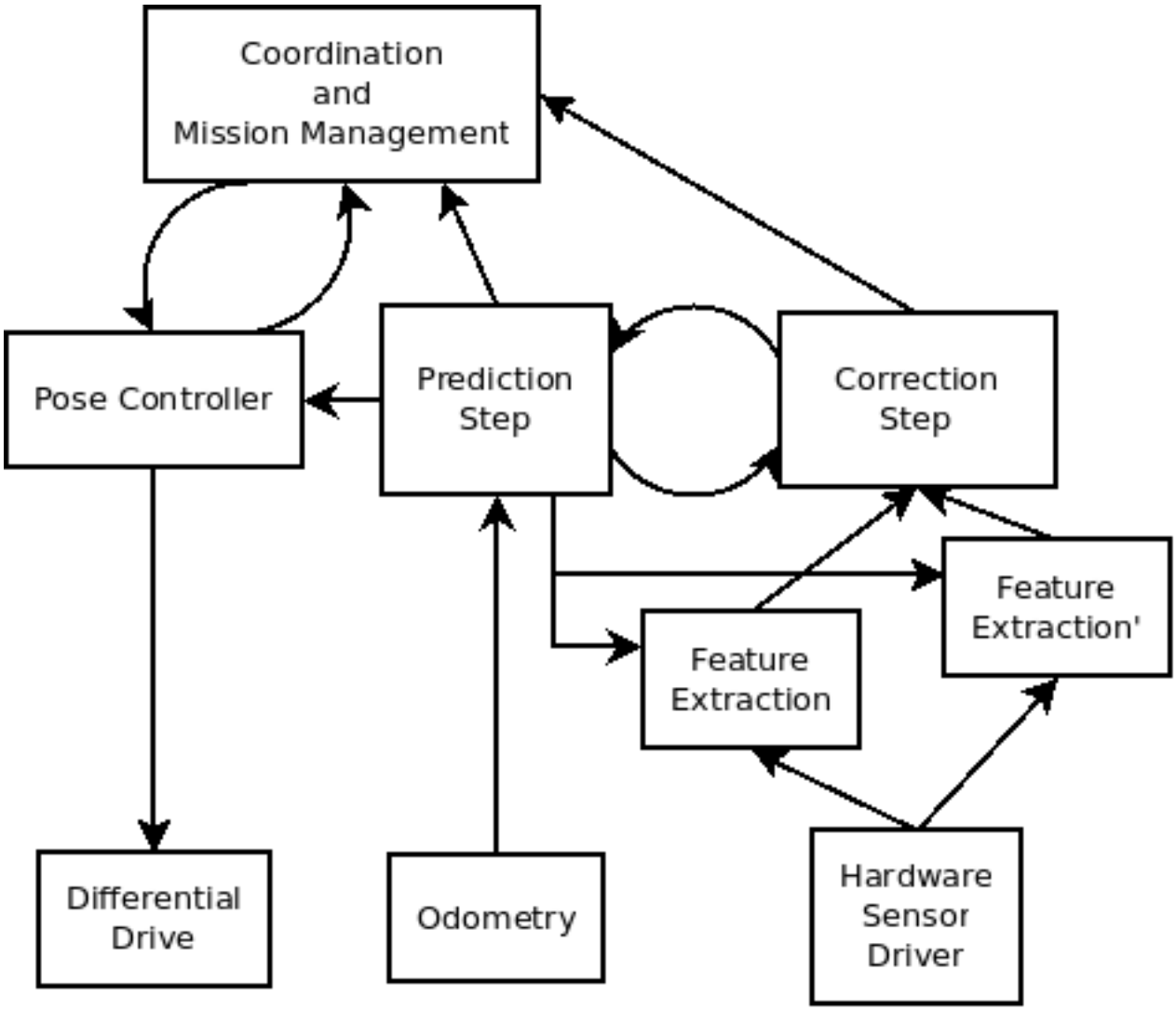


Figure 4.7: Proposed Services Oriented Architecture for Mobile Robots

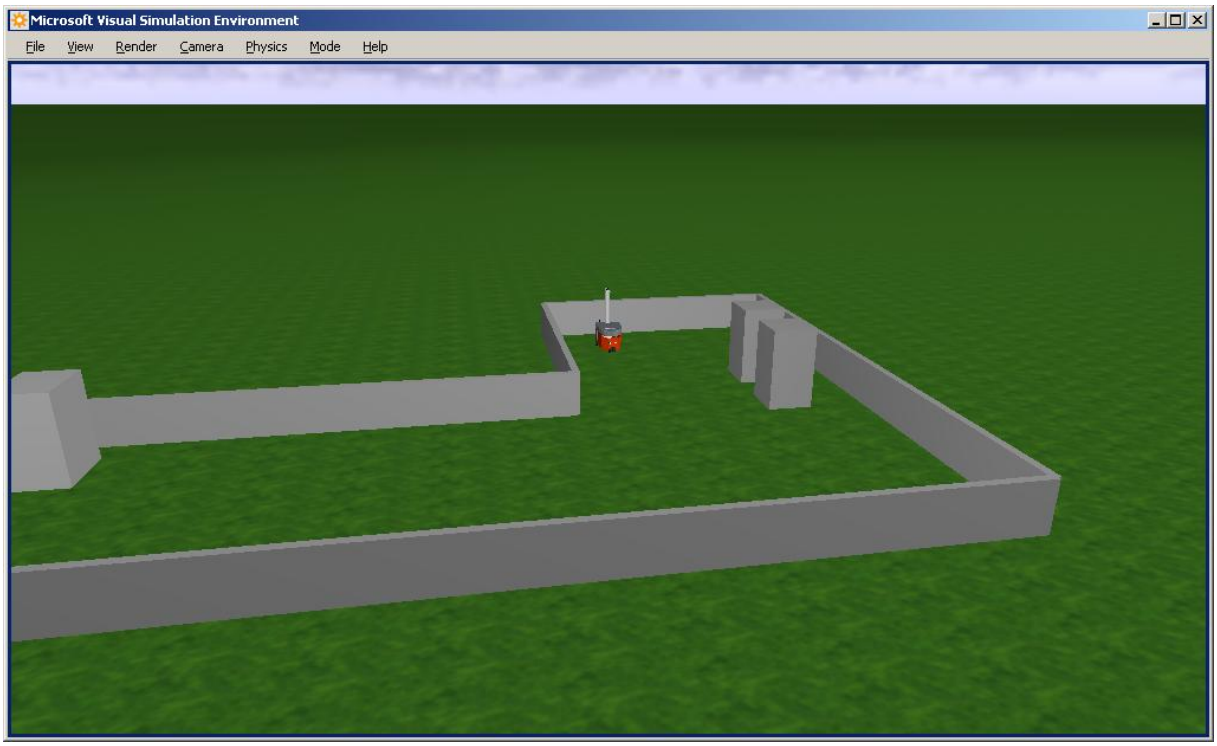


Figure 4.8: Screenshot of the simulation environment

This architecture proportioned a very flexible and satisfactory environment to develop and deploy applications like the one described at chapter 6.

4.4: Advantages for simulation

The aforementioned architecture together with the SOA and MRDS technologies enable exchange between simulation and real operation without any rework nor code compilation for the higher level services.

Looking at the lower row of contracts on figure 4.7, it can be noted that if services obeying those contracts are implemented for simulated environments, all it takes to perform a simulation is to lunch these services instead of the ones that deal with the "real" hardware and the *services directory* will take care of linking all the higher level services to this simulated ones. Figure 4.8 shows a *screenshot* from the robot inside MRDS's simulation environment.

Chapter 5: Services Implementations

The robotics software developed during this work provides a good high level environment for creating mobile robotics solutions in various different niches. Most imaginable mobile robotics solutions must a combination of perform localization, mapping, pose control, trajectory following, feature detection and recognition and task management.

In this chapter, a brief description of the implementations of the developed services is presented. The complete software solution utilizes the C#, .NET and MRDS technologies.

5.1: Feature Extraction

As discussed in section 2.4.4.3 from Chapter 2, feature extractions services generates sets of higher abstraction level information from raw sensor readings in order to ease the implementation and lower the computational complexity of the localization system. Since the main feedback sensor used is a laser range finder able to detect reflecting surfaces, only two types of *feature extraction* services were implemented: `ReflectorExtraction` and `LineExtraction`.

5.1.1: ReflectorExtraction

One of the features easily detected by some kinds of Laser Range Finders are reflector objects. To take advantage of that capability, cylinders covered with some kind of reflector paint or sticker can be deployed to represent a *punctual* landmark. Figure 5.1.1 shows the objects used in this work.

As mandated by the *Feature Extraction* contract, the extracted features are represented by the estimated position of their centers on the global reference frame.

Extracting this features consists in finding adjacent clusters of reflecting echoes in the data from the laser range finder, validate this cluster against the predicted response of the used artificial reflectors and then estimating their centers.

The LRF output data consists in a array of *echoes* objects that carry a distance

(a) Cylinders with reflecting surface (b) Photo taken with *flash* to emphasize the reflecting surface

Figure 5.1: Landmarks for detected by the ReflectorExtraction

measurement and a boolean flag that distinguishes reflecting from non-reflecting surfaces. Finding the cluster of adjacent reflecting echoes is achieved by a trivial iteration through the data. Validation, on the other hand, consists in confronting the cluster's size against a rough prediction computed from equation 5.1.

$$N_{points} = \frac{L_{marker}}{\rho R_{laser}} \tag{5.1}$$

Where L_{marker} is the reflector marker's diameter, R_{laser} is the LRF's resolution (rad) and ρ is the distance reported by the echo in the middle of the cluster. If any big discrepancy is detected, the reflector is discarded. Obviously this validation is not perfect and it is expected that noisy measurements that pass trough this check will be rejected by the Feature Matching process (see section 2.4.4.4).

5.1.2: LineExtraction

This service, also obeying the *Feature Extraction* contract, generates line abstraction objects from LRF data. Line extraction, or line fitting, alone is a wide research area and produced many different approaches to this problem. One popular approach to this problem usesthe Hough transform [12]; this technique is well known and pro-

duces very good results but: a) does not take into account the nature of the LRF sensor; b) produces infinite lines instead of segments and c) is less conservative and thus produces significantly more false positives. Other approaches can be found at [3] and [5].

The implemented algorithm uses different parts of the approaches found in the literature, is implemented in a multi-step manner and in fact extracts line-segments (i.e. with defined start and end points). One interesting assumption adopted is that the input points are ordinated according to their angle to the sensor and that no two points lay in that same angle. This assumption holds true specially for this kind of sensor since the index of an *echo* in the data from the LRF maps directly to a measurement angle (confirming ordination) and since only the closest object in a given direction from the sensor is detected; the further ones are occluded by this first and thus not detected.

One useful consequence of that assumption is that for any two points that were to be adjacent in the same line segment on the *optimal* (i.e. desired) output, they are also adjacent on the LRF data.

Conceptually, the implemented solution can be divided into:

1. Separate LRF data into “connected” groups by detecting break points (see section 5.1.2.1)
2. Sequentially divide each group into a large number of small groups of N_{min} points and computes the best fitting line for that group with the traditional least squares approach (see section 5.1.2.2)
3. Recursively merge adjacent lines that are close in the lines space and that exhibit small dispersion

It is easy to note that this algorithm is probabilistic since it starts with a *pseudo-random* sample and goes refining it until a stop. It relies in the assumption that some of the *pseudo-randomly* generated small segments are in fact good and part of a bigger line. Although not deterministic and with predictable pitfalls, this approach still produces very good and fast results in structured environments, which are the ones that one would be looking for lines in the first place.

This approach is also very conservative and rarely produces false positives, property also desired for example for localization purposes. Following, notable parts of

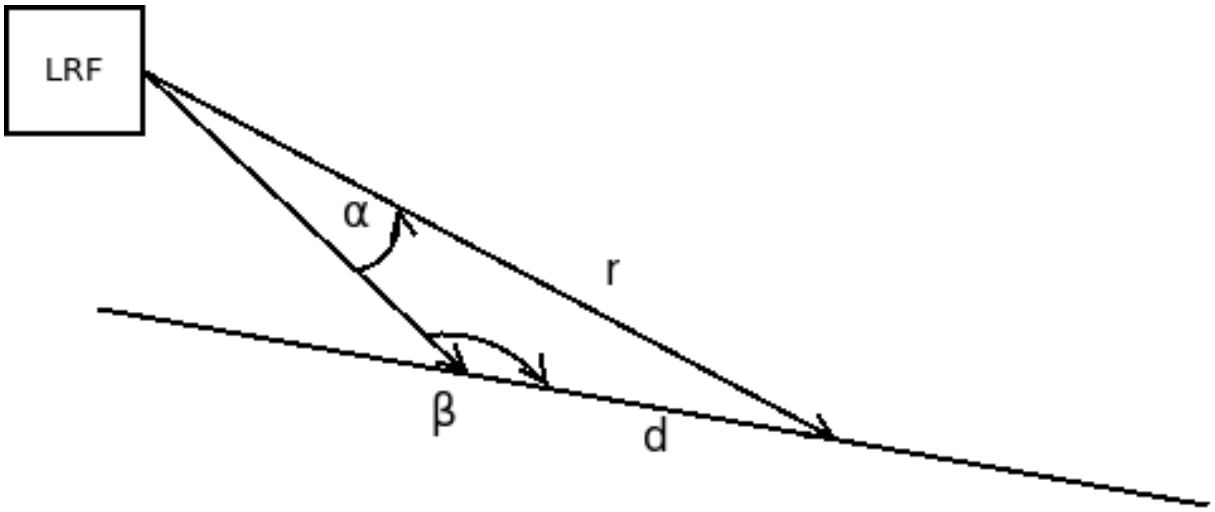


Figure 5.2: Maximum Distance Computation

the algorithm are discussed, a more detailed version and a short display of its performance are presented.

5.1.2.1: Break Point Detection

In order to identify adjacent echoes that are *too far* from each other to be on a line, one could compute a maximum value to be used as a threshold. The problem for computing this value is that, theoretically it does not exist. If the angle β between the measurement laser beam and the "wall" it is measuring grows near π, the distance d between the two adjacent echoes grows to infinity. One partial solution is to assume that there is a *limit* for β. With this limit and given the LRF resolution α and the echo distance r one can use the law of sines (equation 5.2) to compute a reasonable maximum d. Equation 5.4 shows this result.

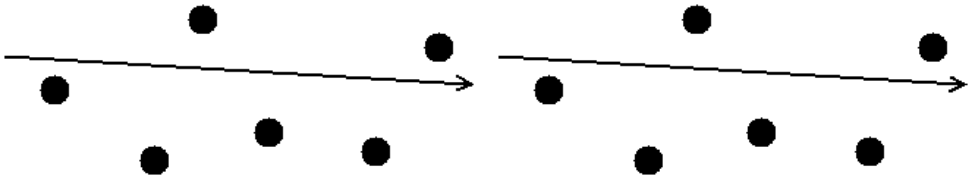
$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} \tag{5.2}$$

$$d_{max} = \frac{r \sin \alpha}{\sin \beta} \tag{5.3}$$

Which for $\alpha = 0.5°$ and $\beta = 175°$ is:

$$d_{max} = r0.10012576 \tag{5.4}$$

(a) With $N_{min} = 3$ lines become biased by the echoes noise



(b) With $N_{min} = 5$ the noise tends to be attenuated by the least squares line fitting

Figure 5.3: Noise immunity effects for different N_{min}

5.1.2.2: N_{min} Rationale

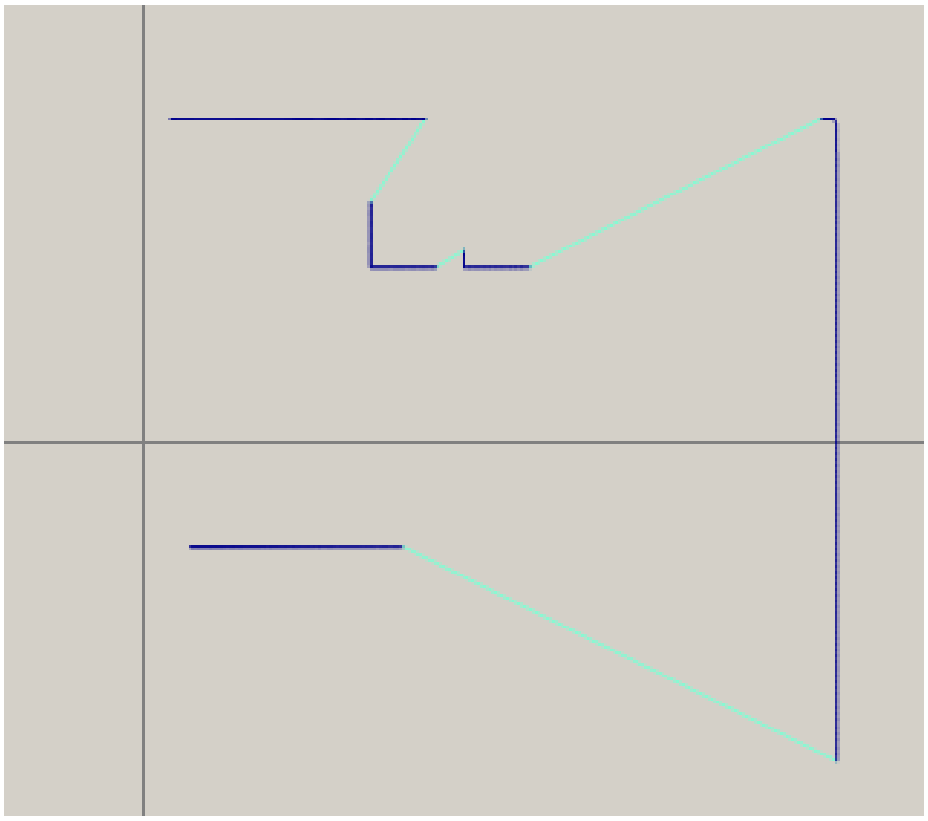
The choice of this parameter is influential to the algorithm's noise immunity, execution speed and output quality. In short, for smaller N_{min} the algorithm becomes less noise immune and slower but tends to produce more "complete" lines; for bigger N_{min} it becomes more robust, faster but produces many "incomplete" lines. Figure 5.1.2.2 shows the resulting small lines for different values of N_{min} for the same set of points that should relate to a single horizontal line but with noisy measurements. In our implementation, $N_{min} = 5$.

5.1.2.3: Detailed algorithm

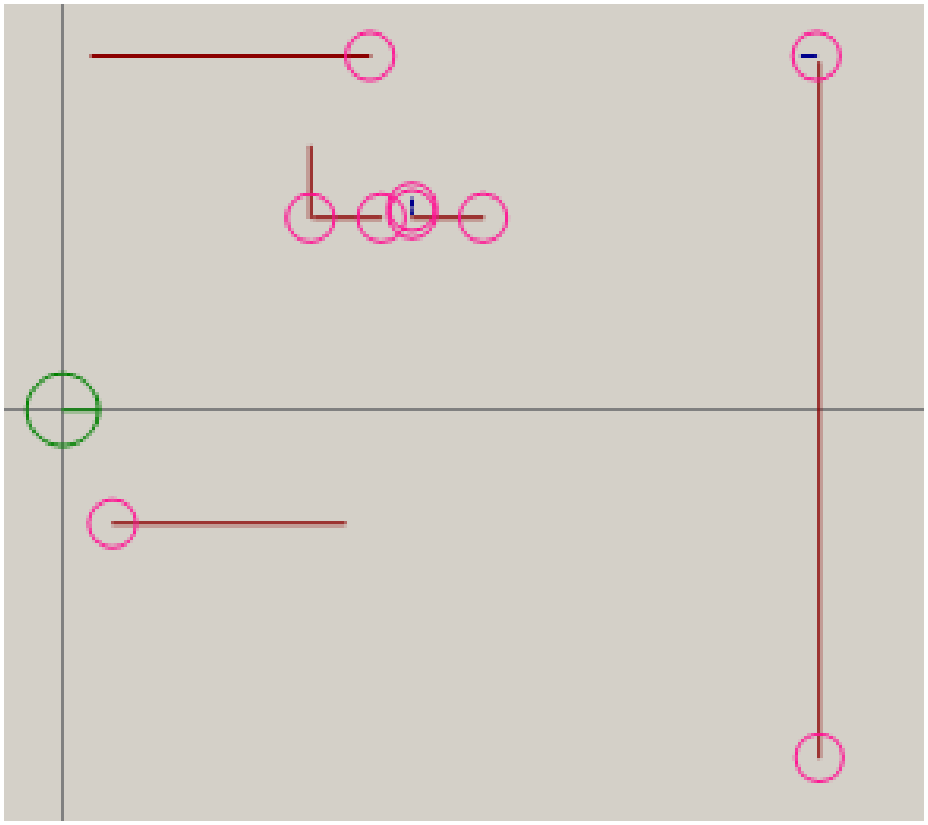
Algorithm 1 shows in depth the iterations and function calls performed by that algorithm.

5.1.2.4: Performance and Preliminary Results

Figures 5.4 and 5.5 show the output of this service and its algorithm. The first shows the output for a simulated environment and intermediary results after the break point detection step. The last shows the output for a real environment, in this case one of the corridors of the DIIGA department at UNIVPM, from three different positions.



(a) Green lines denote connections between two adjacent echoes in a break point



(b) Small circles denote the start points of a extracted line feature and blue lines denote lines that did not merged to any other

Figure 5.4: LineExtraction intermediary and final output for a simulated environment

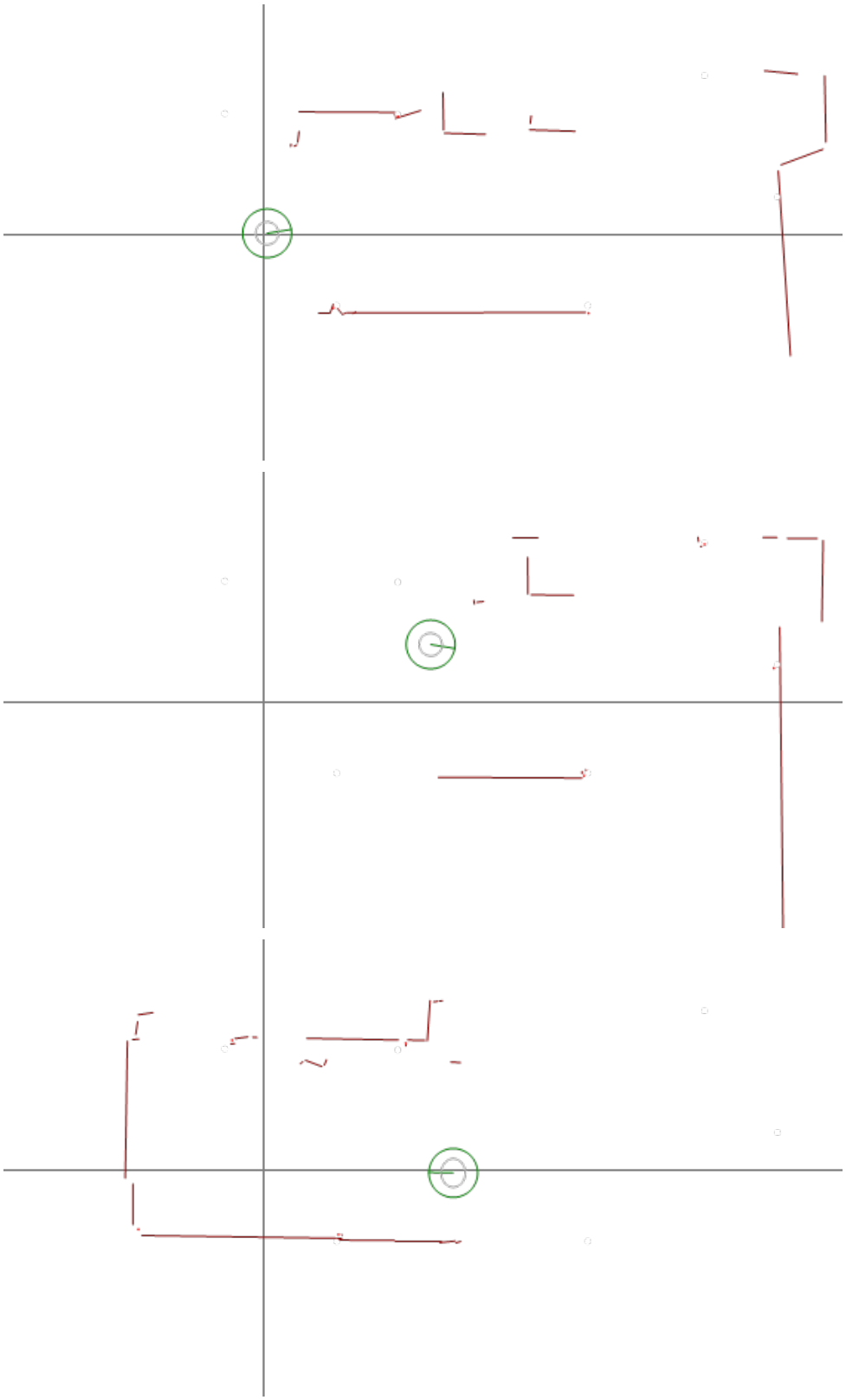


Figure 5.5: LineExtraction final output on a real environment

Algorithm 1 Line Feature Extraction

```
LineExtraction(rawData)

 //Split step
 clusters = split_by_break_point(rawData)
 clusters = split_by_N_min(clusters)
 lines = least_square_lines(clusters)

 //Merge step
 lineAdded = 1
 while lineAdded > 0
  lineAdded = 0
  for line1,line2 in adjacent(lines)
   candidate = merge_lines(line1,line2)
   if dispersion(candidate) < threshold
    lines.remove(line1,line2)
    lines.add(candidate)
    lineAdded++
 lines = remove_too_small(lines)
 return lines
```

5.2: Localization

The localization is maybe the most complex subsystem in this architecture; its theoretical background and justification rely on some relatively sophisticated control theory but its implementation, on the other hand, is quite simple. Given the proposed architecture (i.e. feature abstraction layer and decoupled prediction and correction) the majority of this implementation, including its inner details, follow what was extensively discussed at section 2.4.4.

5.2.1: Prediction Step

The `ProbOdo2Loc` (which stands for Probabilistic Odometry to Localization) service implements the *Prediction Step* contract. In a nutshell it holds: the robot belief (which include pose and associated covariance matrix), updates that belief at every message from the odometry (applying equations 2.23 and 2.24 with the matrices presented in section 2.4.4.2) and provides an interface for belief correction to be used by the *Correction Step* service.

5.2.2: Correction Step

The `EKFLocalization` service implements the *Correction Step* contract and in addition to the normal operations described on section 2.4.4, it offers a simple mapping capability and support for loading and saving the current map belief to non-volatile files. This is very useful in practical applications for fast deployment.

When deploying the system in an environment with static landmarks but which positions are not known, the capability to save the map to a file is essential; the whole "setup" process of deploying such system in a different environment consists in driving the robot around with its mapping capability enabled, then saving the map to a file for permanent use.

Figures 2.10 and 2.11 were produced with this implementation.

The mapping capability was implemented in this service. It was not described previously due to its *ad hoc* nature. Since this capability was not one of the main requisites for the first application, little effort was put on it. Mapping occurs when an extracted feature does not match with any in the map (and is not dubiously *close* to another mapped feature); this newly mapped features carry a low credibility that is increased from subsequent detections. This *ad hoc* solution is explained in figure 5.6.

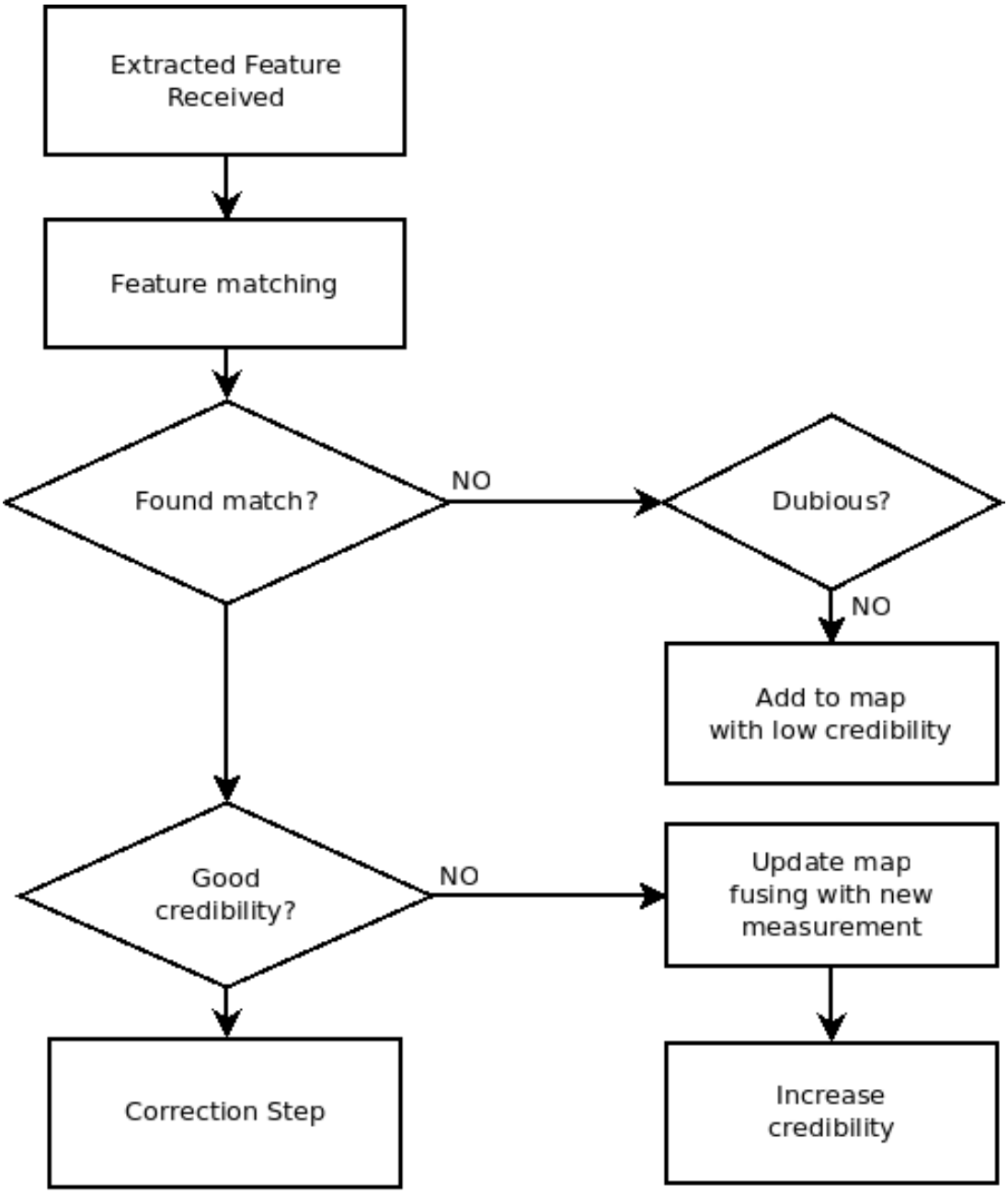


Figure 5.6: Implemented *ad hoc* mapping system

5.3: PoseControl

Pose Control is achieved by the `PoseControl` service which implements the controller described in chapter 3 together with some *ad hoc* additions. For example, it is common to assume a *stop condition* for this kind of controller so that the robot stops when it is in the boundaries of the target pose. Given the error representation (from

chapter 3):

$$\begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} \tag{5.5}$$

a reasonable stop condition is:

$$\rho < e_\rho \tag{5.6}$$

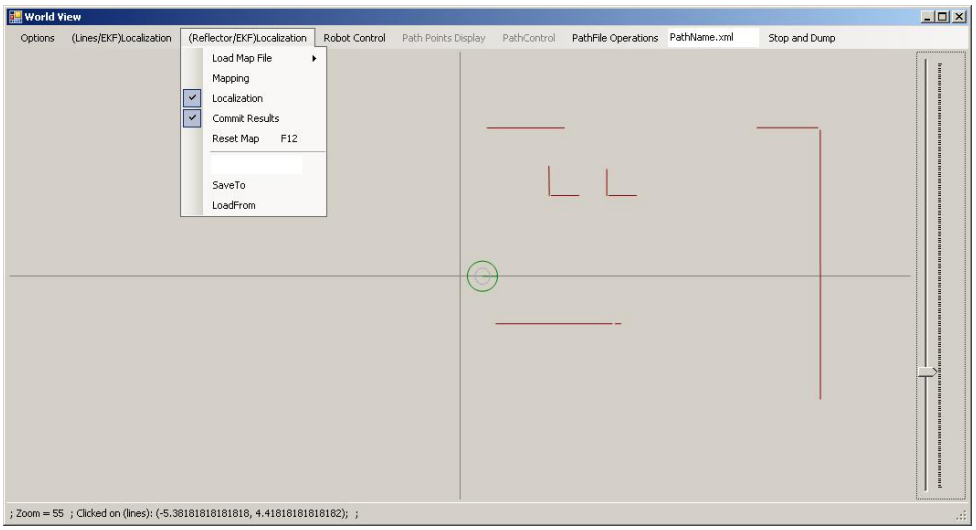
In our application, typical values of e_θ range from $0.02m$ for accurate positioning to $0.14m$ in mid-path points. Another ad hoc addition is a heading controller since in some applications, accuracy in the robot's heading angle is more important than the position itself. In this cases, the robot uses the pose controller until the stop condition 5.7 is achieved and then switches to a simple proportional controller that spins the robot until:

$$d(\theta_{ref}, \theta_i) < e_\theta \tag{5.7}$$

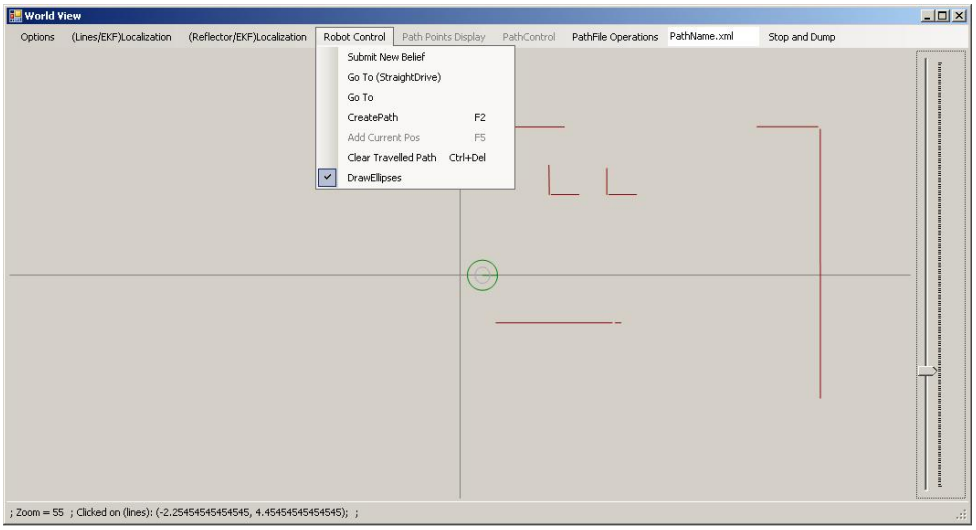
with e_θ ranging from $0.08rad$ to $0.015rad$ where $d(\omega_1, \omega_2)$ is the nonlinear function that gives the smallest real distance between any two angles. Section 6.6 presents the performance of this system in some experiments for that application.

5.4: Supervision and Development

In order to ease development, facilitate tests and provide results visualization, a high-level service `CentralMapGUI` was implemented to provide a development-oriented Graphic User Interface (GUI) for the system as a whole. This GUI provides visualization of the robot's pose and map belief together with associated uncertainties, current sensor measurements and the current path(s) being followed by the robot. Complete control and configuration of the localization services are possible through user friendly menus. Figure 5.7 gives a general idea of the user interface and its controls.



(a) Control menu for the localization service relating to the use of reflectors



(b) Menu with robot control operations useful for debug and some drawing control

Figure 5.7: Screenshots from the service's GUI and some of its menus

5.5: Network Interface and Mission Management

On top of all the other services, `FSMTaskManager` implements the *Coordination and Mission Management* contract. This service is responsible for all the switching in between services and for providing a rich network interface for the whole system.

Aiming at the integration of the current system with existent or legacy (i.e. costumer) systems, the network interface was implemented so that the robot stands as a *web service* for the network. This interface provides general information about the robot's position belief, current state (idle, busy), current task, etc. In terms of operation requests, the *user* software can send: a) list of tasks, which will be executed

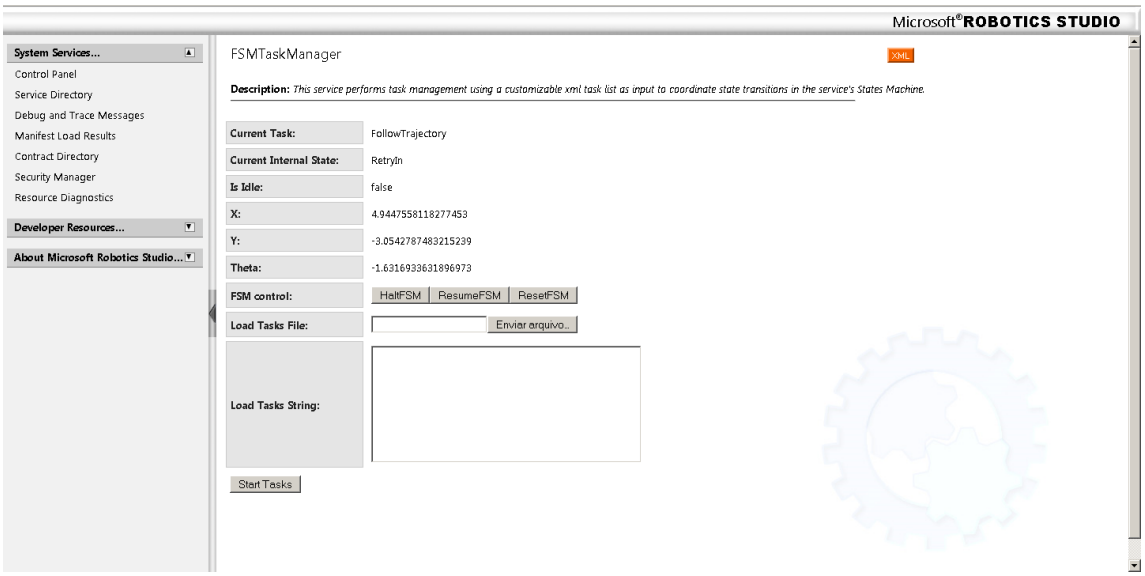


Figure 5.8: Web interface screenshot

by the robot in a FIFO manner or b) high priority tasks for operation's control (e.g Halt,Resume, Reset).

The provided information can be queried by executing a HTTP GET operation on the service's endpoint, namely `http://[robot_ip]:[robot_node_port]/fsmtaskmanager/raw` and to enable human operators to easily get information from the system, a web (*HTML*) formatted interface is also available at `http://[robot_ip]:[robot_node_port]/fsmtaskmanager`. Figure 5.8 is a screenshot of this web interface.

Task requests are sent through HTTP POST operations to the `http://[robot_ip]:[robot_node_port]/fsmtaskmanager` endpoint with a list of tasks encoded in XML.

Listing 5.1: An XML task that determines three waypoints the robot should follow

```
1 <FollowLdcVxList>
2   <Vertex X="5.45" Y="-19.0" Theta="-1.57079" />
3   <Vertex X="5.45" Y="-19.0" Theta="1.57079" />
4   <Vertex X="5.45" Y="-10.0" Theta="1.57079" />
5 </FollowLdcVxList>
```

More examples of these XML messages are presented in Annex A.

In terms of Information Technology, this capability is very desirable in such a system and already produced many results. For the application presented on chapter 6, the partner company uses a LabVIEW® application to handle the specific measurement equipment to which the tested Washing Machine is subjected to and to provide

a Human Machine Interface (HMI) for a central operator; integrating that application to the robot's system was accomplished within less than a week. This event showed the vast business advantage of using such rich interfaces for this kind of application. Figure 5.9 shows the *Use Cases* diagram for the mission management system which in some ways can be seen as the use case diagram for the robot itself. Additionally,

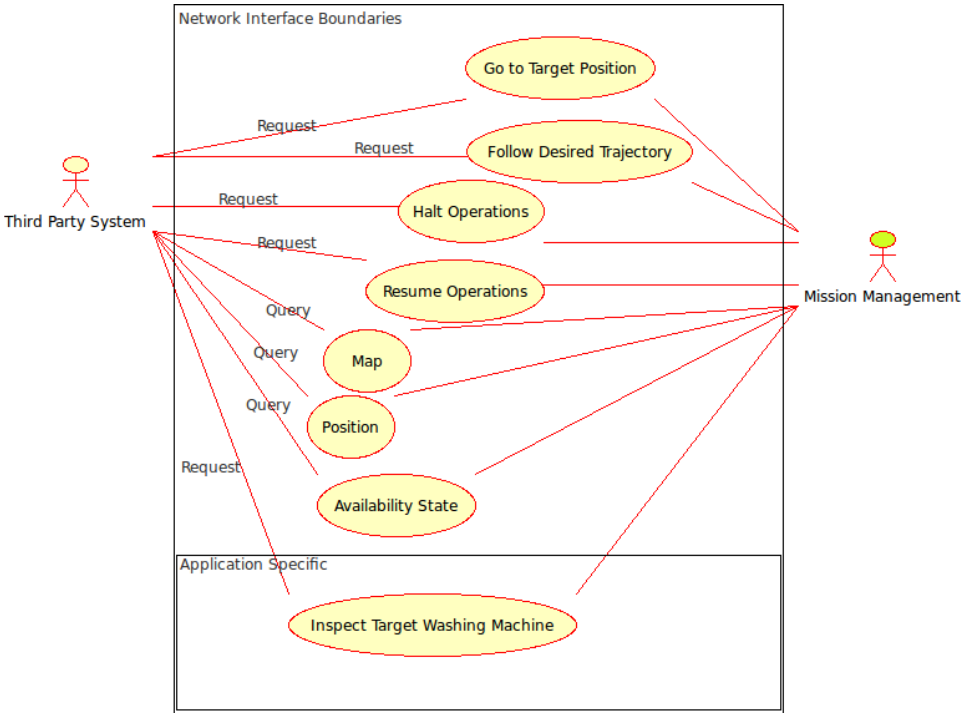


Figure 5.9: Use Case diagram for Mission Management with an application specific task

support for application specific tasks can easily be added as shown in chapter 6.

5.6: Adaptations for Real-time operation

It is quite obvious that real-time operation is crucial to the satisfactory performance of this kind of robots. Related works (see e.g. [25], [11])in this area generally assume that, in order to enable real-time operation, a complete iteration of the localization (and mapping if applicable) algorithm must be completed before it's deadline, which generally is the system's period itself, so that at each time step the robot produces a new state estimation stemming from observations carried inside that time step. Such approach poses great constraints to both, by the same reason, state estimation frequency and computational complexity of the localization algorithm.

Our work in this area differs in the way that we assume that those deadlines

won't be met; this assumption enables the *Prediction* service to run in a much higher rate than the *Correction* one. The problem that arises from this assumption is that newer state estimations (from the Correction step) are delayed by δ_t (which in our experiments could easily reach $1s$) and thus simply replacing the current state by them would generate brutal localization and mapping errors.

To solve such problem we assume that each sensor reading has an accurate *time-stamp*, assigned as early as possible, and that either all the services share the same clock source or all clocks are properly synchronized. Possessing such information, we ensure that the *Prediction* step is computed in real time at frequency f_{odo}, which is reasonable due to the very low computational cost of its operations, and that a circular buffer retains the last K received odometric increments. In this way, when a new pose belief is generated, all odometry increments newer (in relation to its time-stamp) than the new belief are used on N successive *simulated* prediction steps[1].

$$N = \delta_t f_{odo} \tag{5.8}$$

This newer pose belief is then used to replace the current one. The operation is carried if $N < K$, otherwise the update is discarded for being "exceedingly old". This approach is similar to the ones described in [20] and [24] but applied to the EKF and mobile robotics.

Note also that circular buffers are used in each *Feature Extraction* service for both pose beliefs and sensors' readings, each with its time-stamp. An extracted feature is only transfered to the global reference frame when there is a pose belief with matching timestamp is available. This match will only occur after some time thus adding more delay. Figure 5.10 illustrates this aspect.

Figure 5.11 shows the performance of the `LineExtraction` service when the robot is spinning with constant velocity (+$1rad/s$). For figure 5.11a the current (old) localization belief was used to transform the detected line from the robot's reference frame to the global one; figure 5.11b shows the same situation but using a localization belief with matching time-stamp. The consequences of using an old position belief for the feature extraction can be noted on Figure 5.11a where the extracted features are drifted from real feature. The same behavior is present when extracting punctual landmarks (e.g. Reflector Beacons). Figure 5.12 shows a conceptual diagram of our solution.

[1] This process is also called refiltering

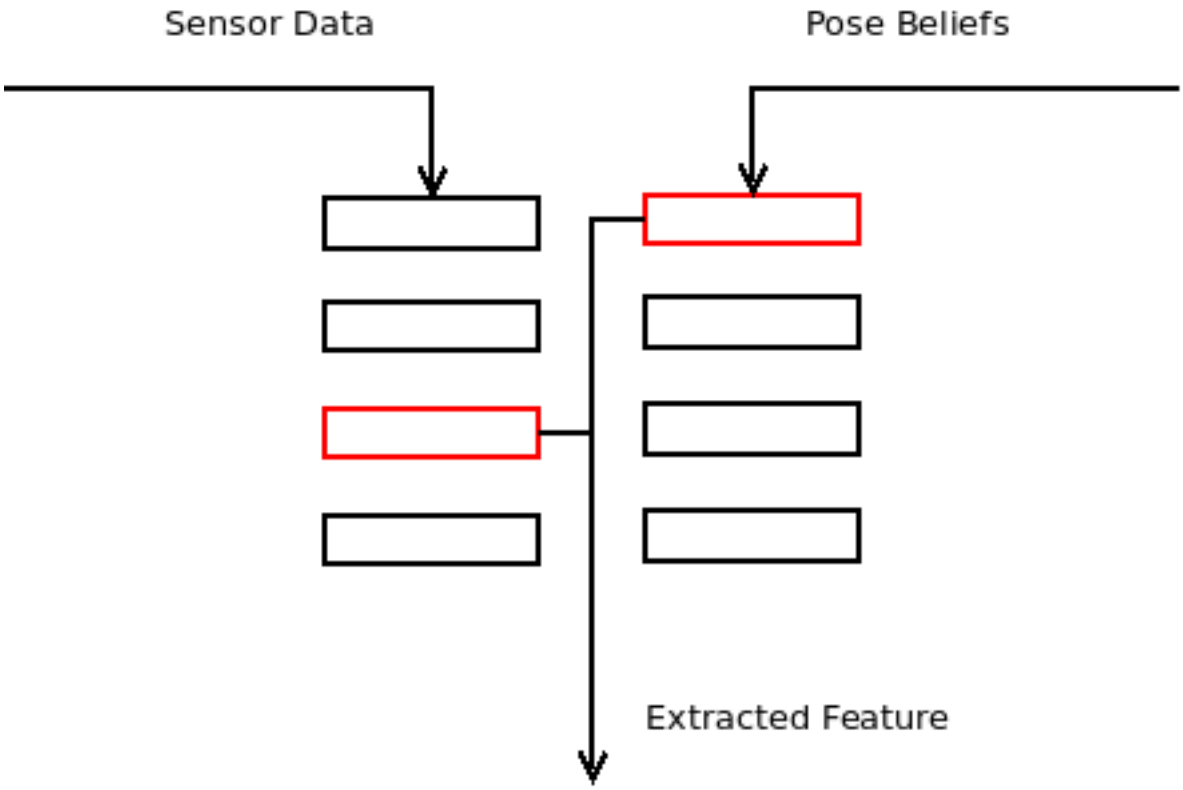
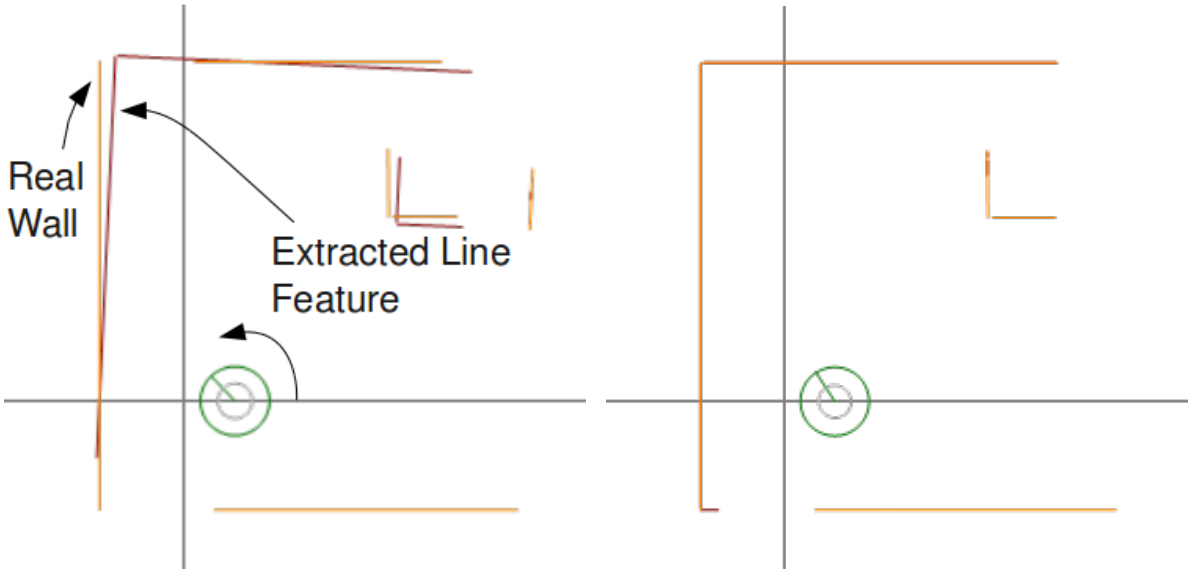


Figure 5.10: Time-stamp matching at feature extraction



(a) Erroneous Transformation

(b) Correct Transformation

Figure 5.11: Line Feature Extraction while spinning at $+1 rad/s$; orange lines represent *correct* walls from a previously generated map.

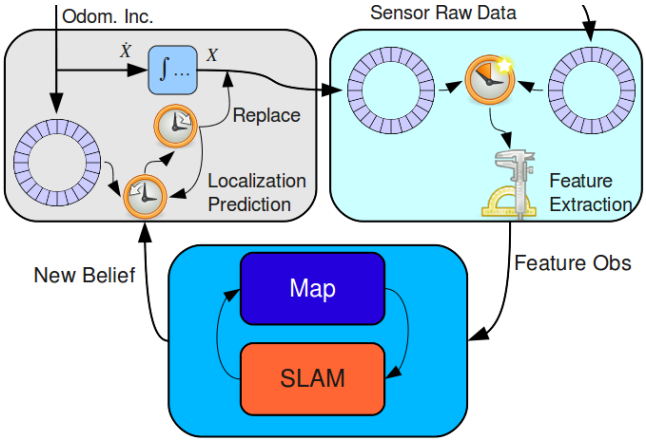


Figure 5.12: Deadline Extension Mechanism

From a control point of view, the stability of the localization algorithm depends on the ratio of uncertainty growth (due to successive predictions) to uncertainty shrinking (due to corrections) which must assure a bounded uncertainty for permanent regime states (constant or null robot velocity). In the case where no movement is carried, the uncertainty tends to decrease beyond reasonable values; to avoid this, an artificial lower bound is used. Note that such lower bound is imposed to the covariance matrix itself after the correction step and thus not affecting eventual accuracy increase. As for constant velocity movement, the uncertainty tends to grow until a stable limit depending on velocity itself as long as N is sufficient.

Table 5.1: Temporal Aspects

Sensor	Period(ms)	Latency(ms)	Computation Time(ms)
Odometry	50 ± 15	< 2	< 5
Line Features (LRF)	500 ± 50	< 25	150
Reflector Beacon Features (LRF)	500 ± 50	< 25	10
EKF Prediction	-	< 1	< 5
EKF Correction	-	< 1	90

Table 5.1 shows many timing aspects of the system. Tasks related to data acquisition and pre-processing have well known periods, latency and computation time. Localization related tasks (Prediction & Correction), on the other hand, are triggered by notifications from the *Feature Extraction* services. Note that due to the higher rate of *Odometry* readings, in contrast to *Line Features* ones, an average of five prediction steps need to be refiltered after a Localization Correction steming from *Line Features*.

Chapter 6: Specific Application

6.1: Introduction

In the last years the use of instrumented industrial robots to test production line result's quality is increasing. The development of a mobile diagnostic robot in this context satisfies the needs of standards and repeatability of the quality controls and guarantees flexibility according to the product under diagnosis and the environment where the test is executed. The robot can autonomously move in weakly structured environments and reconfigure the measurement instruments placement in relation to the type of product and its position. Given the mobile nature of such instrumented robot, the use of expensive measurement instrumentation is less prohibitive than if used on multiple different static testing stations. The system developed in this work was deployed in one of this applications.

A collaboration project between the mechanical and automation departments of UNIVPM and an external partner was established in 2009 to build one of this inspection robots where the mechanical department would develop the measurement equipment and the automation one would develop the robotics related software. The mobile platform itself would be bought from a commercial provider and should include the necessary mechanical and electronic infrastructures.

This project is now on its final deployment stages and in the following sections we describe the application requisites from the robotics view, the chosen commercial mobile platform, the additional services and functionalities implemented for this specific application and finally a discussion about the performance of the system in this application.

6.2: The Washing Machine Reliability Lab

In a reliability test laboratory for household washing machines (WM), multiple rows of WMs are kept in operation for long periods in order to enhance quality control. Figure 6.1 shows a typical layout of this laboratories. Note that the WMs are confined in multiple narrow corridors mandating accurate navigation. Figure 6.2 shows a realistic experimental setup that gives the reader a better impression of the WM corridors. The

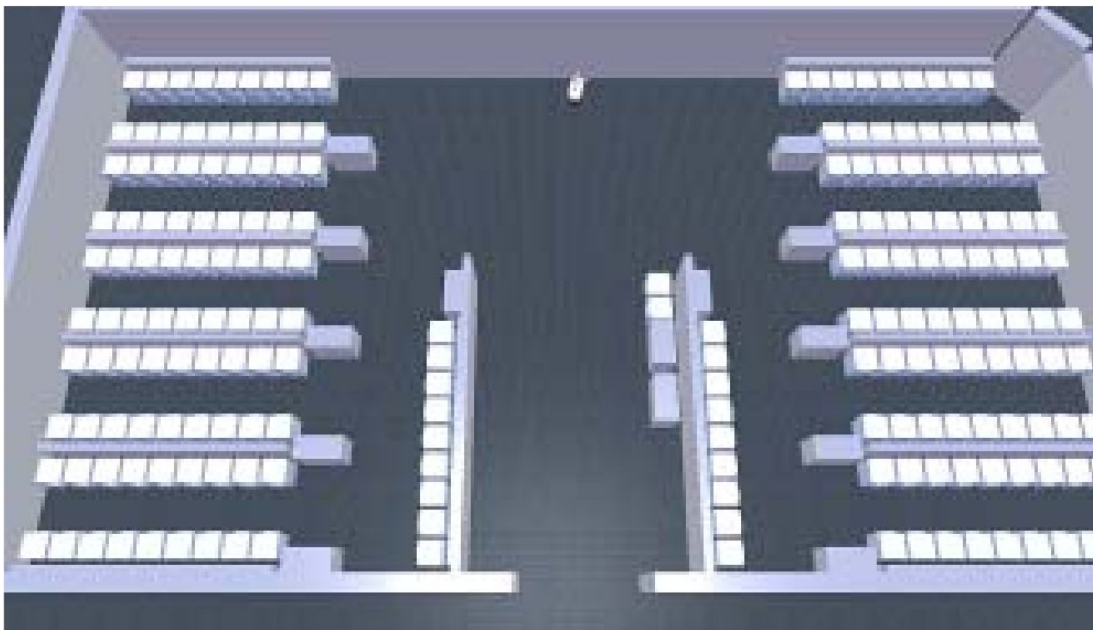


Figure 6.1: Laboratory layout reproduced in a simulation environment

Figure 6.2: Experimental Setup

following brief list summarizes other requisites of the system:

Requisite 1: The robot should repeatably approach a desired washing machine with accurate position and heading ($\theta_{rel} < 0.02$) with regard to the WM's frontal face;

Requisite 2: WMs eventually move from their assigned poses due to constant vibration, clothes loading & unloading, etc;

Requisite 3: The robot should be able to track its position with good accuracy across the whole lab.

6.3: Mobile Platform

The proposed work will take advantage of a *Robulab80* mobile robot base manufactured by Robosoft ®[1]. This platform provides a very flexible and powerful environment for robotics related development and research.

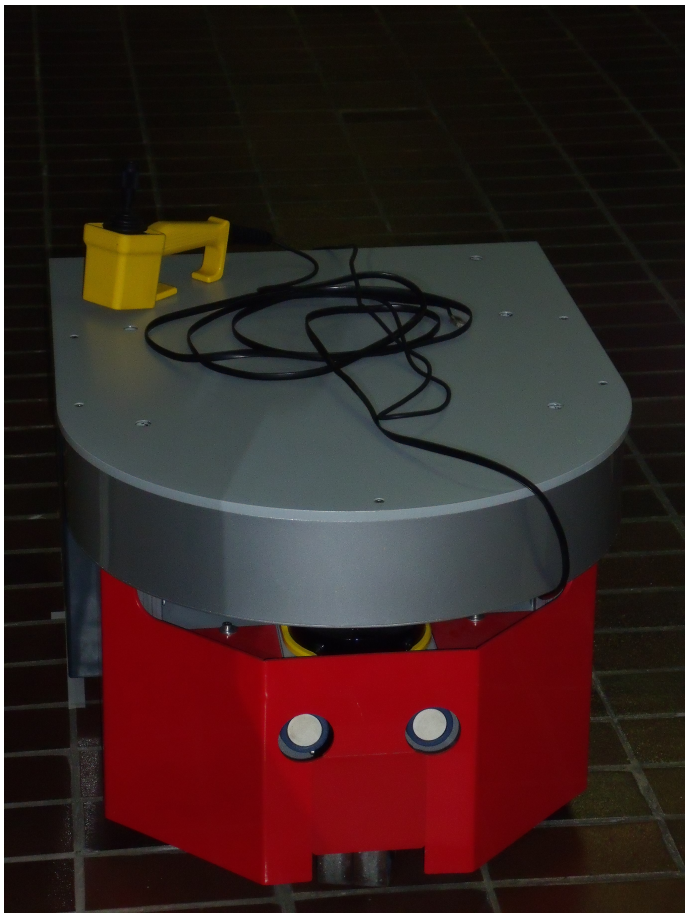


Figure 6.3: Robulab80 Mobile Robot Base

This robot's embedded processing unit is a PC-104 running *Windows XP Embedded*®. For locomotion, a standard differential drive with a Castor wheel setup is used and precision encoders are available in both driving wheels. The robot also employs a SICK S3000 Laser Range Finder and two generic ultrasound single-beam range finders. Table 6.1 shows some electromechanical specifications of the robot.

Table 6.1: Robulab 80 Specifications

Specification	Value	Unit
Maximum Payload	80	Kg
Maximum Wheel Speed	2.6	$\frac{m}{s}$
Autonomy (between charges)	4	h
Top base for extensions	772 x 590 x 475 (L x W x H)	mm

6.4: Additional Service

In a minimalist approach, a single service `WMApproach` was implemented to cope with the application specific aspects of this application. This service is responsible for detecting a WM's face, inferring its center and computing the correct approach pose.

Detecting the WM's face consists in finding a contiguous line (from the LineExtraction service) with the same length of the WM's face in question (typically $60cm$). Inferring the WM's center consists in computing the point behind that line that denotes the virtual center of the WM. This naive approach, in the sense that many other lines may pass this criteria generating false positives, is not a problem in this application since the approximate positions of the WMs are known and thus can be checked against the detected ones; this detection is only necessary to detect small displacements or rotations.

Computing the correct approach pose takes into account three parameters that vary depending on the measurement to be taken. Figure 6.4 illustrates this three parameters together with two more parameters that dictate the WM's dimensions: `WMInstance.Depth` and `WMInstance.FaceSize`. `WMInstance.AppYDist` specifies the distance between the robot's movement axis and the WM's face and `WMInstance.AppXDist` is the displacement between the robot's center (`App.Point`) and the center of the WM's face projected onto the robot's movement axis.

Using those parameters, the service applies the correct translation and rotation

to map the approach point to the global reference frame. After this, the service optionally sends the correct request to the `PoseControl` service or just reports the computed approach position.

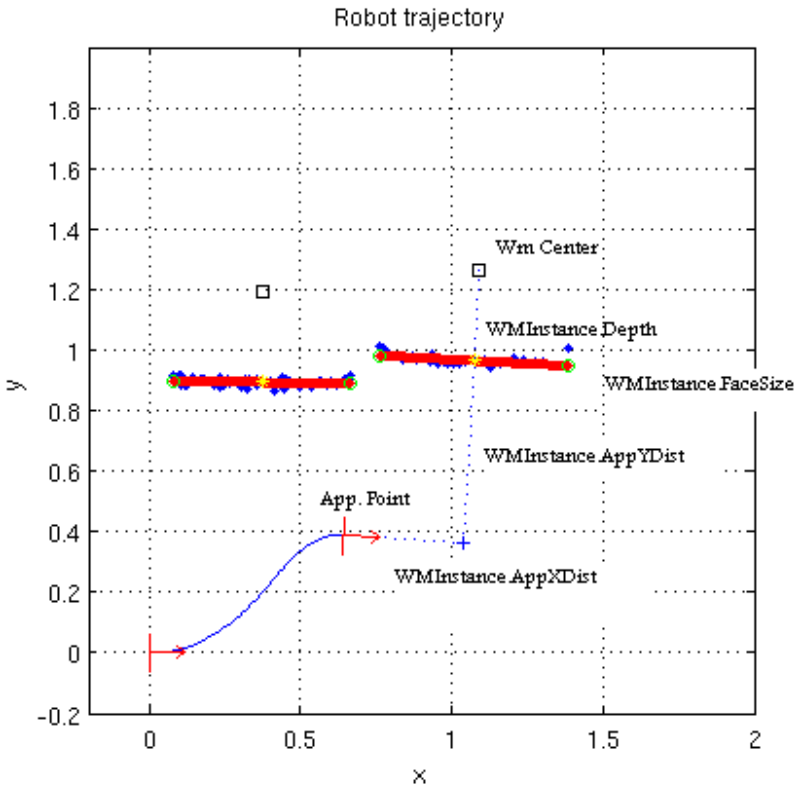


Figure 6.4: Approach Pose (red arrows) Parallel to the Washing Machine's Face (bold red line)

6.5: Additional Tasks at the Mission Management Layer

To support this new capcability, a specific task was added to the `FSMTaskManager` as previously shown in figure 5.9. An example of this task can be seen in listing 6.1.

Listing 6.1: XML task determining a target WM to be approached

```
1  <InspectWM>
2    <WM X="6.1" Y="-8.5" Theta="3.14" dTol="0.8" thTol="0.5"/>
3    <SearchEndPoint X="5.1" Y="-7.5" Theta="-1.57"/>
4  </InspectWM>
```

In line 2, the target WM is specified by its center's pose, being the heading determined by the WM's frontal face and with distance and angle tolerances. Line 3 specifies a

point to which the robot starts driving to while searching for the specified WM; if this point is reached without successful WM detection and approach, the task is considered failure.

Listing 6.2: XML task to redefine the WM's parameters

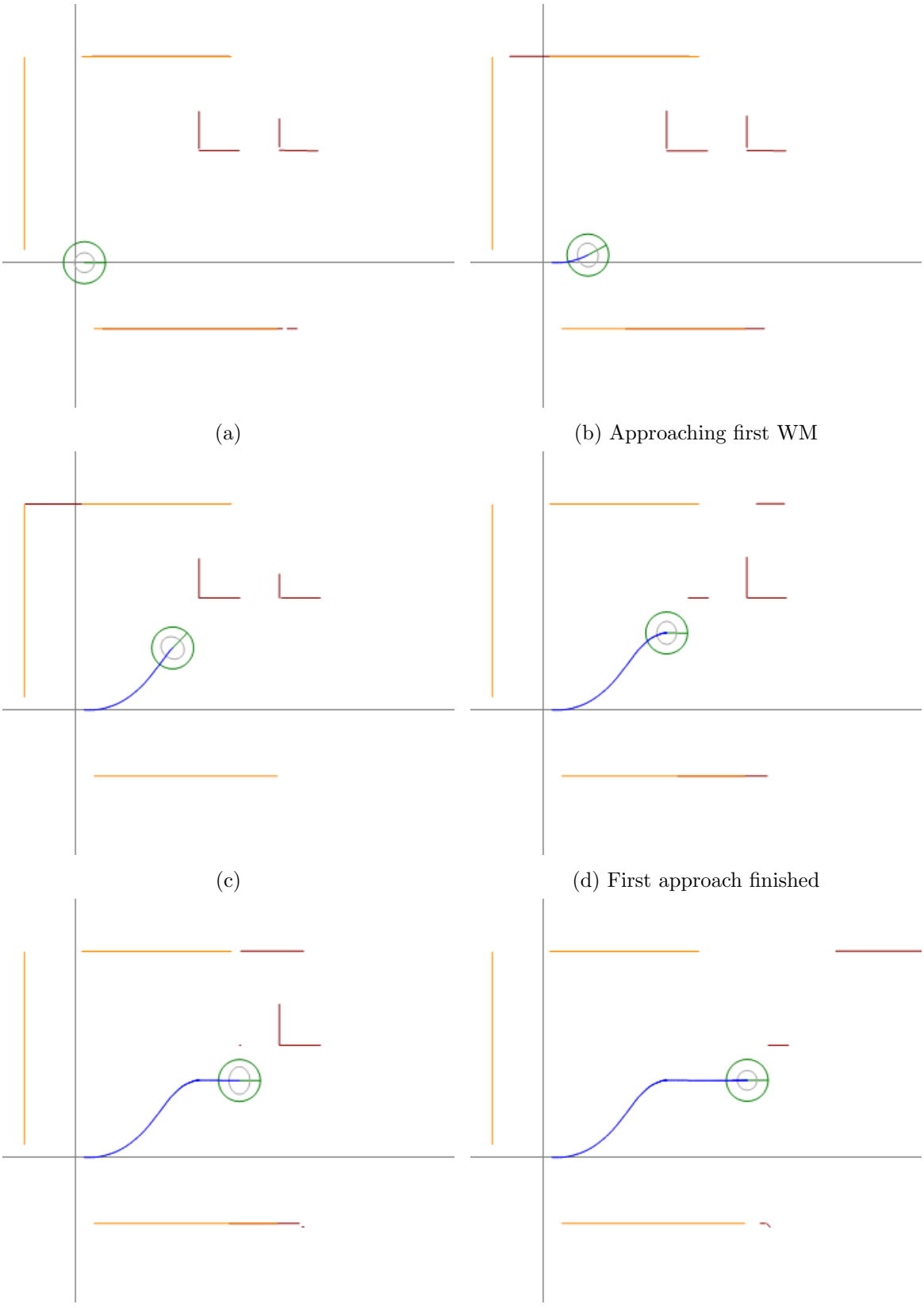
```
1  <SetAppWMParams FaceSize="0.6" Depth="0.3" AppYDist="0.46"
       AppXDist="0.25"/>
```

In listing 6.2, the redefinition of the WM's dimensions and the definition of the approach point are updated. This is very important in an environment where different models of washing machines are present. This parameters are discussed in section 6.4.

6.6: Results

6.6.1: Washing Machine Approach

Figure 6.5 shows the robot approaching a WM from a distant location and then approaching another WM right in front of its current position. Note also the effects of the concurrent localization system working and thus bounding the uncertainty to a small value. This also showcases the operation of the `PoseControl` service that drove the robot from the current position, near $(0, 0)$, to the approach position.

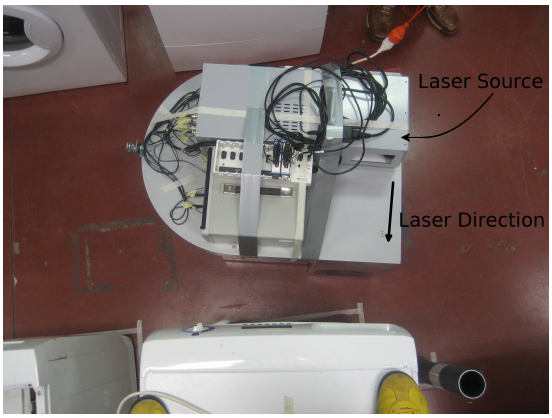


(a) (b) Approaching first WM

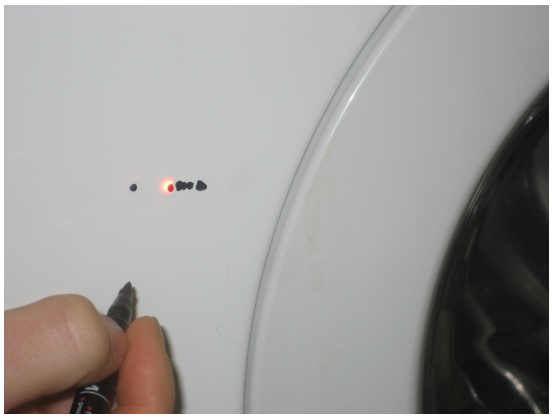
(c) (d) First approach finished

(e) Approaching Second WM (f) Second approach finished

Figure 6.5: Washing Machine's approaches using the `WMApproach` service together with the `PoseControl` one and concurrently with the localization system



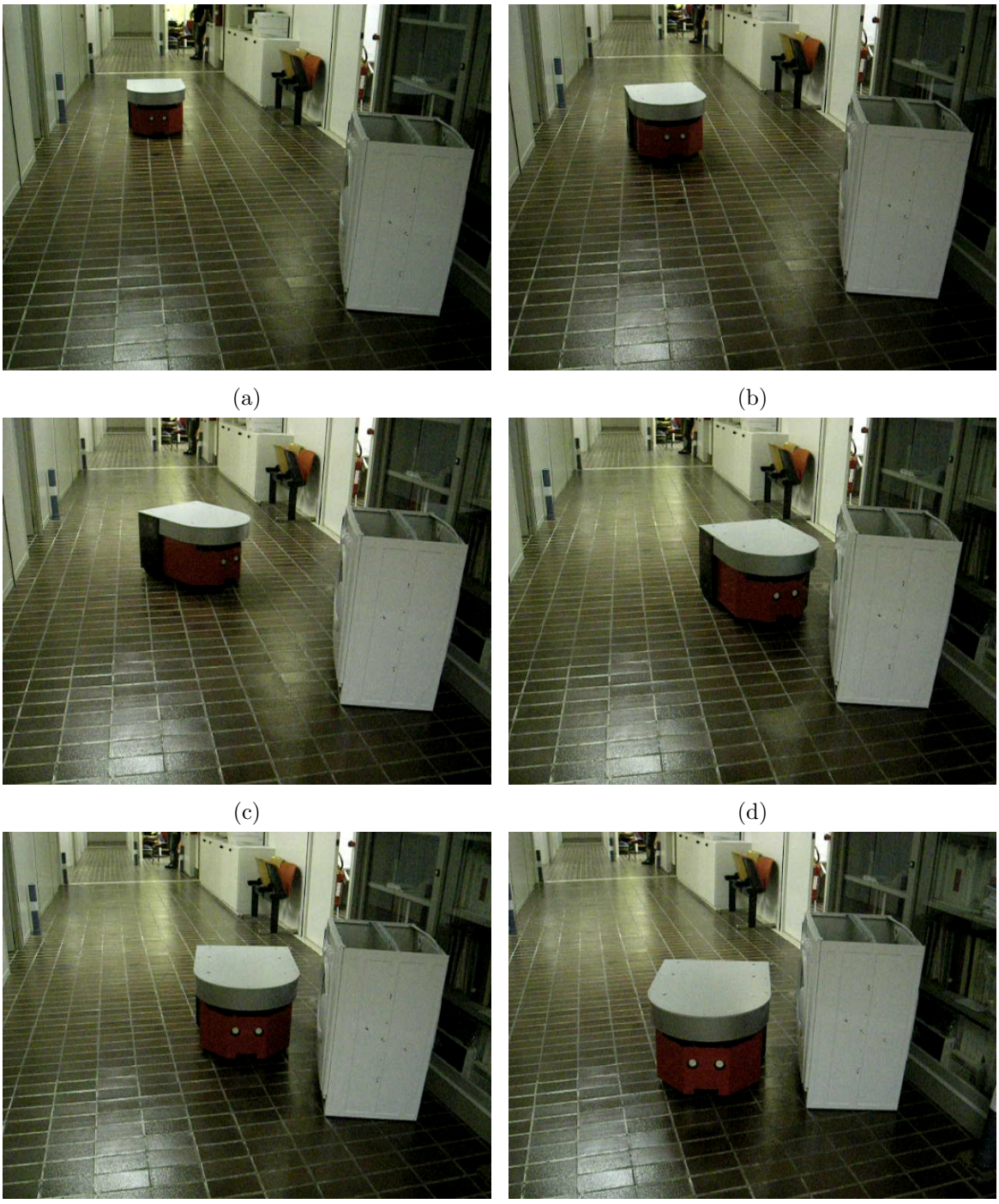
(a) Top view of the setup for the experiment; note the measurement laser placed on top of the robotic base at each's totation.



(b) Results after many approaches with random WM disturbances

Figure 6.6: Approaches Accuracy Experiment

In order to better evaluate the accuracy and repeatability of the heading positioning, which was more relevant to this application, a simple experiment was carried using a laser positioned in the origin of the robot's reference frame and pointing to the left in the perpendicular direction to the robot's movement axis. With this setup, we set the robot to approach the WM many times from different starting positions and with small WM's pose disturbances; then with a magic marker the projection of the laser on the washing machine was registered. The setup for this experiment can be seen at figure 6.6a and the results at figure 6.6b. Figure 6.7 shows frames extracted from a video showing the robot first detecting the WM and inferring its approach position than proceeding to perform such the approach.



(a) (b) (c) (d) (e) (f)

Figure 6.7: Washing Machine approach for a radially drifted WM

6.6.2: Integration with external software and Deployment

As mentioned earlier, this solution was integrated to a LabVIEW® software through a network interface. In this event the advantage of the services-oriented approach was evident. At first, a series of meetings were carried to set up the contracts

between the two systems and to define a first scratch of the tasks representation. A longer development period was then assigned to the implementation of the appropriate application specific service. Finally two more meetings were necessary to formalize the XML interface and to address unpredicted factors in the last meeting. Additionally, the simulation version of the system, which exhibits the same network interface, was given to the other team so that the integration process could be simulated by the other team as early as possible.

Finally, the two systems were put to work together through the network and in a complete test environment (with eight WMs). It took less than two work days to deploy the system in this new environment. This was partly due to the flexibility offered by the system and was deemed satisfactory by both teams.

The deployment procedure at the target environment could be summarized by:

- Establishing the origin of the coordinates reference frame so that the robot starts its mapping task from there; this choice is not important and no accurate positioning of the robot is necessary. In our case this was made to coincide with the WM's plant origin.
- Distributing the reflector beacons described in section 5.1.1 in the environment.
- Driving the robot around (or setting up a set of trajectories to do so) the environement in a slower controlled manner with its mapping capability enabled.
- Saving the generated map to a file for posterior use.

Chapter 7: Conclusion

7.1: Summary

This project focused on the deployment of modern mobile robotics techniques into a final industrial solution. In pursuing that goal we developed some interesting mechanisms that may be of great avail for similar works. The adopted services oriented architecture from chapter 4 seems very promising and it already proved itself very useful in an industrial application, we believe that similar schemes will eventually become an industry standard for robotics applications using this kind of technology.

Although the localization system developed in this work, from section 2.4.4 does not employ the most modern algorithms as outlined by section 2.5, its performance was very satisfactory and it easily met the requisites of the first targeted application (see section 6.6).

Also related to the localization system, another contribution of this work is the simple mechanism used to enable real-time operation presented in section 5.6. This approach was imperative to the successful operation of the final system and some variant of this solution necessarily should be present in commercial mobile robotics products.

In the mission management end, the network interfaces provided by the robot and described in section 5.5 produced very good results and were essential for the application shown in chapter 6. The integration between the robot's software and the outer systems was accomplished in less than a week mostly due to the use of XML messages through the HTTP interface.

7.2: Future Work

During the course of this work, many improvement directions were unfolded. Some of the possible future improvements and standing challenges are:

- The localization system itself offers many improvement directions that could certainly enhance the robustness of the system. Future works shall focus on implementing a parallel service for *pose recovering*.

- The current mapping system is useful but uses a heuristic, credibility based, approach. Although this solution satisfactorily met the requisites for the deployed application, a more formally backed approach is desired for applications where the mapping capability plays a more important role.
- The actual Pose Controller should be enhanced to restrict the control action near the target position since with the current implementation some undesired oscillations may occur when reaching final accurate poses.
- Currently the gains for the Pose Controller are chosen based on the current task since they govern the trajectory followed by the robot but further development should focus on an automatic adaptive approach to enhance obstacle avoidance.
- Future development of an user friendly graphical environment for task list generation.
- Although the system has been exhaustively tested in its final deployment environment, more evaluation of the implemented services in order to produce more tactile and quantitative results is indeed needed.

> Beware of bugs in the above code; I have only proved it correct, not tried it.
>
> *Donald Knuth*

Annex A: XML Messages

In this annex complete XML messages are exposed to give an deeper idea of the network interface described in section 5.5.

Listing A.1 shows a typical response for a HTTP GET request at the services endpoint. This response contains a representation of the current tasks list, some flags that express the operation state of the robot and finally data regarding the actual robot's *estimated* position and velocity at the present moment.

Listing A.1: HTTP GET State Response, Commented

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FsmtaskmanagerState xmlns="http://schemas.tempuri.org/2009/09/
       fsmtaskmanager.html" xmlns:d="http://schemas.microsoft.com/
       xw/2004/10/dssp.html" xmlns:s="http://www.w3.org/2003/05/
       soap-envelope" xmlns:wsa="http://schemas.xmlsoap.org/ws
       /2004/08/addressing">
3   <taskList><!-- shows the tasks still in the execution queue
        -->
4    <Task><description>InspectWM</description></Task>
5    <Task><description>FollowTrajectory</description></Task>
6    <Task><description>FollowTrajectory</description></Task>
7    <Task><description>InspectWM</description></Task>
8    <Task><description>FollowTrajectory</description></Task>
9   </taskList>
10  <currentTaskRef><!-- shows the current task -->
11   <description>InspectWM</description>
12  </currentTaskRef>
13
14  <isHalted>false</isHalted> <!-- tasks execution halted by
        HalfFSM command -->
15  <isIdle>false</isIdle> <!-- tasks list is empty -->
16
17  <!-- collision detection blocking control -->
18  <lastBlock>2010-02-23T12:58:09.800132+01:00</lastBlock>
19  <blocked>false</blocked>
```

```
20
21  <!-- robot's state -->
22
23    <!-- velocities -->
24    <v_ang>0</v_ang>
25    <v_tan>0</v_tan>
26
27    <!-- position -->
28    <rX>5.4966397367243394</rX>
29    <rY>-7.1952432364080208</rY>
30    <rTheta>-1.10764479637146</rTheta>
31
32  </FsmtaskmanagerState>
```

Listing A.2 shows all the available operations supported by the `FSMTaskManager` service.

Listing A.2: Current Supported Tasks, Commented

```
1    <ResetTaskList/>
2    <HaltFSM/>
3    <ResumeFSM/>
4
5    <FollowLdcVxList>
6  <!-- specifies a custom list of points (trajectory) to follow
       -->
7      <Vertex X="5.45" Y="-19.0" Theta="-1.57079" />
8      <Vertex X="5.45" Y="-19.0" Theta="1.57079" />
9      <Vertex X="5.45" Y="-10.0" Theta="1.57079" />
10   </FollowLdcVxList>
11
12   <LocalMove a="0.5" d="2.0"/>
13 <!-- requests a movement relative to the current position; "a"
       stands for angle (radians) and "d" for forward movement -->
14
15   <FollowTrajectory>
16
```

```
17      <Trajectory number="0" file="Trajectories.xml"/>
18  <!-- loads a previously computed trajectory from file -->
19
20      <StopPoint X="5.45" Y="-6.0" Tol="0.5" stopAtPoint="true"/>
21  <!-- specifies a stop point different than the end of the
        trajectory -->
22
23      <ReturnPoint X="5.45" Y="-10.0" Theta="-1.57079"
            resumeAtPoint="true"/>
24  <!-- used to specify a start point different from the first
        point of the trajectory -->
25
26    </FollowTrajectory>
27
28
29    <InspectWM>
30
31      <WM X="6.1" Y="-8.5" Theta="3.14" dTol="0.8" thTol="0.5"/>
32  <!-- specifies a target washing machine (WM). dTol and thTol
        specify the maximum tolerance between specified WM and
        detected -->
33
34      <SearchEndPoint X="5.45" Y="-7.5" Theta="-1.57"/>
35  <!-- Which point to stop searching for the WM; this point is
        only reached if no proper WM is recognized -->
36
37    </InspectWM>
38
39    <SetAppWMParams FaceSize="0.6" Depth="0.3" AppYDist="0.46"
          AppXDist="0.25"/>
40
41    <SetupEKF mapFile="Mapped_5.xml"/>
42  <!-- specifies a file to load environment information -->
43
44    <Sleep delay="3.0"/>
```

And listing A.3 shows a typical task list.

Listing A.3: Typical Task List

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <TaskList>
3    <FollowTrajectory>
4      <Trajectory number="0" file=""/>
5      <StopPoint X="5.1" Y="-6.0" Tol="0.5" stopAtPoint="true"/>
6    </FollowTrajectory>
7
8    <InspectWM>
9      <WM X="6.1" Y="-8.5" Theta="3.14" dTol="0.8" thTol="0.5"/>
10     <SearchEndPoint X="5.1" Y="-7.5" Theta="-1.57"/>
11     <Inspection wait="2000"/>
12   </InspectWM>
13   <FollowTrajectory>
14     <Trajectory number="0" file=""/>
15     <ReturnPoint X="5.1" Y="-9.5" Theta="-1.57079"
           resumeAtPoint="true"/>
16   </FollowTrajectory>
17 </TaskList>
```

Bibliography

[1] Robosoft - advanced robotics solutions, http://www.robosoft.com/.

[2] L. Armesto, G. Ippoliti, S. Longhi, and J. Tornero. Probabilistic self-localization and mapping - an asynchronous multirate approach. Robotics & Automation Magazine, IEEE, 15(2):77–88, June 2008.

[3] L. Armesto and J. Tornero. Robust and efficient mobile robot self-localization using laser scanner and geometrical maps. In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 3080–3085, Oct. 2006.

[4] A. Astolfi. Exponential stabilization of a mobile robot. Proceedings of 3rd European Control Conference, 1995.

[5] G.A. Borges and M.-J. Aldon. A split-and-merge segmentation algorithm for line extraction in 2d range images. In Pattern Recognition, 2000. Proceedings. 15th International Conference on, volume 1, pages 441–444 vol.1, 2000.

[6] Y. Chen and X. Bai. On robotics applications in service-oriented architecture. 2008.

[7] C. Christo and C. Cardeira. Service oriented architecture for mobile robot localization. pages 888 –891, sept. 2007.

[8] Vinícius Menezes de Oliveira, Walter Fetter Lages, and Edson Roberto de Pieri. Mobile robot control using sliding mode and neural network. IFAC Robot Control, 2003.

[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. volume 2, pages 1322 –1328 vol.2, 1999.

[10] Chiara Fulgenzi, Gianluca Ippoliti, and Sauro Longhi. Experimental validation of fastslam algorithm integrated with a linear features based map. IFAC Mechatronics 19, 609-616, 2009.

[11] J. E. Guivant and E. M. Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. Transactions on Robotics and Automation, Vol. 17, No. 3, 2001.

[12] P. V. C. Hough. Method and means for recognizing complex patterns. U.S. Patent 3069654, 1962.

[13] Wei-Han Hung, P. Liu, and Shih-Chung Kang. Service-based simulator for security robot. pages 1 –3, aug. 2008.

[14] T. Kailath. Linear systems. Prentice-Hall Information and System Science Series, 1980.

[15] R. E. Kalman. A new approach to linear filtering and prediction problems. Journal of Basic Engineering 82, 1960.

[16] H. Lang, Y. Wang, and C. W. de Silva. Mobile robot localization and object pose estimation using optical encoder, vision and laser sensors. 2008.

[17] Sung-On Lee, Young-Jo Cho, Myung Hwang-Bo, Bum-Jae You, and Sang-Rok Oh. A stable target-tracking control for unicycle mobile robots. volume 3, pages 1822 –1827 vol.3, 2000.

[18] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. Transactions on Robotics and Automation, 1991.

[19] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on, pages 1442–1447 vol.3, Nov 1991.

[20] Xiao Lua, Huanshui Zhang, Wei Wangb, and Kok-Lay Teo. Kalman filtering for multiple time-delay systems. Automatica, August 2005.

[21] Nardênio A. Martins, Douglas W. Bertol, Edson R. De Pieri, and Eugênio B. Castelan. Control of Mobile Robot Considering Actuator Dynamics with Uncertainties in the Kinematic and Dynamic Models. Springer, 2009.

[22] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. volume 2, pages 1985 – 1991 vol.2, sept. 2003.

[23] P. Newman, J. Leonard, J.D. Tardos, and J. Neira. Explore and return: experimental validation of real-time concurrent mapping and localization. In Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, volume 2, pages 1802–1809 vol.2, 2002.

[24] Sirichai Pornsarayouth and Manop Wongsaisuwan. Sensor fusion of delay and non-delay signal using kalman filter with moving covariance. In Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics, 2009.

[25] D. Schleicher, L. M. Bergasa, M. Ocana, R. Barea, and M. E. Lopez. Real-time hierarchical outdoor slam based on stereovision and gps fusion. Transactions on Intelligent Transportation Systems, Vol. 10, No. 3, 2009.

[26] R. Siegwart and I. Nourbakhsh. Introduction to Autonomous Mobile Robots. MIT Press, 2nd edition, 2004.

[27] S. Thrun, W. Burgard, and D. Fox. Probabilistic Robotics. MIT Press, 2nd edition, 2006.

[28] Jiang Yan, Liu Guorong, Luo Shenghua, and Zhou Lian. A review on localization and mapping algorithm based on extended kalman filtering. In Information Technology and Applications, 2009. IFITA '09. International Forum on, volume 2, pages 435–440, May 2009.