Olga Meshcheryakova

Mestrado em Engenharia Electrotécnica,
Sistemas e Computadores

Probabilistic constraint reasoning with Monte Carlo integration to Robot Localization

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica, Sistemas e Computadores

Orientador: Pedro Alexandre da Costa Sousa,
Professor, FCT-UNL
Co-orientador: Jorge Carlos Ferreira Rodrigues da Cruz,
Professor, FCT-UNL

Júri

Presidente: João Paulo Branquinho Pimentão

Arguente(s): José António Barata de Oliveira
Pedro Alexandre da Costa Sousa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2014

Probabilistic constraint reasoning with Monte Carlo integration to Robot Localization

Acknowledgements

I would like to express my deepest gratitude to Professor Jorge Cruz, who gave me guidance, suggestions and help, which was decisive for the work. I am very grateful to Professor Pedro Sousa, who gave me motivation and provided all kinds of support. Also, I would like to thank Marco Correia, who helped me in the implementation and gave valuable advices.

Moreover, I want to express special thanks to my family and friends who always supported and motivated me.

Resumo

Este trabalho estuda a combinação de raciocínio seguro e probabilístico através da hibridação de técnicas de integração de Monte Carlo com programação por restrições em domínios contínuos. Na programação restrição contínua existem variáveis que vão sobre os domínios contínuos (representados como intervalos), juntamente com as restrições sobre elas (as relações entre as variáveis) e o objetivo é encontrar valores para as variáveis que satisfazem todas as restrições (cenários consistentes). Algoritmos programação por restrições "branch-and-prune" produzem resultados de todos os cenários consistentes. Algoritmos especiais propostos para o raciocínio probabilístico por restrição calculam a probabilidade de conjuntos de cenários consistentes que implicam o cálculo de um integral sobre estes conjuntos (quadratura). Neste trabalho, propomos estender os algoritmos "branch-and-prune" com técnicas de integração de Monte Carlo para calcular essas probabilidades. Esta abordagem pode ser útil na área da robótica para problemas de localização. As abordagens tradicionais são baseadas em técnicas probabilísticas que buscam o cenário mais provável, que não pode satisfazer as restrições do modelo. Nós mostramos como aplicar a nossa abordagem para lidar com este problema e fornecer funcionalidade em tempo real.

Palavras-chave: programação por restrições em domínios contínuos, análise do intervalo, integração de Monte Carlo, Localização de Robots

Abstract

This work studies the combination of safe and probabilistic reasoning through the hybridization of Monte Carlo integration techniques with continuous constraint programming. In continuous constraint programming there are variables ranging over continuous domains (represented as intervals) together with constraints over them (relations between variables) and the goal is to find values for those variables that satisfy all the constraints (consistent scenarios). Constraint programming "branch-and-prune" algorithms produce safe enclosures of all consistent scenarios. Special proposed algorithms for probabilistic constraint reasoning compute the probability of sets of consistent scenarios which imply the calculation of an integral over these sets (quadrature). In this work we propose to extend the "branch-and-prune" algorithms with Monte Carlo integration techniques to compute such probabilities. This approach can be useful in robotics for localization problems. Traditional approaches are based on probabilistic techniques that search the most likely scenario, which may not satisfy the model constraints. We show how to apply our approach in order to cope with this problem and provide functionality in real time.

Keywords: Continuous Constraint Programming, Interval Analysis, Monte Carlo integration, Robot Localization

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CCP - Continuous Constraint Programming

CCSP - Continuous Constraint Satisfaction Problem

CSP – Constraint Satisfaction Problem

CP – Constraint Programming

CPA – Constraint Propagation Algorithm

DC - Direct Current

LADAR - LAser Detection And Ranging

MC – Monte Carlo

MCL - Monte Carlo Localization

MFTP – Multivariable Fuzzy Temporal Profile model

SLAM - Simultaneous Localization and Mapping Methods

p.d.f. - probability density function

PID – Proportional-Integral-Derivative

1

Introduction

A key element in robotics is *uncertainty* that arises from many factors, such as environment, sensors, models, computations and robot actuators and motors. Probabilistic robotics is an approach for dealing with hard robotic problems that relies on probability theory and has a number of developed algorithms and implemented solutions that will be specified further in this dissertation. In the case of the robot localization problem, where sensors play a major role, errors in measurements are unavoidable and must be considered together with the model constraints. This requires a method for uncertainty reasoning with mathematical models involving nonlinear constraints over continuous variables.

1.1 Goals

The general goal of this work is the development of an approach that combines safe and probabilistic reasoning for dealing with the uncertainty in nonlinear constraint models. We propose a new technique that is based on the combination of methods from continuous constraint programming with Monte Carlo integration techniques. We aim to overcome the scalability problem of the pure (safe) constraint programming approach providing the means to quickly obtain an accurate characterization of the uncertainty. We envisage to successfully apply this technique to probabilistic robotics and, in particular, to show its advantages with respect to the traditional approaches for the robot localization problem.

1.2 Area

Uncertainty occurs in stochastic environments and is a subject of study in a number of fields, including physics, economics, statistics, engineering, and information science. In robotics uncertainty arises from many sources.

Physical worlds are unpredictable. In robotics the environment is highly dynamic and unpredictable that leads to the high degree of uncertainty. In some task (such as path planning) the ignorance of uncertainty is not possible.

Sensors have number of limitations that arise from the following factors. The range and resolution of a sensor are limited by physical laws. Image sensors cannot see through walls and, moreover numbers of the parameters characterizing the performance are limited. Another problem is noise, which disturbs the measurements of the sensors. The noise in unpredictable and it limits the information that can be extracted from sensors measurements [1].

Robot actuation involves motors that could be subject to control noise. Some of the actuators are characterized by low noise level. However others cannot provide accurate positioning.

All of these factors give rise to uncertainty. A probabilistic approach considers uncertainty and uses models to abstract useful information from the measurements. The actual sensors always give some scatter of values measured with some accuracy. Errors always occur in the sensor measurements.

Probabilistic robotics deals with the concepts of control and perception in the face of uncertainty, inherent to the location of the robots which are usually in unstructured environments. The key idea is to represent the uncertainty in an explicit way, representing information by probability distributions over all space of possible hypotheses instead of relying only on best estimates. In this way, the models can represent the ambiguity and the degree of confidence in a solid mathematics, allowing them to accommodate all sources of uncertainty.

However one of the limitations of probabilistic algorithms is the computational complexity, because the computation of the exact posterior distributions can be unaffordable. Also those algorithms make approximations, since robots perform continuous processes. Another problem is lower efficiency when compared with non-probabilistic algorithms, since the best estimate but probability densities are considered. This is an important issue because robots, being real-time systems, limit the amount of computation that can be carried out. Operating in a real-time requires fast time response. Many state-of-the-art algorithms are approximate, and are not enough accurate.

In this work we propose usage of the probabilistic constraint techniques in the context of probabilistic robotics and in particular to solve robot localization problems. Mobile robot localization, also known as position estimation, is the problem of determining the pose of a robot relative to a given map of the environment. The robot pose cannot be measured directly from the sensors, but it can be obtained from the available with consideration of the model constraints and the underlying uncertainty. Instead of providing a single scenario, the most probable position of the robot in the current moment, the proposed approach is able to characterize all possible positions (consistent with the model) and their probabilities (in accordance with the underlying uncertainty).

The continuous probabilistic constraint framework provides expressive mathematical model that can represent the robot localization problem. The uncertainty and nonlinearity are taken into account in the probabilistic constraint reasoning. The hybridization constraint branch-and-prune algorithms with of Monte Carlo integration techniques increases the efficiency of the approach. Figure 1.1 illustrates the relevant areas of research.

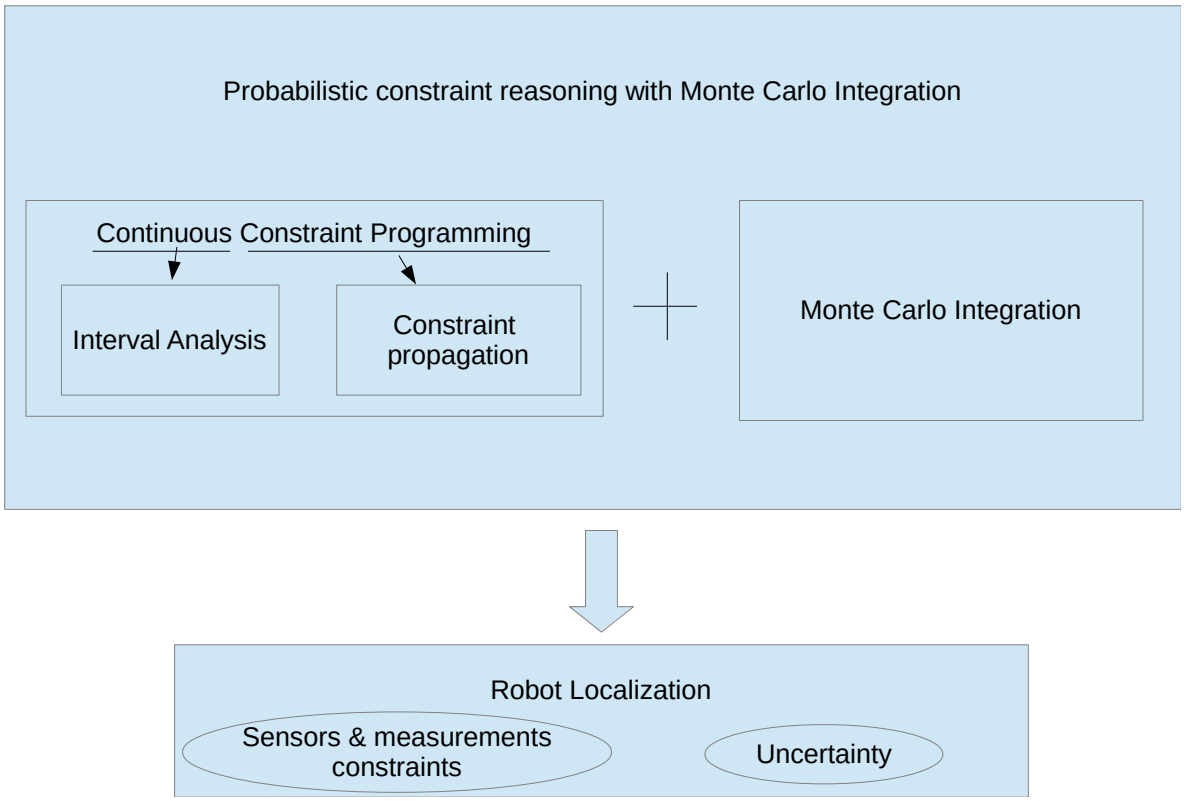

Figure 1.1 Area of the research.

In continuous constraint programming the computations are supported by interval analysis which is a set extension of numerical analysis. The floating-point numbers are replaced by intervals that guarantee safe enclosures of the correct values. The relations between variables are stated in the form of constraints.

The branch-and-prune algorithms of continuous constraint programming are aimed to cover sets of exact solutions for the constraints with sets of interval boxes. The box is Cartesian product of intervals bounded by floating-points.

These algorithms recursively refine initial crude cover of the feasible space (the set of all values, Cartesian product of the initial domains). There are two interleaving steps - branching and pruning, which repeat until a stopping criterion is satisfied. On the branch step the interval box is splitted into a partition of sub-boxes (usually two). On the pruning step an interval box is reduced from the covering into a smaller (or equal) box such that all the exact solutions included in the original box are also included in the reduced box [2].

Continuous constraint programming provides reasoning on safe enclosures of all consistent scenarios. Intervals are used to include all their possible values [3]. The framework provides safe constraint propagation techniques. Constraint propagation reduces the search space by eliminating combinations of values that do not satisfy model constraints. However, in some cases obtained safe enclosure of all consistent scenarios may be too extensive and insufficient to support decisions. This can happen, if initial intervals are wide.

The continuous probabilistic constraint framework is an extension to constraint programming that bridges the gap between pure safe reasoning and pure probabilistic reasoning. This framework provides the decision support in the presence of uncertainty. The interval bounded representation of uncertainty is complemented with a probabilistic characterization of the values distributions. That makes possible to further characterize the scenarios with a likelihood value.

Moreover, this approach does not suffer from the limitations of approximation techniques, due to the constraint programming paradigm that supports it. The approach guarantees safe bounds for the solutions and their likelihoods.

The probabilistic constraint framework provides methods to compute certified enclosures for the probability of consistent scenarios based on reliable interval techniques for multidimensional quadrature. However, these methods are time consuming, limiting the usage of the technology in real problems.

This work explores an hybrid approach that relies on constraint programming and Monte Carlo integration to obtain estimates for the probability of consistent scenarios. The idea is to compute close estimates of the correct probability value by applying Monte Carlo integration techniques that benefit from the contribution of constraint programming to reduce the sample space into a sharp enclosure of the region of integration. Instead of computing guaranteed results with a computationally demanding method, we aim at obtaining accurate estimates much faster.

1.3 Contributions

This work studies the hybridization of Monte Carlo integration techniques with continuous constraint programming for effective probabilistic constraint reasoning. The main contributions are:

- The development of an hybrid approach for probabilistic constraint reasoning that improves the efficacy of the approximate quadrature methods in the context of the constraint programming branch-and-prune algorithm.
- Implementation of the solution proposed and its benchmarking based on a set of generic experimental tests on multidimensional integrals over nonlinear constrained boundaries.
- Application of the developed techniques to the robot localization problem and analysis of its performance on a set of simulated problems.

1.4 Scheme of the document

This dissertation is structured in 5 chapters. After the first introductory chapter the rest of the work is organized as follows:

- Chapter 2 overviews the state-of-the-art, including the basic concepts of continuous constraint programming, probability theory and quadrature methods. Additionally it briefly summarizes the traditional applications of these techniques to robotic problems.

- Chapter 3 presents our proposal to extend the constraint programming branch-and-prune algorithms with Monte Carlo integration techniques. The probabilistic constraint reasoning is analyzed and alternative integration approaches are suggested and evaluated on a set of experiments.
- Chapter 4 discusses the application of our approach to probabilistic robotics and, in particular to robot localization problems. The added value of the approach is highlighted on a set of simulated problems.
- In chapter 5 the contributions of this work are summarized, some open problems are identified, and directions for future work are set.

2

State of the Art

A mathematical model is a mathematical representation of the reality, which describes a system by a set of variables and constraints that establishes relations between them. In engineering systems, particularly in robotics, nonlinearity and uncertainty occur very often. Reasoning with mathematical models under uncertainty is traditionally based on probability theory.

There are two general types of approaches for reasoning under uncertainty. Stochastic approaches reason on the basis of the most likely scenarios, but don't guarantee the safeness of the calculations. The drawback is that such approaches rely on approximations and may miss relevant satisfactory scenarios leading to erroneous decisions.

The other type of approaches provides safe enclosures of all consistent scenarios. Continuous constraint programming operates with intervals that are used to include all the possible values of the variables. Safe constraint pruning techniques only eliminate combinations of values that definitely do not satisfy the model constraints. The drawback is the consideration that all consistent scenarios are equally likely, which can be insufficient to support decisions.

We believe that the combination of those two types of approaches can provide a powerful tool for reasoning under uncertainty which may be applied in several real-world problems, including robotic problems.

This chapter overviews the theoretical basis of continuous constraint programming, probability constraint reasoning and Monte Carlo quadrature methods. It also describes the real-world applications of developed methods of those areas to robotic problems.

2.1 Continuous Constraint Programming

Constraint programming [4], [5] is a programming paradigm wherein relations between variables are stated in the form of constraints. Mathematical constraints are relations between variables, each ranging over a given domain.

A constraint satisfaction problem (CSP) is a classical artificial intelligence paradigm introduced in the 1970's. CSP is characterized by a set of variables (each variable has associated domain of possible values), and a set of constraints that specify relations among subsets of these variables. The solutions of CSP are assignments of values to all variables that satisfy all the constraints [2],[6].

Constraint programming is a form of declarative programming in the sense that instead of specifying a sequence of steps to execute, it relies on properties of the solutions to be found which are explicitly defined by the constraints. The idea of constraint programming is to solve problems by stating constraints which must be satisfied by the solutions. As such, a constraint programming framework must provide a set of constraint-based reasoning algorithms that take advantage of constraints to reduce the search space, avoiding regions that are inconsistent with the constraints. These algorithms are supported by specialized techniques that explore the specificities of the constraint model such as the domain of its variables and the structure of its constraints.

Origins of continuous constraint programming comes from Davis [7], Cleary [8] and Hyvonen [9] and later research that extended constraint programming to continuous domains. In continuous constraint programming the domains of the variables are real intervals and the constraints are equations or inequalities represented by closed-form expressions (expression that can be evaluated in a finite number of operations). In this sense the Continuous Constraint Satisfaction Problem (CCSP) is a triple $< X, D, C >$ where:

$-X$ is a tuple of n real variables $< x_1, \ldots, x_n >$

$-D$ is a box, the Cartesian product of intervals $I_1 \times \ldots I_n$ where each I_i is the interval domain of variable x_i

$-C$ is a set of numerical constraints (equations or inequalities) on subset of variables in X.

Continuous constraint programming integrates techniques from constraint programming and interval analysis into a single discipline. Interval analysis, which addresses the use of intervals in numerical computations, is an important component of continuous constraint programming.

2.1.1 Interval Analysis

Interval analysis was introduced in the late 1950's [10] as a way to represent bounds in rounding and measurement errors, and to reason with these bounds in numerical computations. The introduction to interval analysis can be found in [11]. In recent decades, interval analysis was widely used as the basis for the reliable computing, calculations with guaranteed accuracy [12].

A real interval is a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set. The interval of numbers between a and b, including a and b, is denoted $[a, b]$ (in this work we only consider closed intervals):

$$[a, b] = \{ r \in R \mid a \leq r \leq b \} \tag{2.1}$$

If a = b a = b the interval is degenerated and is represented as a.

The generalization of intervals to several dimensions is of major relevance in this work. An n-dimensional box B is the Cartesian product of n intervals and is denoted by $I_1 \times \cdots \times I_n$, where each I_i is an interval.

Elementary set operations, such as ∩ (intersection), ∪ (union), ⊆ (inclusion) are valid for intervals. While the intersection between two intervals is still an interval, this is not the case with the union of two disjoint intervals, where the result is a set that cannot be represented exactly by a single interval. The operation ⊎ (union hull) gives the smallest interval, containing all the elements of both interval arguments:

$$[a, b] \uplus [c, d] = [min(a, c), max(b, d)] \tag{2.2}$$

Interval arithmetic defines a set of operations on intervals, and is an extension of real arithmetic for intervals. The obtained interval is the set of all the values that result from a point-wise evaluation of the arithmetic operator on all the values of the operands.

Let I_1 and I_2be two real intervals. The basic interval arithmetic operators $\Phi = \{+, -, \times, /\}$are defined as:

$$I_1 \Phi I_2 = \{r_1 \Phi r_2 \mid r_1 \in I_1 \wedge r_2 \in I_2\} \; I_1 \Diamond I_2 = \{ x \Diamond y : x \in I_1 \wedge y \in I_2 \} \tag{2.3}$$
$$with \Diamond \in \{+, -, \times, /\}$$

Under the basic interval arithmetic, the division I_1/I_2 is not defined if $0 \in I_2$.

In practice, interval arithmetic simply considers the bounds of the operands to compute the bounds of the result, since the involved operations are monotonic. Given two real intervals $[a, b]$ and $[c, d]$ the basic interval arithmetic operations can be defined as:

$$[a..b][c..d][a..b] + [c..d] = [a + c..b + d] \quad (2.4)$$

$$[a..b] - [c..d] = [a - d..b - c] \quad (2.5)$$

$$[a..b] \times [c..d] = [min(ac, ad, bc, bd)..max(ac, ad, bc, bd)] \quad (2.6)$$

$$[a..b] / [c..d] = [a..b] \times [1/d..1/c] \quad if\ 0 \notin [c..d] \quad (2.7)$$

The implementation of the interval arithmetic operators is conditioned by the floating point representation of real numbers. In order to guarantee the correctness of the results, the above operations on the bounds of the operands must be performed with outward rounding to the closest floating point. As such, the computed interval is bounded by floating points and always includes the correct real interval.

Several extensions to the basic interval arithmetic were proposed over the years and are available in extended interval arithmetic libraries, namely: redefinition of the division operator, allowing the denominator to contain zero; generalization of interval arithmetic to unbounded interval arguments; extension of the set of basic interval operators to other elementary functions (e.g., *exp*, *ln*, *power*, *sin*, *cos*).

In continuous constraint programming interval arithmetic is used as a safe method for evaluating an expression, by replacing each variable by its interval domain, and applying recursively the interval operator evaluation rules. In fact, the computed interval includes all possible values for the expression, but its width may be much wider than the width of its exact range. Consequently, in interval analysis special attention has been devoted to the definition of interval functions that compute sharp interval images of real functions.

Moreover, interval methods are frequently used in constraint programming due to their efficiency and reliability. One of the applications is finding roots of equations with one variable. The combination of the Newton method, interval analysis, and the mean value theorem gives the interval Newton method [13]. This method can be used to provide rigorous bounds for

the solutions of the system of non-linear equations or to prove non-existence of solutions or.

2.1.2 Constraint Reasoning with Continuous Domains

Constraint reasoning implies the techniques for eliminating values, which do not satisfy the constraints, from the initial search space (the Cartesian product of all variable domains). Branch-and-prune algorithms contain dividing and pruning steps that are recursively applied until a stopping criterion is satisfied. The sets of values that can be proved inconsistent are eliminated on the pruning step [3].

The safe narrowing operators (mappings between boxes) are associated with the constraint. The operators aim to eliminate value combinations that are incompatible with a particular constraint. These operators must satisfy the following requirements: be correct (do not eliminate solutions) and contracting (the obtained box is contained in the original). The properties are guaranteed due to interval analysis methods applied.

For example, consider the constraint:

$$x + y = z \tag{2.8}$$

The following narrowing operators may be associated with this constraint to prune the domain of each variable:

$$X \leftarrow X \cap Z - Y, \quad Y \leftarrow Y \cap Z - X, \quad Z \leftarrow Z \cap X + Y \tag{2.9}$$

If, for instance, the domains of the variables are X = [1,3], Y = [3,7] and Z = [0,5] then interval arithmetic can be used to prune them to:

$$X \leftarrow [1,3] \cap [0,5] - [3,7] = [1,3] \cap [-7,2] = [1,2]$$

$$Y \leftarrow [3,7] \cap [0,5] - [1,2] = [3,7] \cap [-2,4] = [3,4]$$

$$Z \leftarrow [0,5] \cap [1,2] + [3,7] = [0,5] \cap [4,9] = [4,5]$$

With this technique, based on interval arithmetic, safe narrowing operators may be associated with the above constraint which are able to reduce the original box <[1,3],[3,7],[0,5]> into <[1,2],[3,4],[4,5]> with the guarantee that no possible solution is lost.

Once narrowing operators are associated with all the constraints of the continuous constraint satisfaction problem, the pruning can be achieved through constraint propagation. Narrowing operators associated with a constraint eliminate some incompatible values from the domain of its variables and this information is propagated to all constraints with common variables in their scopes. The process terminates when a fixed point is reached i.e., the domains cannot be further reduced by any narrowing operator [14].

The pruning achieved through constraint propagation is highly dependent on the ability of the narrowing operators for discarding inconsistent value combinations. Further pruning is usually obtained by splitting the domains and reapplying constraint propagation to each sub-domain. In general, continuous constraint reasoning is based on such a branch-and-prune process which will eventually terminate due to the imposition of conditions on the branching process (e.g. small enough domains are not considered for branching).

Since no solution is lost during the branch-and-prune process, constraint reasoning provides a safe method for computing an enclosure of the feasible space of a CCSP. It applies, repeatedly, branch and prune steps to reshape the initial search space (a box) maintaining a set of working boxes (a box cover) during the process. Moreover, some of these boxes may be classified as inner boxes, if it can be proved that they are contained in the feasible space (again interval analysis techniques are used to guarantee that all constraints are satisfied).

Figure 2.1 illustrates the results that can be achieved through continuous constraint reasoning for a CCSP with two variables x and y, both ranging within $[-\pi,\pi]$, and a single constraint:

$$x^2y + xy^2 \leq 0.5 \qquad (2.10)$$

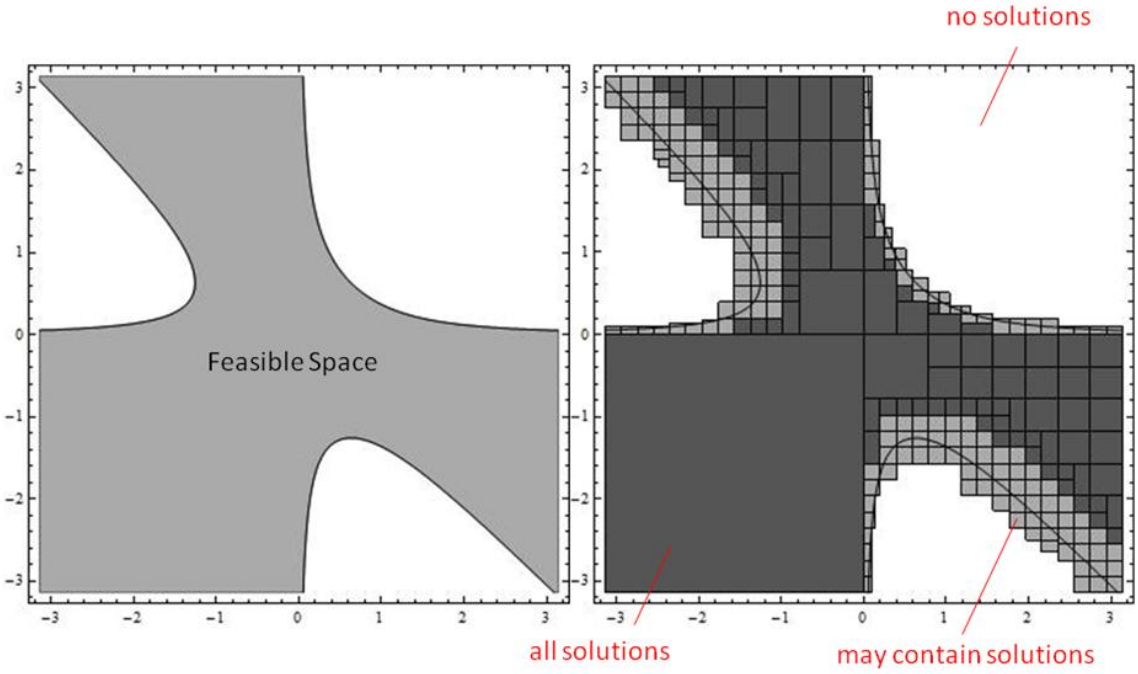

Figure 2.1 A box cover of the feasible space obtained through constraint reasoning.

As show in the figure 2.1, constraint reasoning can prove that there are regions with no solutions (white) and find boxes where every point is a solution (dark gray). Additionally there are some boxes that cannot be proved to contain or not solutions (light gray). These boxes denominated boundary boxes represent the uncertainty on the feasible space enclosure which may be reduced by further splitting and pruning.

2.2 Probabilistic Constraint Reasoning

Probability provides a classical model for dealing with uncertainty [14]. The basic elements of probability theory are random variables, with an associated domain, and events, which are appropriate subsets of the sample space. A real-valued random variable is a function from the sample space into the real numbers.

A probabilistic model is an encoding of probabilistic information that allows the probability of events to be computed, according to the axioms of probability. In the continuous case, the usual method for specifying a probabilistic model assumes, either explicitly or implicitly, a full joint probability density function (p.d.f.) over the considered random variables, which assigns a probability measure to each point of the sample space.

The probability of an event H, given a p.d.f. f, is its multidimensional integral on the region defined by the event:

$$P(H) = \int_H f(X)dX \tag{2.11}$$

In accordance to the axioms of probability, f must be nonnegative and when the event is the entire sample space Ω then P(Ω)=1.

The idea of probabilistic constraint reasoning [3] is to associate a probabilistic space to the classical CCSP by defining an appropriate density function. A constraint (or set of constraints) can be viewed as an event whose probability can be computed by integrating the density function over its feasible space.

In general these multidimensional integrals cannot be easily computed, since they may have no closed-form solution and the event may establish a complex nonlinear integration boundary.

The probabilistic constraint framework relies on continuous constraint reasoning to get a tight box cover of the region of integration and compute the overall integral by summing up the contributions of each box in the cover. Generic quadrature methods are used to evaluate the integral at each box.

Taylor models [15] provide certified methods to obtain sharp enclosures for the integral at each box and are used in probabilistic constraint reasoning [14] to compute interval enclosures for the probability of an event. To guarantee correctness, the contributions of inner and boundary boxes are processed

differently. The inner boxes contribution to the overall integral is the interval obtained by the certified method. However, in the case of boundary boxes, the region is some unknown subset of the box (eventually empty) and its contribution ranges from zero to the obtained integral over the entire box.

Although this probabilistic constraint reasoning approach outputs guaranteed results it is computationally demanding. This justifies an hybrid approach which relies on constraint programming to obtain a cover of the feasible space and then uses an approximate quadrature method on this reduced space to obtain an estimate for the probability of an event.

The study of such hybridization with approximate quadrature methods is the subject of this work. In particular, approaches based on Monte Carlo integration techniques are investigated.

2.3 Monte Carlo Quadrature Methods

Quadrature methods are numerical methods that aim to approximate the value of a definite integral [16],[17]. Numerical integration is used when integrand itself is not specified analytically; analytical representation of the integrand is known, but it is not expressed in terms of the primitive analytic functions; primitive function is complex.

Methods for one-dimensional integrals include [16]: Newton-Cotes Formulas, Midpoint Rule, piecewise step function approximation, Trapezoid Rule: piecewise linear approximation, Simpson's Rule, piecewise quadratic approximation, Gaussian Formulas, Euler–Maclaurin formula, Romberg's method. Methods for multidimensional integrals include: Sparse grids, Quadrature Rules over Simplices, Monte Carlo and its modifications.

Quadrature methods based on the sparse grid approach [18],[19] consider integration over the d-dimensional hypercube, $\Omega = [0,1]^d$. For spaces where functions have bounded mixed derivatives Smolyak's construction (Smolyak,

1963) effectively combats the curse of dimensionality. The multidimensional quadrature formulas based on tensor products of one-dimensional formulas for constructing sparse grids. Regardless of the size of the problem, this process preserves the number of function evaluations and numerical within a logarithmic factor.

Different quadrature methods were proposed for cases where the domain is not an hypercube, but a simplex. A simplex is defined by $n + 1$ vertices in the n-space. The multidimensional regions of integration can often be approximated by union of simplexes. One of the methods is the Duffy transformation, that reduces the problem to a quadrature rule over a hypercube.

In general, Monte Carlo methods are computational algorithms that rely on repeated random sampling to obtain numerical results. The applications of these methods are: integration, simulation and optimization, computational mathematics, inverse problems.

Sampling methods, such as Monte Carlo and its modifications, contrary to numerical quadrature rules, are more efficient for multidimensional integrals of large dimension (d≥15). For Monte Carlo integration, the error scales like O(1/√n), where n is the number of samples, independently from the dimension of the integral. However, for substantial accuracy the convergence, a rate of error is extremely slow. Hence, several techniques to improve the efficiency of Monte Carlo integration were developed, including variance reduction methods, Quasi-Monte Carlo and Adaptive Monte Carlo methods.

The Monte Carlo method is defined as following. We consider the integral of a function f over Ω, subset of R^m with volume V:

$$I = \int_{\Omega} f(x)dx, \tag{2.12}$$

The naive approach is to sample N random points on Ω and calculate the average value of the function. Then the integral can be approximated by:

$$I = F_n \approx V \frac{1}{N} \sum_{i=1}^{n} f(x_i) \tag{2.13}$$

where n is the number of samples and x_i is random point in Ω.

The Monte Carlo estimate converges to the true value of the integral as $n \to \infty$(Law of Large Numbers):

$$\lim_{n\to\infty} F_n = I \tag{2.14}$$

The error estimate for finite n is characterized by the variance of the function $f(x)$:

$$\sigma^2(f) = V\sqrt{\frac{<f^2>-<f>^2}{N}} \tag{2.15}$$

The main advantage of Monte Carlo integration is that the error estimate is independent of the number of dimension. The disadvantage is that integral converges slowly to the true value as the sample size increases. There are several approaches to improve this drawback, such as stratified sampling, importance sampling, and control variates.

Stratified Sampling [20] is the sampling method that partition the domain of integration into sub-domains (strata), and the overall result is computed by summing up the results of Monte Carlo integration in each sub-domain. This method can improve the accuracy of statistical results of Monte Carlo method.

The integration domain Ω is partitioned into a set of m disjoint subspaces $\Omega_1, \ldots, \Omega_m$ (*strata*). Then we can evaluate the integral as a sum of integrals over each stratum Ω_i.

Stratified sampling works well for low-dimensional integration, in case of high dimensionality it does not scale well for integrals. The expected error of this method is lower than variance of ordinary unstratified sampling.

Importance sampling [21] is another modification of Monte Carlo method, and is used for variance reduction. Here a density function p should be chosen similar to the integrand f. This method corresponds to a change of integration variables

$$\int_{\Omega} f(x)dx = \int_{\Omega} \frac{f(x)}{p(x)} p(x)dx \tag{2.16}$$

the estimator is:

$$\hat{I}_p = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \tag{2.17}$$

The variance of the estimator $\hat{I}_p$ depends on the density $p(x)$ from which random samples are drawn. If we choose the density $p(x)$ intelligently, the variance of the estimator is reduced. $p(x)$ is called the *importance density* and

$$w_i = \frac{f(X_i)}{p(X_i)} \tag{2.18}$$

is the importance weight.

The best possible sampling density is $p^*(x)=c\ f(x)$ where c is the proportionality constant:

$$c = \frac{1}{\int_{\Omega} f(x)dx} \tag{2.19}$$

The constant ensures that p^* is normalized:

$$\int_{\Omega} p(x)dx = 1 \tag{2.20}$$

This gives us an estimation to the integral with zero variance, since

$$\frac{f(x_i)}{p(x_i)} = \frac{1}{c}, \forall x_i \in \Omega \tag{2.21}$$

In practice, we cannot use this density, because we must know the value of the integral we want to compute to evaluate c. However, if we choose an importance density $p(x)$ that has a similar shape to $f(x)$, the variance can be reduced. It is also important to choose an importance density p such that it is simple and efficient to evaluate.

Importance sampling is very effective when function $f(x)$ has large values on small portions of the domain.

But this method has different problems and is not practical. If p goes to zero somewhere where f is not zero, the variance can be increased and can actually be infinite. Another common problem that happens in importance sampling is when the sampling density has a similar shape to $f(x)$ except that $f(x)$ has longer (wider) tails. In this case, the variance can become infinite.

The control variates method is a variance reduction technique. It reduce the error of an estimate of an unknown quantity by using the information about the errors in estimates of known quantities.

If we can rewrite the estimator as

$$I = \int_{\Omega} p(x)dx + \int_{\Omega} (f(x) - g(x))dx \tag{2.22}$$

where integral of function $g(x)$ is known and function has the following property:

$$V[f(x) - g(x)] \leq V[f(x)], \tag{2.23}$$

then a new estimator is

$$F = \int_{\Omega} p(x)dx + \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)-g(X_i)}{p(X_i)} \tag{2.24}$$

The variance of this new estimator will be lower than the original estimator.

The method of control variates is more stable than importance sampling, since zeros in g cannot induce singularities in $(f - g)$.

Quasi-Monte Carlo methods are modifications of the Monte Carlo method that do not require that the samples be chosen randomly. The idea is to deterministically distribute the samples uniformly. In this methods the irregularity of the distribution of the samples is measured and is called discrepancy measure.

Adaptive Monte Carlo Methods introduce the idea of adaptive sampling (or sequential sampling), to take more samples where the integrand has the most variation. The previous taken samples are examined, and using this information the placement of future samples is controlled. The variance of the samples in a given domain is computed, and then more samples are taken if the variance exceeds a given threshold.

2.4 Application of Constraint programming and Interval Analysis to Robotics

In [5] some applications of constraint programming for robotics are presented. The example of an application is the design of controllers for sensory-based robots.

Many of the tools developed up to that moment in the CSP and CP paradigms were not adequate for the task, because techniques presume an offline model of computation, whereas controllers in real physical systems should be designed in an online model. Furthermore, the online model must be based on various time structures: continuous, discrete and event-based, and computations should be performed over various type structures domains: continuous and discrete. That requires new models of computation and constraint programming. For this task Zhang and Mackworth [22] defined constraint satisfaction as a dynamic system process. It approaches

asymptotically the solution set of the given constraints, possibly time-varying. In [23], [24] robots are considered as on-line constraint-satisfying devices.

Automatic synthesis of a correct constraint-satisfying controller can be implemented with given a constraint-based specification and a model of the plant and the environment, that was shown for a ball-chasing robot in [25]. Later [26], [27], was designed the system to support hybrid agents i.e., agents with both continuous and discrete interactions with their environment. A constraint satisfying controller could be synthesized by driving the system toward satisfying all constraints (if the robotic system deviated from a satisfactory state it will be driven toward a satisfying state). For reactive systems, such as controllers and signal-processing systems, timed concurrent constraint programming was developed [28].

More recent papers in application of constraint programming in robotics suggest different approaches. For example, in [29] in order to provide a solution in real-time a framework was developed, that models this decision process as a constraint satisfaction problem. It uses techniques and algorithms from constraint programming and constraint optimization.

Using constraint programming for finding the maximal pose error (position and rotational errors) in robotic mechanical systems is proposed in [30]. The authors claim that their global optimizer is very competitive compared to the other methods and provides more robust results.

Another work deals with landmark recognition in mobile robotics, using Multivariable Fuzzy Temporal Profile model (MFTP) based on Constraint Satisfaction Problems (CSP). This model successfully detects 95% of the landmarks on the reference wall [31].

The application of continuous constraint programming to modular robot control was proposed in [32]. They consider the control of modular, hyper-redundant robots, namely robots with many more degrees of freedom than

required for typical tasks. They assumed the control problem as a constraint problem which is a promising approach for robustly handling a variety of non-standard constraints. A parametric model for robotic control was presented and a generic benchmarking model for continuous constraint satisfaction problems was proposed.

In [33] interval analysis was applied to the design and the comparison of three-degrees-of-freedom (3-DoF) parallel kinematic machines. They introduced an algorithm describing this method and two 3-DoF translational parallel mechanisms designed for machining applications were compared using that method. Also [34] considered a Gough-type parallel robot. A numerically robust algorithm based on interval analysis was developed. It allows to solve the forward kinematics, to determine all the possible poses of the platform for given joint coordinates. It is possible to take into account physical and technological constraints on the robot, such as limited motion of the passive joints. This algorithm is competitive in terms of computation time with a real-time algorithm such as the Newton scheme, while being safer.

Application of interval methods for certification of the kinematic calibration of parallel robots was proposed by [35]. A certified approach for this problem in the case of a Gough platform was developed. This approach avoids wrong solutions produced by the classical approaches.

2.5 Summary

In this chapter the basic notions of continuous constraint programming and interval analysis were introduced. It was discussed how probabilistic constraint reasoning extends pure constraint reasoning with special propose techniques for computing the probability of events defined as constraints. This requires the computation of multidimensional integrals over constrained nonlinear regions. Classical methods for multidimensional quadrature were overviewed with special emphasis to Monte Carlo integration techniques.

Finally it is shown how different methods and techniques of constraint programming and interval analysis have been applied to robotics.

The next chapter discusses our probabilistic constraint reasoning approach based on Monte Carlo integration techniques.

3

Probabilistic Reasoning with Monte Carlo Integration

Probabilistic constraint reasoning depends on the joint cooperation of continuous constraint programming with a multidimensional integration method. Here the uncertainty is represented as bounded intervals with a probabilistic characterization of the values distribution.

The probabilistic constraint framework relies on integral computations of probabilistic density functions over constrained regions. This chapter presents alternative approaches for the hybridization of constraint programming with Monte Carlo integration. Continuous constraint programming is used to obtain the feasible space, and then Monte Carlo Integration is applied on this reduced space.

3.1 Probabilistic constraint reasoning

The probabilistic constraint framework complements the representation the uncertainty by means of intervals with a probabilistic distribution of values within such intervals. The main goal of the hybrid approach is the development of a fast and convenient method for calculating the integral on a bounded region. The usage of continuous constraint programming techniques provide the results as a set of boxes that represent an enclosure of the bounded region. Numerical and formal methods of integration can be used to compute the value of integrals with certain accuracy. However, the error committed in such computations can be critical for the specific problems. Whereas, certified Taylor methods guarantee safe computations of the integral returning an interval enclosure that contains the actual value, approximate methods aim at obtaining fast accurate estimates.

In Chapter 2 we already introduced the notions of continuous constraint programming and talked about the branch-and-prune process. Several branch-and-prune algorithms were developed, their main principle is to recursively refine the initial interval box, which is trivially a covering of the feasible space. The two main procedures of those algorithms are branch and prune steps which are recursively applied until a stopping criterion is satisfied. The branching procedure splits an interval box from the covering into a partition of sub-boxes (usually two). The prune procedure narrows down an interval box; it either eliminates a box from the covering or reduces it into a smaller (or equal) box maintaining all the exact solutions. For this task a combination of constraint propagation (CPA) and consistency techniques are usually used. A box is reduced through the consecutive application of the consistency techniques associated with the constraints until a fixed-point is attained [4], [6].

Algorithm 3.1 illustrates a general branch-and-prune scheme. It computes a joint box cover from the initial domains box and is based on the algorithm presented in [6].

Branch&Prune(*D*, *C*, *split*, *eligible*, *order*, *stop*)

Input: *D*: initial domains box; *C*: set of constraints; *eligible*, *stop*: predicate; *split*, *order*: function;
Output: *S*: box cover;

```
1    S ← {D};
2    while (¬ stop(S)) do
3        B ← remove (S, eligible, order);
4        S_i ← split(B);
5        for each B_i ∈ S_i
6            B_i ← CPA(B_i, C);
7            if B_i ≠ ∅
8                S ← S ∪ {B_i};
9            end
10       end
11   end
12   return S;
```

Algorithm 3.1 Branch-and-prune

The input parameters of Algorithm 3.1 are: the initial domains box *D*; a set of constraints *C*; predicates *eligible,* and *stop*; and functions *split* and *order*. The *eligible* predicate checks a box for being appropriate for further processing (for example, it may check if it contains an interval domain that can be further subdivided). The predicate *stop* imposes the stopping criterion such as precision, cardinality of the covering, or computation time. The function *split* implements the branch procedure, it defines how to partition the box into sub-boxes. The function *order* defines a total order between the boxes in the cover determining which box is retrieved for processing.

The output of Algorithm 3.1 is a box cover *S* for the feasible space of the constraints in *C*. In order to compute it, the box cover is initialized with the domains box (line 1) and maintained during the algorithm main cycle (lines 2-11). The first box from the current box cover that verifies the eligible predicate is selected (boxes are ordered accordingly to the order function), and removed from the list (line 3). Then, on the branch step, that box is split into a set of boxes S_i (line 4). Next is the prune step (lines 5-10), where each box in S_i is narrowed by the constraint propagation algorithm *CPA* (line 6) and, if not

discarded (line 7), is added to the box cover (line 8). The algorithm stops when the stopping criterion is fulfilled (line 2).

Branch-and-prune algorithms can be applied for solving different problems. In this work we are focused on the quadrature problem, i.e. evaluation of the multidimensional integrals. Considering this problem, Algorithm 3.1 can be modified in order to compute the value of the integrals. The basic idea is to calculate the value of integral on each box obtained during the branch-and-prune algorithm. The integration can be implemented using any method, for example using Taylor models or Monte Carlo Integration. In this work, we compare the integration using the Taylor method, which produce accurate results and alternatives based on Monte Carlo integration techniques which provide approximate results. This idea is presented by Algorithm 3.2. Basically, it is a modification of Algorithm 3.1, in which the value of the integral on each box is computed at each step. The result of each integration is represented as an interval, the width of which, proportional to the uncertainty around the correct value, will be used as a selection criterion for the next box to process.

Branch&Prune Quadrature(*D*, *C*, *split, eligible, order, stop*)

Input: *D*: initial domains box; *C*: set of constraints; *eligible*, *stop*: predicate; *split, order*: function;
Output: *Q*: interval;

```
1    I ← integrate(D, C) ;
2    S ← {< D, I >}
3    Q ← I ;
4    while (¬ stop(S, Q)) do
5        < B, I > ← remove (S, eligible, order);
6        Q ← Q \ I;
7        S_i ← split(B);
8        for each B_i ∈ S_i
9            B_i ← CPA(B_i, C);
10           if B_i ≠ ∅
11               I_i ← integrate(B_i, C) ;
12               Q ← Q + I_i;
13               S ← S ∪ {< B_i, I_i >};
```

```
14                  end
15              end
16      end
17      return Q
```

Algorithm 3.2 Branch-and-prune with quadrature

The input parameters of the Algorithm 3.2 are similar to the previous algorithm: the initial domains box D, the set of constraints C, predicates *eligible* and *stop*, and functions *split* and *order*. As in the previous case, the *eligible* predicate checks a box for being appropriate for further processing, the predicate *stop* imposes the stopping criterion (such as precision). The function *split* implements the branch procedure and the function *order* defines which box is retrieved next for processing.

The first step of the algorithm is the integration of the entire domains box D considering the constraints C (line 1). The integration, represented in the pseudo code by function *integrate*, can be implemented by any method, such as Taylor, Monte Carlo or any other method. In our approach we assume that the result of the *integrate* function is always an interval. If the function is implemented with the Taylor Model method then this interval must contain the correct value. This is not the case when the Monte Carlo approach is used, where the center or the interval is the estimated approximate value and the width of the interval is made proportional to the estimated deviation.

During the execution of Algorithm 3.2 a box cover S is maintained where the interval computed by function *integrate* is kept associated with the respective box. At beginning the box cover is initialized with the box domains together with the computed interval (line 2). Then, the interval is assigned to variable Q (line 3) that is maintained during the processing to represent the overall integral resulting from the contributions of every box in the cover.

The function *order* selects the first box B from the current box cover that verifies the eligible predicate, and this box is removed from the list (line 5). The

order of boxes is determined by the width of the corresponding intervals. Ordering them by descending interval width we aim at choosing first the boxes with largest uncertainty on the computed integral.

The contribution of box B to the overall integral must be removed from Q (line 6) because we will replace such contribution by the contributions of its sub-boxes. Notice that Q and I are intervals and so the correct way to remove the contribution of I from Q is to use the special interval operator \ that reverse the effect of the interval operator + and is defined as:

$$[a, b] \setminus [c, d] = [a - c, b - d] \tag{3.1}$$

Afterwards, the box B is splitted into sub-boxes on the branch step (line 7). On the prune step the constraint propagation algorithm CPA is applied for each of the sub-boxes (lines 8-9) in order to reduce or eliminate them. If the sub-box is not discarded (line 10), the integral value is computed over this sub-box considering the constraints (line 11). The integral value represented as an interval is added to the previous Q according to the rules of interval arithmetic (line 12). Finally the sub-box with its integral value is added to S (line 13).

Basically, the integral is computed for each sub-box and the obtained values of the integral over these sub-boxes are added (line 12) according to the addition rule of interval analysis discussed in the Chapter 2. The widths of those intervals serve as a criterion for choosing the box on line 5. The sum of all those intervals is the final value of the integral.

Function $integrate$ can be implemented with any quadrature method. The first method proposed in the original probabilistic constraint framework [14] is a certified quadrature method based on Taylor models which is briefly discussed next. Approximate implementations based on Monte Carlo integration methods are presented is section 3.2.

Taylor Models

Taylor models can be used for the computation of the quadrature over a box. Taylor models provide efficient methods to compute enclosures for the quadrature of multivariate functions.

A Taylor model of $f: R^n \to R$ inside an n-dimensional box B is a pair $< p, R >$, where p is a polynomial and R is an interval satisfying $\forall x \in B, f(x) \in p(x) + R$. The degree of the Taylor model is the degree of p.

A Taylor model of a function can be obtained from its multivariate Taylor expansion, using the interval evaluation of the highest order derivatives to compute rigorous bounds for the remainder [6].

Given a Taylor model $< p, R >$ of a function $f: R^n \to R$ inside an n-dimensional box B:

$$\int_B f(x)dx \in \int_B p(x)dx + Rvol(B) \tag{3.2}$$

The enclosure provided can be very sharp: if a Taylor model of order o is used, the quadrature computation has an order of convergence of o + 1 + n. To compute an integral of a function over some region defined as a box, the method can be applied to obtain a sharp enclosure. However, when the region is some unknown subset of the box (eventually empty) the integral ranges from zero to the integral of the function maximum (minimum) over the entire box. In this case, a sharp (and more costly) enclosure is no longer worth computing and a cruder enclosure can be used.

3.2 Methods and algorithms of Monte Carlo integration

The Monte Carlo integration method was discussed in Chapter 2. It provides an approach to estimate the value of the multidimensional integrals.

Let consider the region $H \subseteq R^n$ with a possible nonlinear boundary, indicator function $1_{\mathcal{H}}: R^n \to \{0,1\}$, an n-dimensional box B and function $f: R^n \to R$. Consider N random sample points $\{x_1, \ldots, x_N\}$ uniformly distributed inside B, then the approximation of the integral value is following:

$$\int_{B \cap H} f(x)dx \approx \frac{\sum_{i=1}^{N} 1_H(x_i) f(x_i)}{N} vol(B) \tag{3.3}$$

The indicator function $1_{\mathcal{H}}$ assumes the value 1 if the sample point x_i satisfy the constraints and value 0 if not.

Monte Carlo integration provides an estimate of the uncertainty in the approximation Î obtained by the method. From the central limit theorem the standard deviation σ of the estimate of the integral is:

$$\sigma = \frac{vol(B)}{N} \sqrt{\sum_{i=1}^{N} (1_H(x_i) f(x_i))^2 - (\sum_{i=1}^{N} 1_H(x_i) f(x_i))^2 / N} \tag{3.4}$$

Thus we can represent the obtained value of the integral as the interval $I = [\hat{I} - \sigma, \hat{I} + \sigma]$. The sequence of steps for computing the interval will be the following:

1) Select N random points;
2) Calculate $1_{\mathcal{H}}(x_i) f(x_i)$ for these points and the square of this value
$$(1_{\mathcal{H}}(x_i) f(x_i))^2; \tag{3.5}$$
3) Estimate the mean of f using the average of these samples:
$$\bar{f} = \frac{\sum_{i=1}^{N} 1_{\mathcal{H}}(x_i) f(x_i)}{N}; \tag{3.6}$$
4) Multiply with the volume $vol(B)$ to get an approximation of the integral:
$$\hat{I} = \frac{\sum_{i=1}^{N} 1_{\mathcal{H}}(x_i) f(x_i)}{N} vol(B); \tag{3.7}$$
5) Calculate deviation:
$$\sigma = \frac{vol(B)}{N} \sqrt{\sum_{i=1}^{N} (1_{\mathcal{H}}(x_i) f(x_i))^2 - (\sum_{i=1}^{N} 1_{\mathcal{H}}(x_i) f(x_i))^2 / N}\ ; \tag{3.8}$$
6) Calculate interval:
$$I = [\hat{I} - \sigma, \hat{I} + \sigma]. \tag{3.9}$$

The error of the estimation decreases as $1/\sqrt{N}$. The advantage of Monte Carlo integration is that this result does not depend on the number of dimensions of the integral, while most deterministic methods depend exponentially on the dimension. The Algorithm 3.3 describes the simple Monte Carlo integration method.

Algorithm 3.3. *simpleMonteCarlo Integrate*(*B*, *C*, *N*, *f*, *randomGenerator*, *indicator*, *volume*)

Input: B: n-dimensional box; C: set of constraints; N: number of samples;
f, *randomGenerator*, *indicator*, *volume*: function;

Output: I : interval;

```
1   Î ← 0;
2   S ← 0;
3   i ← 0;
4   while (i ≤ N) do
5       x_i ← randomGenerator(B)
6       f_i ← indicator(C) · f(x_i)
7       Î ← Î + f_i
8       S ← S + f_i^2
9       i++;
10  end
11  Î ← Î · volume(B)/N
12  σ ← sqrt(S − Î^2/N) · volume(B)/N
13  I ← [Î − σ, Î + σ]
14  return I
```

Algorithm 3.3 Monte Carlo integration

The input parameters are the n-dimensional box, the set of constraints, the number of samples, the integrand function f, and such functions as, *randomGenerator* that generates a random point within a box, *indicator* that define whether a point satisfy the constraints, and *volume* that calculates the volume of a box. The output is the integral, represented as an interval.

The algorithm works for a fixed number of samples N (line 4) and maintains two accumulators $\hat{I}$ and S initialized in lines 1 and 2, respectively.

The random generator generates values inside box B (line 5). Then the algorithm calculates the value of the integrand function in that point (line 6), considering the constraints with usage of function $indicator$. The sum of obtained results is calculated (line 7), as well as the sum of the square of the result (line 8). The estimate of the integral $\hat{I}$ is computed by multiplying the average of these samples by the volume of the box (line 11). The deviation σ is computed by the formula above (line 12). The result I is an interval enclosing the estimated value within the computed deviation (line 13).

This method assumes that the number of samples is initially defined. However, if we chose a fixed number of samples, we cannot guarantee that the result of the calculation will be acceptable. In case of a too small number of samples, the accuracy may be unsatisfactory. In case of large number of trials the calculations may take too long. An alternative is to proceed the calculation until some predefined required deviation is obtained (or width of interval represented as double deviation). This approach is shown in Algorithm 3.4.

Algorithm 3.4. *deviationMonteCarlo Integrate*(B, C, N, ε, f, *randomGenerator*, *indicator*, *volume*)

Input: B: n-dimensional box; C: set of constraints; N: number of samples; ε: accuracy;
f, *randomGenerator*, *indicator*, *volume*: function;
Output: I : interval;

1 $\hat{I} \leftarrow 0$;
2 $S \leftarrow 0$;
3 $i \leftarrow 0$;
4 $\sigma \leftarrow 0$;
5 **while** $(i \leq N)\ or\ (\sigma > \varepsilon)$ **do**
6 $x_i \leftarrow randomGenerator(B)$
7 $f_i \leftarrow indicator(C) \cdot f(x_i)$
8 $\hat{I} \leftarrow \hat{I} + f_i$
9 $S \leftarrow S + f_i^2$
10 $\sigma \leftarrow \sqrt{S - \hat{I}^2/i} \cdot volume(x_i)/i$
11 $i++$
12 **end**

13	$\hat{I} \leftarrow \hat{I} \cdot volume(B)/N$
14	$\sigma \leftarrow \sqrt{S - \hat{I}^2/N} \cdot volume(B)/N$
15	$I \leftarrow [\hat{I} - \sigma, \hat{I} + \sigma]$
16	**return** I

Algorithm 3.4 Monte Carlo integration with deviation calculation for the interval representation

This algorithm differs from the previous one in that it calculates the deviation for each sample (line 10). The calculation proceeds while the required value of accuracy ε is not obtained (line 5). All other steps are similar to the previous algorithm.

The possible advantage of this approach is that it can be specified a required accuracy which must be fulfilled after the initial number of trials. The drawback is that this requirement may be too demanding specially in articulation with the branch-and-prune algorithm.

3.3 Experimental results

In this work we implement and compare three methods for integration. One is the pure Monte Carlo integration, the other two are based on Algorithm 3.2 (*Branch&Prune Quadrature*) one uses Taylor Models, and the other uses Monte Carlo integration. We will test those methods on different problems and analyze the results. Since the results of integration are represented as intervals, the criterion of comparison is the error of the interval midpoint. For a given exact value v and an approximation value v_{approx} the relative error is

$$\eta = \left|\frac{v - v_{approx}}{v}\right| \tag{3.10}$$

In the following we present the integrals extracted from [15], the graphs of the integrand functions over the constrained regions, and the results of computations. We compare the described above methods and present tables with the relative errors. The tables contain the values of errors of the midpoints for every 30 seconds execution time. We also present the graphics of the errors.

3.3.1 Experiment 1

Integral $I_1 = \int_{(x_1-1)^2+(x_2-1)^2\leq 1}\left(2+\cos\left(20(x_1^2+x_2^2)\right)\right)dx$.

The illustration of this function is shown on

Figure 3.1. The same function on the constraint region of integration $((x_1-1)^2+(x_2-1)^2 \leq 1)$ is presented on Figure 3.2.

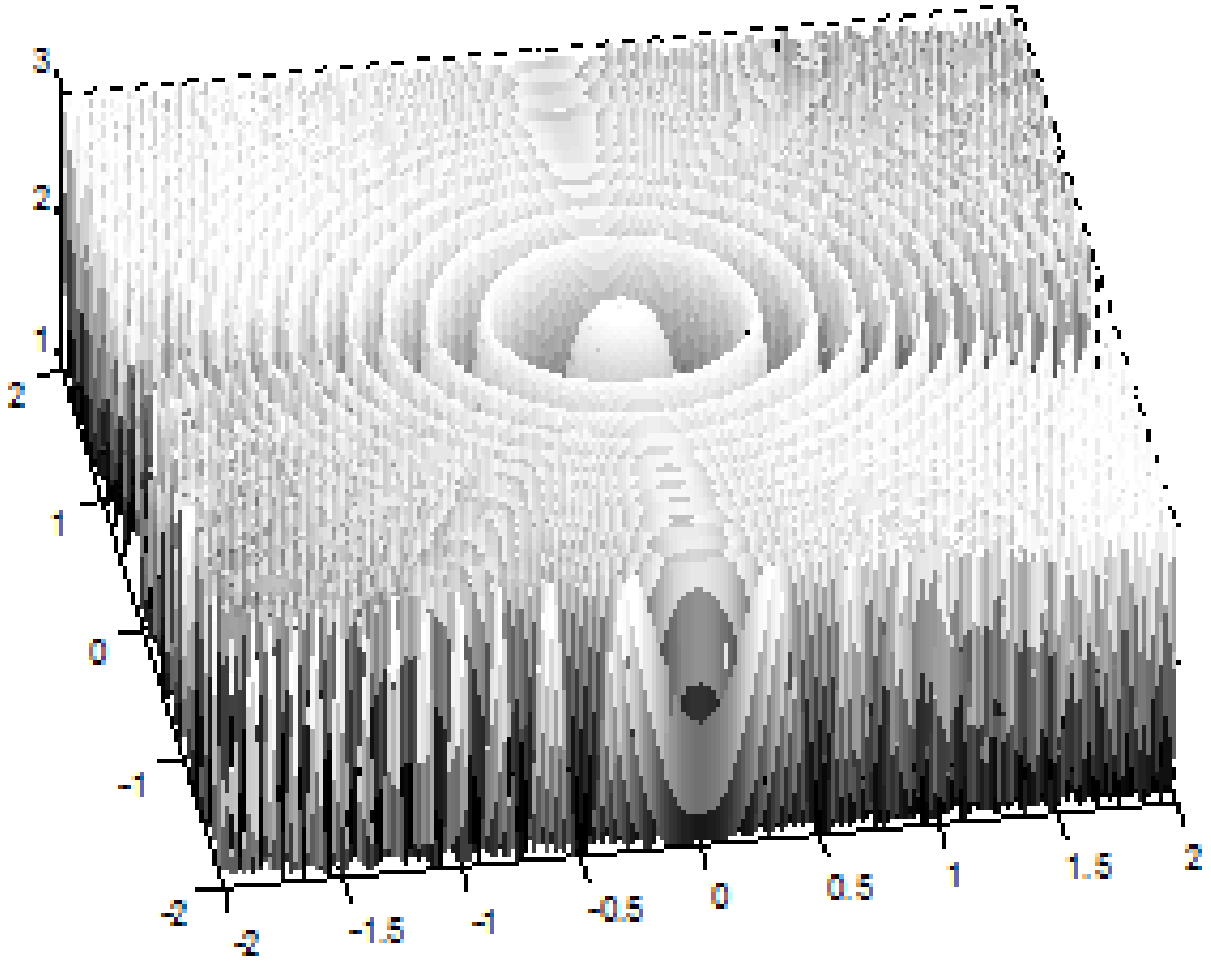

Figure 3.1 Graph of the integrand function f_1

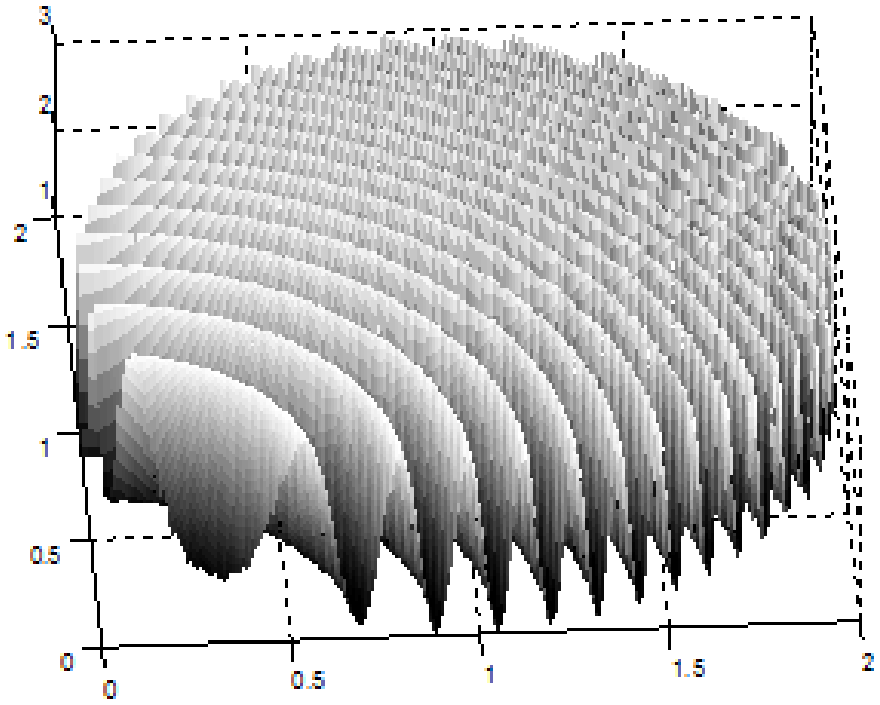

Figure 3.2 Graph of the integrand function f_1 on the constraint region

The integrand function is highly oscillatory. We consider the multidimensional integral with the integration domain represented as unit circle.

We compare the midpoint of the integral with the correct value obtained by formal method in Mathematica [15]. The correct value of the integral is $I_1 = 6{,}300118904167976$. The application of the algorithms gives the following results shown in Table 3.1.

Table 3.1 Integral I_1. Relative errors

Time, [seconds]			30	60	90	120	150	180	210	240	270	300
Taylor CCP			0,000026	0,000013	0,000009	0,000006	0,000005	0,000004	0,000003	0,000003	0,000003	0,000003
Simple MC			0,000220	0,000031	0,000214	0,000115	0,000057	0,000088	0,000125	0,000008	0,000047	0,000072
MC +CCP	10	-	0,000380	0,000315	0,000262	0,000231	0,000204	0,000176	0,000163	0,000154	0,000153	0,000146
		0,1	0,000455	0,000374	0,000276	0,000235	0,000216	0,000193	0,000175	0,000170	0,000158	0,000151
		0,01	0,000814	0,000374	0,000317	0,000272	0,000209	0,000198	0,000183	0,000168	0,000156	0,000148
	100	-	0,000005	0,000068	0,000042	0,000033	0,000043	0,000037	0,000032	0,000026	0,000031	0,000026
		0,1	0,000126	0,000127	0,000052	0,000037	0,000026	0,000033	0,000033	0,000028	0,000033	0,000031
		0,01	0,000247	0,000186	0,000063	0,000042	0,000009	0,000029	0,000034	0,000029	0,000035	0,000037

Table 3.1 shows the values of the errors obtained by the various techniques for execution times of 30, 60, ..., 300 seconds. Comparing values this allows to understand the best method for a specific problem.

In the first line, we present the results for the integration based on continuous constraint programming with Taylor models (*Branch&Prune Quadrature* with function *integrate* based on the Taylor model quadrature technique).

The second line contains the results of calculation errors using the pure Monte Carlo method (*simpleMonteCarlo Integrate* over the initial domains box). For this and the following methods that use Monte Carlo techniques the error is calculated as an average value of 20 independent tests. This is done in order to obtain objective results, since Monte Carlo method is a stochastic nondeterministic method and its results cannot be fairly evaluated based on a single experience (there is a significant fluctuation on the values obtained).

The next 6 rows in the table, are the error values for the Monte Carlo method combined with continuous constraint programming. The first 3 lines show errors for calculations, in which the minimum number of trials for each box equal to 10. In the first of that three lines there are the results of Algorithm 3.3 (*Branch&Prune Quadrature* with *simpleMonteCarlo Integrate*), in which the number of trials is fixed for each box (there are precisely 10 samples). The next row shows the results of the calculation error of Algorithm 3.4 (*Branch&Prune Quadrature* with *deviationMonteCarlo Integrate*), in the case where the minimum value of the deviation is defined and is equal to 0.1, so that sample process ends when the deviation decreases until the required value. In the next row the specified value of the deviation is 0.01. The next 3 lines are similar, but the minimum number of trials is 100. Such experiments are carried out in order to determine which configuration is best for this example.

From this table we can conclude that the best results we obtained with the Taylor model approach. The graphs with the error values are shown on Figure 3.3 and Figure 3.4.

First we compare the results for the Monte Carlo integration method combined with continuous constraint programming. Figure 3.3 shows the errors of the integral values, obtained by this method with different configurations. From this graphic we can conclude that increasing the number of trials decreases the error of computations. In this case Algorithm 3.3 for the Monte Carlo integration presents better results than Algorithm 3.4 with the required value of the deviation. It can be explained by the fact that Algorithm 3.4 takes more time for the calculations. In Algorithm 3.4 the calculations are similar to Algorithm 3.3, but continue, if not achieved the desired deviation, i.e. the calculation takes a bit more time, but it ensures that the value of the deviation is less than or equal to the specified value.

Comparing the obtained results, we conclude that for this example, the best result is at line 6 of table 3.2, corresponding to the hybrid approach with a fixed number of 100 trials.

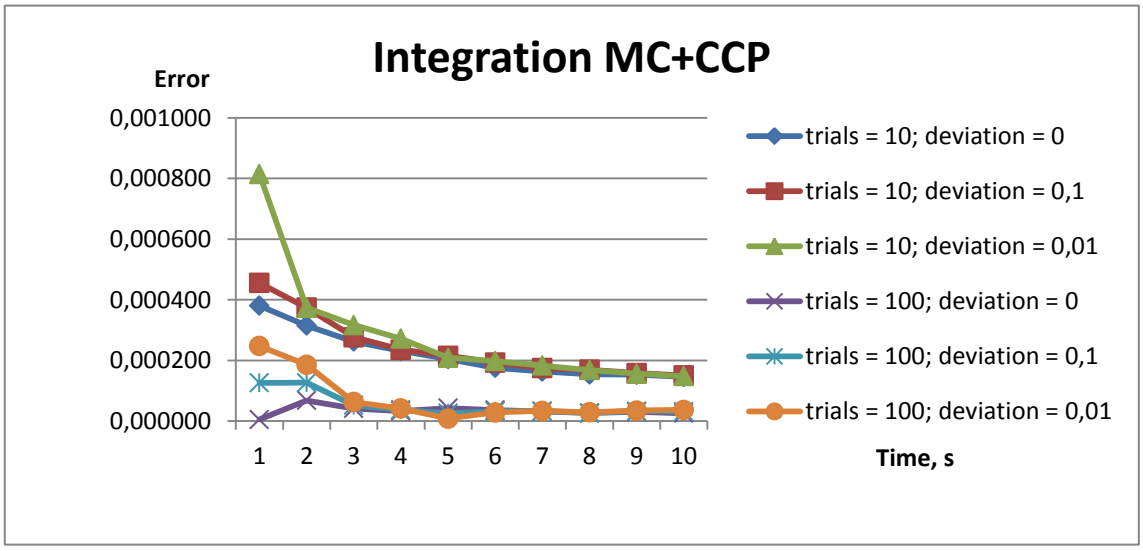

Figure 3.3 Errors of Monte Carlo integration method combined with continuous constraint programming.

Figure 3.4 shows the comparison of the integration with Taylor models, pure Monte Carlo and Monte Carlo with continuous constraint programming (best value). For this example, the best results were obtained using the Taylor method closely followed by the Monte Carlo with CCP method.

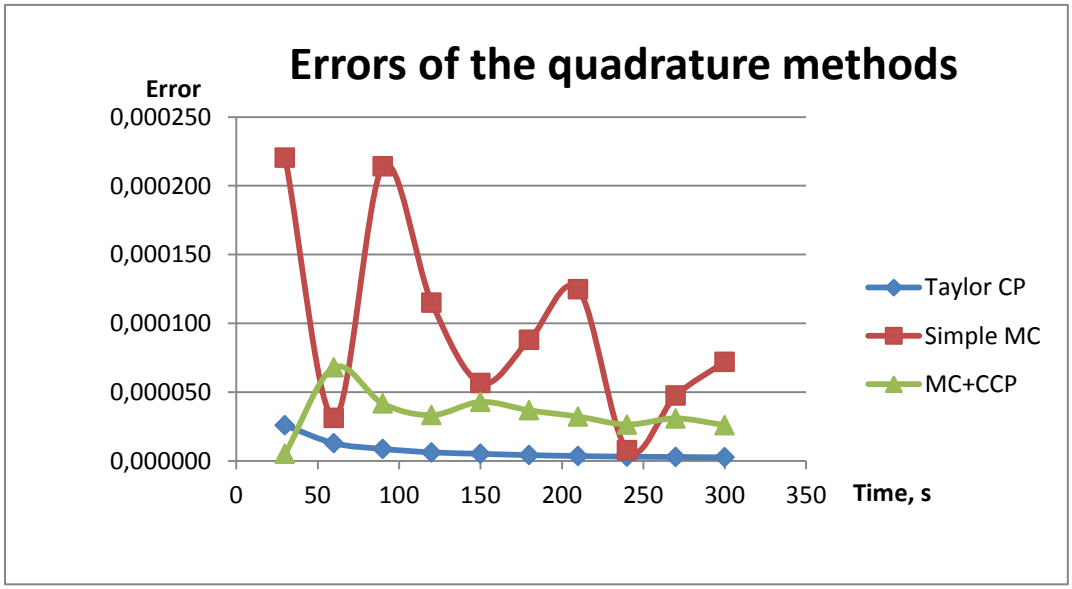

Figure 3.4 Errors of the quadrature methods

For these approaches, the value of the errors decreases with the time. However, for pure Monte Carlo the results are oscillating, that due to the fact that Monte Carlo is a stochastic method.

These results indicate that the combination of Monte Carlo with continuous constraint programming gives an advantage over the pure Monte Carlo integration method. The branch-and-prune algorithm reduces the sampling space that improves the efficiency of the Monte Carlo method and makes it more robust to stochastic oscillations.

Thus, the order of the results from best to worst is:

1) Taylor models
2) Monte Carlo with continuous constraint programming
3) Pure Monte Carlo

3.3.2 Experiment 2

$$\text{Integral } I_2 = \int_{(x_1-1)^2+(x_2-1)^2+(x_3-1)^2 \le 1} (2 + \cos(20(x_1^2 + x_2^2 + x_3^2)))dx$$

This integral is similar to the integral from the previous example, but it has one more dimension, another variable x_3. The increase on the dimension of the integral complicates the calculations.

Table 3.2 shows the results errors of the calculations with the Taylor method, the pure Monte Carlo, and the CCP + Monte Carlo method. The exact value of the integral obtained using formal methods is 8.37845. The table structure is similar to that in Example 1.

Table 3.2 Integral I_2. Relative errors

time			30	60	90	120	150	180	210	240	270	300
Taylor CCP			0,000057	0,003221	0,003459	0,003633	0,003126	0,002786	0,002642	0,002527	0,002506	0,002518
Simple MC			0,000151	0,000798	0,000432	0,000135	0,000410	0,000090	0,000150	0,000232	0,000284	0,000337
MC+CCP	10	-	0,003569	0,003576	0,003709	0,003880	0,004001	0,003991	0,004068	0,004130	0,004133	0,004142
		0,1	0,004454	0,004850	0,005046	0,004960	0,005156	0,005246	0,005249	0,005281	0,005249	0,005319
		0,01	0,012221	0,012495	0,012358	0,012745	0,012672	0,012798	0,012892	0,012893	0,012916	0,012917
	100	-	0,000320	0,000213	0,000057	0,000372	0,000294	0,000177	0,000136	0,000008	0,000108	0,000052
		0,1	0,000534	0,000342	0,000228	0,000346	0,000220	0,000130	0,000110	0,000039	0,000075	0,000044
		0,01	0,000747	0,000471	0,000398	0,000319	0,000145	0,000082	0,000084	0,000070	0,000043	0,000036

Figure 3.5 illustrates the results for the Monte Carlo integration method combined with continuous constraint programming, showing the errors of the integral values, obtained by this method with different configurations.

Comparing the different configurations we note that the best is at line 6 of Table 3.2 with a fixed value of 100 trials.

Figure 3.6 shows the error values obtained by the different methods. Unlike the previous example, the method of Taylor presents the worst results. After 30 seconds of execution time, the error value is acceptable, but it abruptly increases. This example clearly shows that the calculation of multidimensional integrals using the Taylor method may be very inefficient. In this example we added just one dimension, and Taylor method significantly decreased performance. That's why we can talk about the effectiveness of the Monte Carlo

method, for which the multi-dimensionality of the integral does not affect exponentially the efficiency of the algorithm.

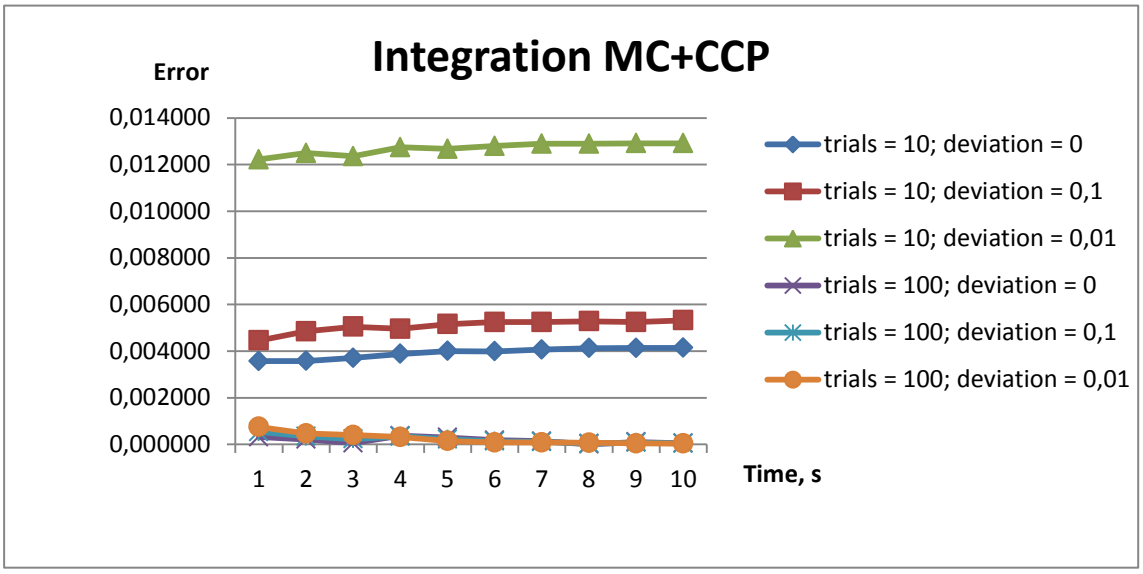

Figure 3.5 Errors of Monte Carlo integration method combined with continuous constraint programming.

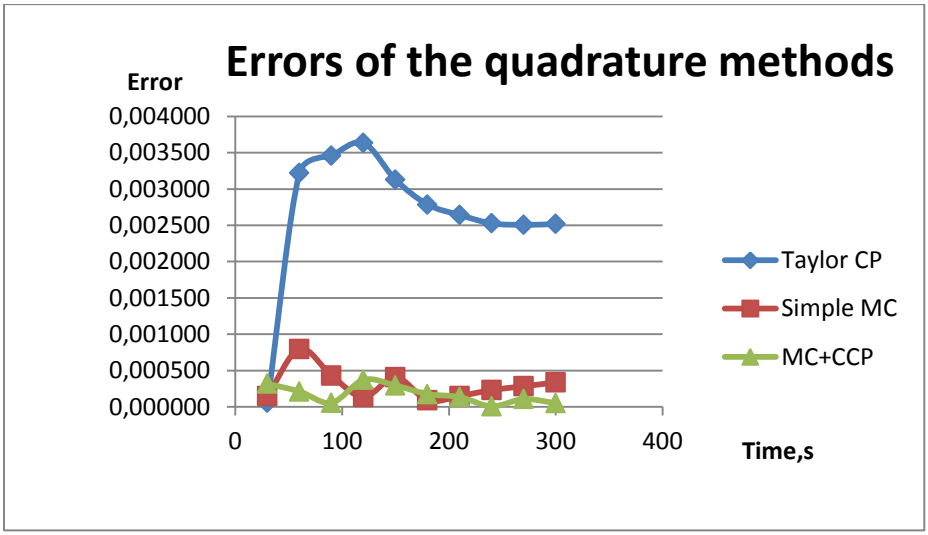

Figure 3.6 Errors of the quadrature methods

Thus, in this example, the order of the results from best to worst is:

1) Monte Carlo with continuous constraint programming
2) Pure Monte Carlo
3) Taylor models

3.3.3 Experiment 3

Integral $I_3 = \int_{\Omega} \arctan(x_1^2 + x_2^2)dx$

$\Omega = \{x \in [0,2]^2 : \|x\|_2^2 \leq 4, x_2 \leq \sin x_1\}$

In this example, the integrand function is shown in Figure 3.7. The same function on the bounded domain is shown in Figure 3.8.

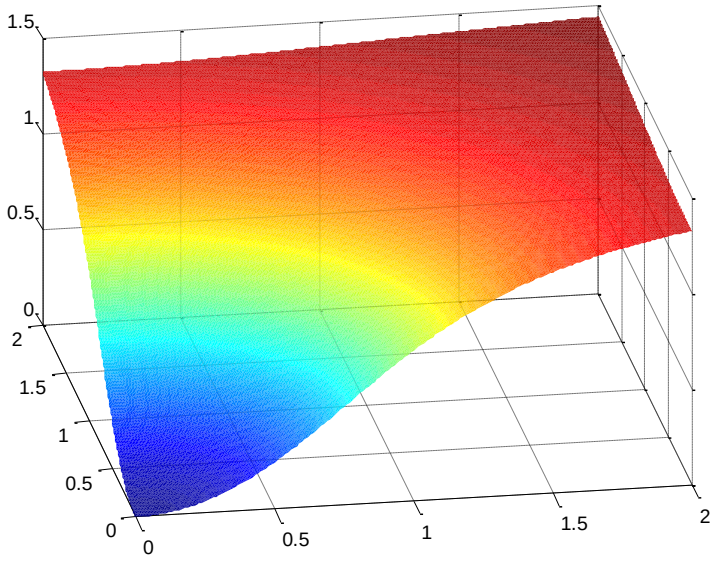

Figure 3.7 Graph of the integrand function f_3

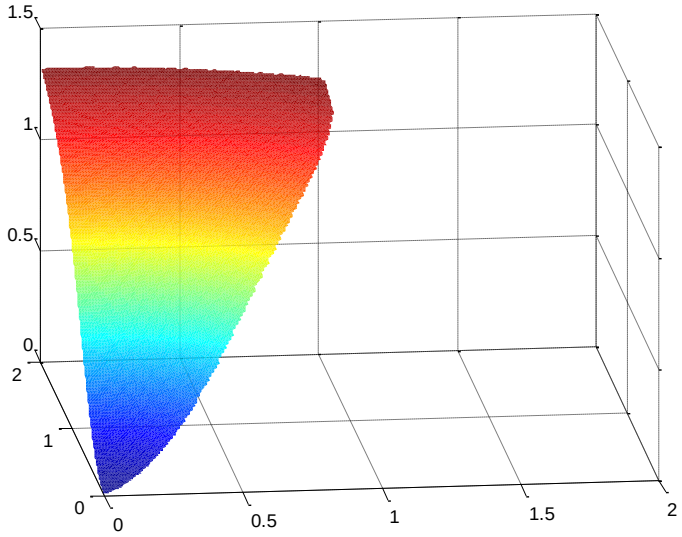

Figure 3.8 Graph of the integrand function f_3 on the constraint region

The exact value of the integral is equal to 1.2562052338296295. As in the previous examples Table 3.3 shows the results obtained by each method.

Table 3.3 Integral I_3. Relative errors

time			30	60	90	120	150	180	210	240	270	300
Taylor CCP			0,000061	0,000023	0,000015	0,000010	0,000008	0,000007	0,000005	0,000004	0,000004	0,000004
Simple MC			0,000525	0,000617	0,000123	0,000127	0,000224	0,000167	0,000215	0,000264	0,000149	0,000197
MC+CCP	10	-	0,000381	0,000268	0,000229	0,000206	0,000190	0,000174	0,000158	0,000145	0,000139	0,000129
		0,1	0,000378	0,000268	0,000208	0,000177	0,000160	0,000148	0,000135	0,000118	0,000108	0,000099
		0,01	0,000415	0,000310	0,000229	0,000206	0,000194	0,000180	0,000169	0,000150	0,000140	0,000127
	100	-	0,000039	0,000030	0,000027	0,000026	0,000020	0,000021	0,000020	0,000019	0,000017	0,000017
		0,1	0,000049	0,000035	0,000028	0,000026	0,000021	0,000020	0,000020	0,000020	0,000018	0,000017
		0,01	0,000060	0,000040	0,000029	0,000026	0,000021	0,000018	0,000020	0,000020	0,000019	0,000017

Figure 3.9 illustrate the results for the Monte Carlo integration method combined with continuous constraint programming. The errors of the integral values obtained by this method with different configurations are presented. Comparing different configurations we note that the best is again at line 6 of Table 3.3, with a fixed minimum value of 100 trials. Also it should be noted, that Algorithm 3.4 for Monte Carlo integration with the required deviation presents better results, than Algorithm 3.3 for a number of trials equal to 10.

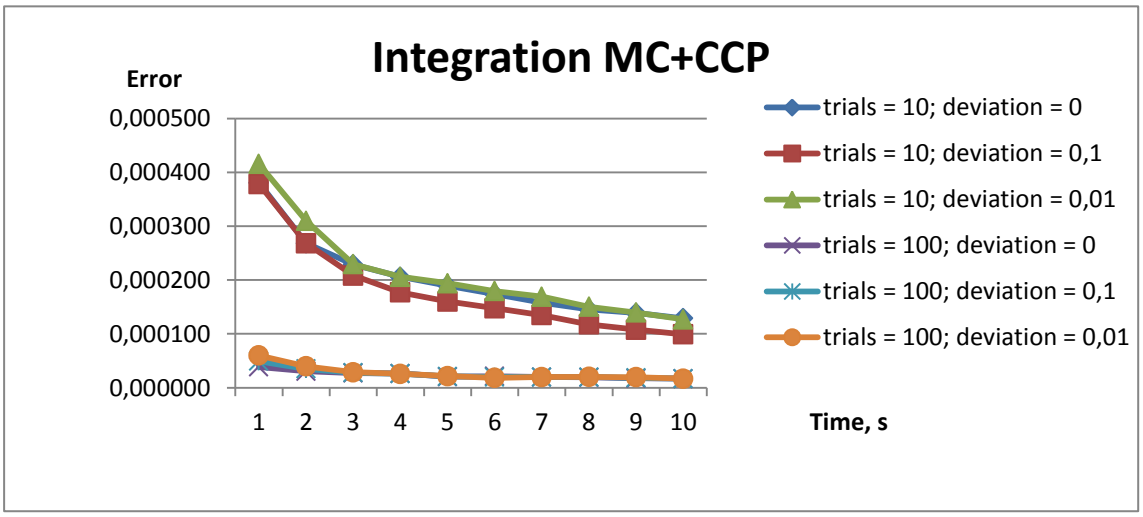

Figure 3.9 Errors of Monte Carlo integration method combined with continuous constraint programming

Figure 3.10 shows the comparison of the integration results obtained with Taylor models, pure Monte Carlo and Monte Carlo with continuous constraint programming.

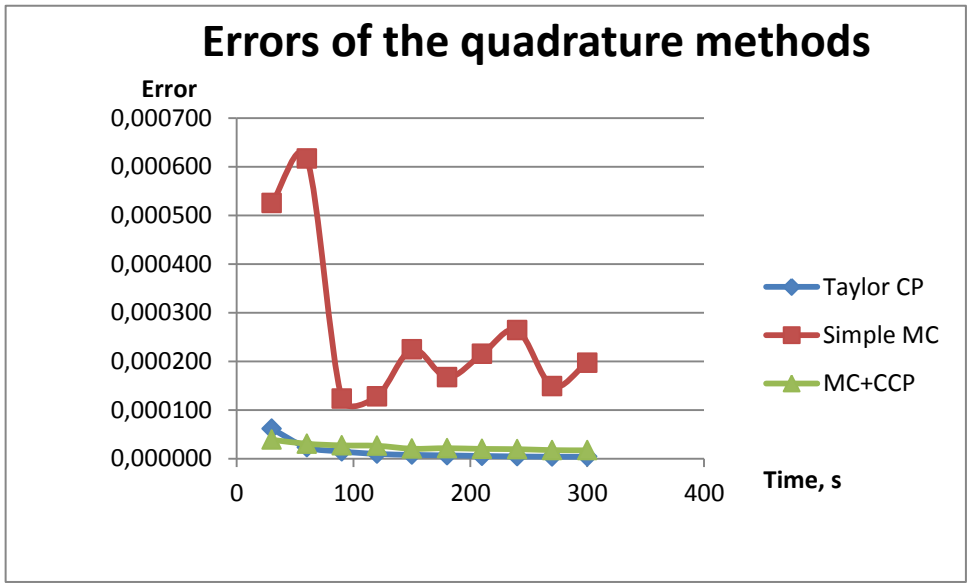

Figure 3.10 Errors of the quadrature methods

In this case, the results are ordered as follows:

1) Taylor models

2) Monte Carlo with continuous constraint programming

3) Pure Monte Carlo

We again observe that the usual method of Monte Carlo loses in efficient with respect to its hybridization with continuous constraint programming.

3.4 Comparison of the methods

Taylor models

In two cases integration with Taylor models gave better results than other methods. But in cases of higher dimension they obviously lose, which is a clear disadvantage of this method.

Monte Carlo

Pure Monte Carlo cannot guarantee the accurate value of the integral and showed the worst performance in the all experiments. The advantage of this method lies in the simplicity of its implementation and application possibilities for solving multidimensional integrals. The disadvantage is the poor efficiency compared to other methods considered.

Monte Carlo with continuous constraint programming

The hybridization of the Monte Carlo method with continuous constraint programming is clearly superior in performance with respect to the pure Monte Carlo method. This is based on the branch and-prune integration algorithm where sampling is performed on reduced boxes requiring less execution time than in the pure Monte Carlo approach. This method can be successfully applied when Taylor method fails as it is more efficient for the calculation of multidimensional integrals.

The work studies a modification of the simple Monte Carlo algorithm, in which further sampling is enforced until a required value for the deviation is achieved. This is done in order to ensure accuracy. Often this approach leads to an increase in computation time. In general it is necessary to find a balance between the number of trials and the required deviation. A possibility could be to specify an adequate maximum number of trials so that calculations do not take too long.

3.5 Conclusion

The goal of this chapter is to propose a method to compute close estimates of the correct integral value by applying Monte Carlo integration techniques that benefit from the contribution of constraint programming to reduce the sample space into a sharp enclosure of the integration region. This integration method will be used in the next Chapter, where a robot localization algorithm is presented based on the developed techniques.

We addressed Monte Carlo quadrature methods in the context of the branch-and-prune algorithm of constraint programming. Several methods and algorithms of integration were developed and their parameterization was studied. We tested the methods on different benchmark examples involving multidimensional quadrature over constrained regions. We compared those methods with simple Monte Carlo integration and certified quadrature methods based on Taylor models.

In many cases the results depend on the integrated function, the kind of constraints and the dimension of the integral. Further experimentation is needed in order to support our main conclusions. The Taylor method approach guarantee the safe enclosure of the integral correct value, but in case of many dimensions it fails to compute the integral with reasonable accuracy. On the contrary, the efficiency of the Monte Carlo methods do not depend exponentially on the number of dimensions of the problem. The hybridization of the Monte Carlo integration method with constraint propagation is a promising approach that allows the estimation of the value of a multidimensional integral, usually faster than the pure Simple Monte Carlo alternative and requires less number of samples. This approach is able to provide accurate results even in cases when the Taylor method fails.

4

Application to Robot Localization

Among the large variety of robots we can distinguish the class of mobile robots that perform operations which require the displacement of the robot in space. Mobile robots have to be accurately positioned in the working environment, i.e. must be identified by their coordinates.

In this Chapter we explain how Monte Carlo Integration with Continuous Constraint Programming can be used for solving localization problems. Research carried out in this work, allowed to implement algorithms designed to solve the problem of mobile robot localization.

To be autonomous, mobile robots must be able to estimate their position from available prior information about the environment and measurements provided by its sensors during navigation. An autonomous robot must integrate the incoming information from the sensory inputs into a consistent model of the environment and simultaneously determine its own location.

4.1 Mobile service robots

The Monte Carlo Integration with Continuous Constraint Programming can be applied to the mobile robot localization. One of such robots ServRobot (Figure 4.1) that was created in the scope of the Research and Technology Development component of the Holos company. ServRobot is an service robot that can be applied in surveillance systems. The robot moves around in their environment and collects the sensory information. ServRobot can work and perform certain tasks in conditions that are difficult to humans thus replacing them. Robot adapts to different types of use and environmental conditions which addresses a new paradigm in video surveillance systems.

Figure 4.1 Mobile robot ServRobot

ServRobot is a skid-steered four-wheel mobile robot developed using state-of-the-art technology. The robot is mechanically robust and simple for outdoor navigation. The motion direction is changed by turning the left- and right-side wheels at different velocities. The robot is equipped with several sensors that receive the information about surrounding environment.

- optical encoders;
- inertial measurement unit with an accelerometer;
- gyroscope;
- magnetometer;
- sonars;
- LADAR.

The estimation of the robot position can be implemented with the information provided by those sensors. The front sonars are used to capture environment information ahead of the vehicle within a range of 3 meters. A LADAR (Laser Detection and Ranging) provides a panoramic view of the environment, it gathers distance measurements within a range of 20 meters.

ServRobot has the following technical paramenters:

- Weight: 80 kg
- Load capacity: 65kg (with maximum slope of 5%)
- Maximum speed: 10 km / h
- Battery: 4 hours
- Ground distance: 25 cm
- Dimensions:
  - Length: 152 cm
  - Width: 60 cm
  - Height: 97 cm

The robot is driven by DC motors that are the most commonly used method for locomotion in mobile robots. These motors can produce sufficient power for a variety of tasks [36]. DC motor has a complicated construction, however, the motor control system is easier than control system induction motor, which nevertheless are applicable in robotics. The research concerning induction motors and the universal method of its general structural analysis was proposed in [37]. It can be used in engineering practice for the purpose of investigating complex electromechanical systems.

The control of the robot can be implemented with various of methods, such as classical approaches (PID) and the modern ones (sliding mode control, passivity-based control) [36]. Another important area is robot learning that allows a robot to adapt to its environment through learning algorithms. Artificial neural networks can be used for a number applications of in robotics. The application of complex-valued neural networks for control of automation systems was proposed in [38]. There were presented the developed algorithms that can be applied for the robot control and robot learning.

4.2 Simultaneous Localization and Mapping methods

The idea of probabilistic robotics is to represent uncertainty explicitly, using the calculus of probability theory. Instead of relying on a "single scenario", probabilistic algorithms represent information by probability distributions over a whole space of possible hypotheses.

Odometry aims to estimate the change in the position of the robot over time. The robot gets the current angular velocity of the wheels and the current rotation angles of the wheels relative to the initial position. Knowing the angles of rotation of the wheels, the current angle of rotation of the robot is calculated by geometry formulas. Coordinates of the robot are calculated as integrals of the angular velocities of the wheels. The formulas depend on the specific kinematic configuration of the robot.

With the use of visual odometry or based on the analysis of ranging data the robot can determine its offset relative to the previous position. In the ideal case, when the calculations are accurate and faultless, it is possible to build a map of environment and describe its trajectory. Unfortunately, in reality, at each step there is a small calculation error (due to measurements error, interference, restrictions imposed by the algorithms, etc.). Over time, total accumulative error continues to grow, so that the global map will be inaccurate. The complexity of the technical process of determining the current location and constructing the map is magnified by the low precision instruments involved in the calculation of the current location. In order to deal with this problem Simultaneous Localization and Mapping Methods (SLAM) were designed. An overview of SLAM methods is given in [39].

SLAM is aimed to solve two problems:

1) build a map of an unknown environment

2) navigating the environment using this map

The large number of researches was conducted in this area. The surveys can be found in [1], [39], [40]. It should be noted that this problem is not completely resolved and is still under investigation.

The important problem in a SLAM algorithm is the representation of the joint distribution over robot poses and maps, because maps are usually represented by an high number of parameters. The mostly used representations are: the feature-based - collection of landmark locations and correlated uncertainty; the grid based - collection of discretized obstacle/free-space pixels; and topological - collection of nodes and their interconnections. SLAM problems can be divided into several connected parts: landmark extraction, data association, state estimation, state update and landmark update.

Different SLAM probabilistic approaches exists, such as Kalman Filters, Particle Filters also called Monte Carlo localization, and Expectation Maximization , which are mathematical derivations of the recursive Bayes rule [40]. There are variations of Kalman Filter: the Extended Kalman Filter and Information Filtering. There is a number of other SLAM methods, for example Compressed Extended Kalman Filter, Extended Information Filtering, Rao-Blackwellised particle filters for laser-based SLAM, incremental Smoothing and Mapping, Tree-based netwORk Optimizer, etc.

The Extended Kalman Filter deals with nonlinear process model and nonlinear observation model. It is aimed to linearize a nonlinear dynamic system for use in a Kalman Filter, which is applied to estimate the position of the robot through a motion model, and its environment through an observation model, based on its odometry and landmark position measurements. It is one of the most successful SLAM algorithms and is used navigation systems and GPS. The example of Extended Kalman Filter from [1] is shown on Figure 4.2.

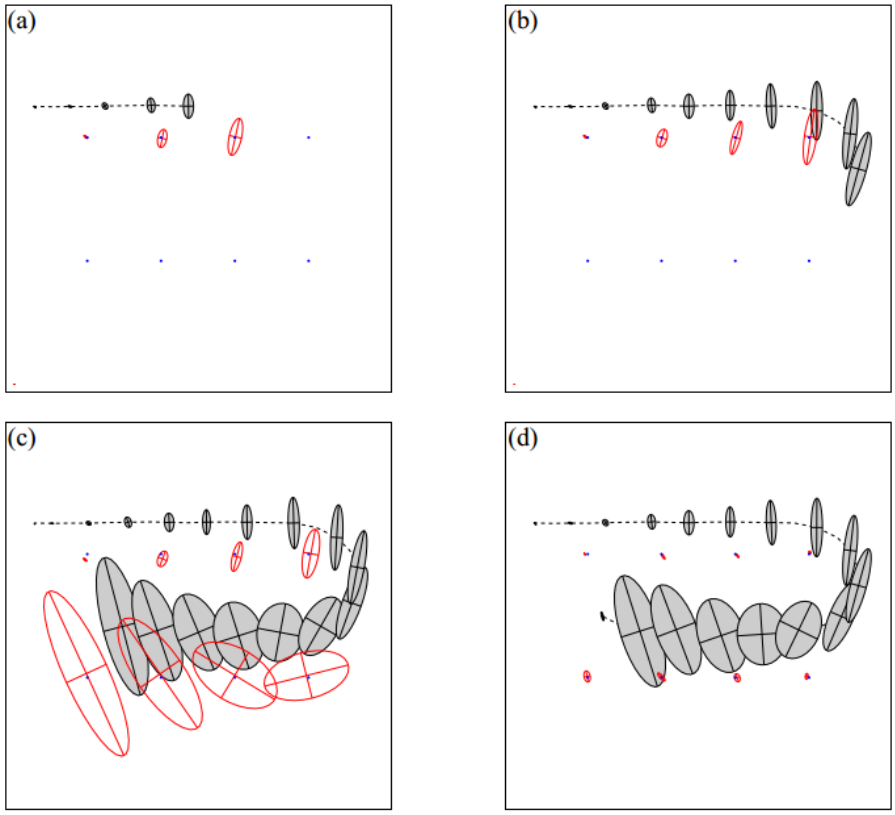

Figure 4.2 Extended Kalman Filter applied to the on-line SLAM problem.

The dotted in is the robot's path, shaded ellipses show the estimations of its own position. The dots mean eight distinguishable landmarks of unknown location, and their location estimations are white ellipses. On the figures (a)–(c) the positional uncertainty of robot is increasing, as is its uncertainty about the landmarks it encounters. When the robot senses the first landmark again (figure (d)) the uncertainty of all landmarks decreases. The uncertainty of the current pose decreases as well.

One of the simplest and most productive SLAM implementations is based on particle filter, also called Monte Carlo Localization (MCL). Particle filters are mathematical models that represent probability distribution as a set of discrete particles which occupy the state space. Particle filters are a way to efficiently represent non-Gaussian distribution. Monte Carlo localization is a particle-filter based implementation of recursive Bayesian filtering for robot localization [7,8]. On the each iteration of MCL, the likelihood function is evaluated at sample points that are randomly distributed according to the posterior estimate of the robot location.

The principle of MCL is the following:

1) The map of the surrounding space is a two-dimensional array of single-byte variables. Initially, the map is empty.

2) Initialize the initial position of the robot.

3) Read the values from the range finder and maps obstacles according to the obtained measurements.

4) Next, proceed in an infinite loop.

a) Use odometry to predict how much the robot have moved relative to the previous measurement.

b) Read the values from the range finder.

c) Calculate the most probable position of the robot using the particle filter for the current map. One particle contains the position and angular orientation of the robot. Probability of a particle is calculated based on the difference between the actual readings of range finder and predicted value for the given particle.

d) Assuming that the robot is in the most probable position, update the current map based on readings from the range finder.

The Monte Carlo localization process is shown in Figure 4.3 (from [41]).

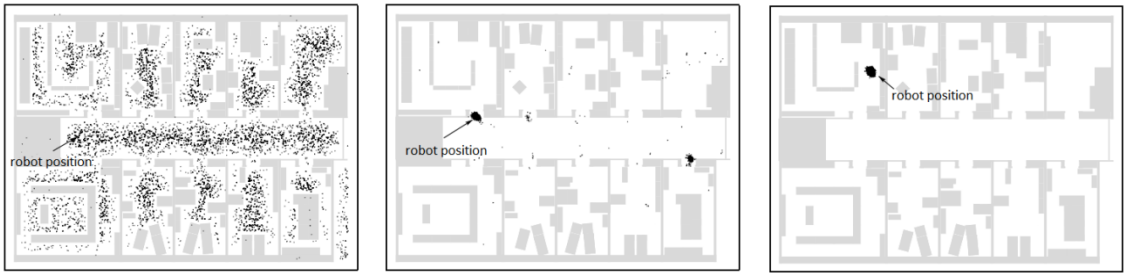

Figure 4.3 Monte Carlo Localization

On the left picture the robot is globally uncertain and the samples are spread uniformly. On the middle picture the robot moved about 1 meter and

the ambiguity is almost resolved. The right picture shows samples that now centered tightly around the correct position, so robot knows where he is [41].

Other methods based on Expectation Maximization are iterative methods for finding maximum likelihood that offer an optimal solution, being an ideal option for map-building, but not for localization. The Expectation Maximization algorithm is able to build a map when the robot's pose is known, for instance, by means of expectation [6,9].

Table 4.1 from [40] shows the advantages and disadvantages of the methods applied into the SLAM framework.

Table 4.1 SLAM filtering approaches

Advantages	**Disadvantages**
Kalman Filter and Extended KF (KF/EKF)	
- high convergence - handle uncertainty	- Gaussian assumption - slow in high dimensional maps
Compressed Extended KF (CEKF)	
- reduced uncertainty - reduction of memory usage - handle large areas - increase map consistency	- require very robust features - data association problem - require multiple map merging
Information Filters (IF)	
- stable and simple - accurate - fast for high dimensional maps	- data association problem - may need to recover a state estimates - in high-D is computationally expensive
Particle Filter (PF)	
- handle nonlinearities - handle non-Gaussian noise	- growth in complexity
Expectation Maximization (EM)	
- optimal to map building - solve data association	- inefficient, cost growth - unstable for large scenarios - only successful in map building

Generally, the advantages of probabilistic robotics are its robustness (the only known methods to perform real-world SLAM) and weaker requirements on sensors and models (because we know they are not perfect). On the other side its drawbacks are the computational complexity (because we consider

more information) and the need to make approximations (it is not feasible to compute exact posterior distributions for continuous worlds).

4.3 Application of Constraint reasoning to Robot Localization

The localization of the robot implies the definition of the robot's current location and orientation. The sensor's measurements provide the information about the environment. We propose a method that uses constraint reasoning for computing the enclosure of all scenarios consistent with the constraint model. We also apply the probabilistic constraint approach to add likelihood information and support reliable solutions that can be useful for the robot navigation.

The distribution of the error of sensor data can be included and propagated into the probabilistic characterization of the scenarios that remain in according to the updated model limitations.

The problem of localization in a probabilistic aspect can be seen as the problem of determining the density of the distribution function of the position of the robot. There is a need to reduce the probability to a small space region.

We assume that the map of the environment is known. A map of the environment is a list of objects and their locations. The walls and the objects on the map can be presented as segments with the coordinates of the beginning and the end of the segment.

The robot pose is defined on the coordinate plane with its (x,y) coordinate and an angle α that is the orientation of a robot, often called bearing, or heading direction (Figure 4.4).

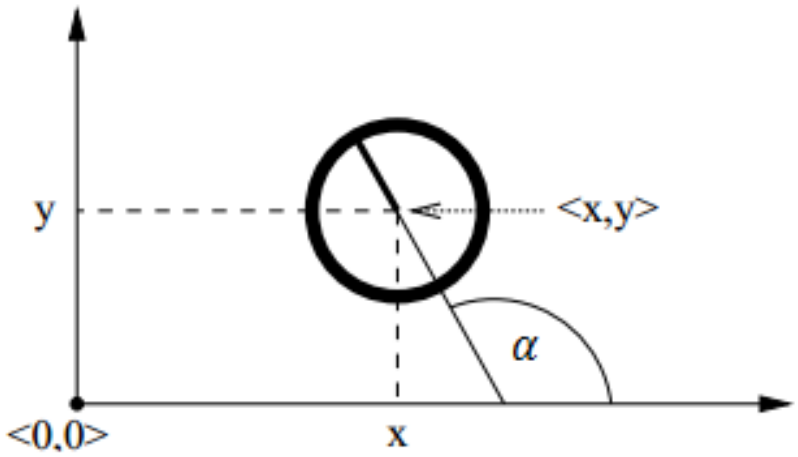

Figure 4.4 Robot pose on the global coordinate system

The application of continuous constraint programming and Monte Carlo integration to the robot localization problem is based on the defining the grid over the region, applying the constraint propagation technique and obtaining the probability over the grid. It can be represented by the following steps:

1) Obtain map of the environment;
2) Set a grid on the map;
3) Apply constraint propagation technique
4) Get probability of the box by Monte Carlo integration method:
   a) Set N random points inside the box
   b) Compute the value of pdf $f(x_i)$ for each point
   c) Estimate the mean of f using the average of these samples:

$$\frac{\sum_{i=1}^{N} f(x_i)}{N}; \tag{4.1}$$

   d) Multiply with the volume to get an approximation of the integral:

$$P(X) = \frac{\sum_{i=1}^{N} f(x_i)}{N} vol(X); \tag{4.2}$$

5) Normalize the computed probabilities

We will use a simplification of the algorithm from [6] that computes a grid over the feasible region and calculates the probability distribution conditioned to the model constraints.

Algorithm 4.1 computes a conditional probability distribution of the variables random vector $X = X_1, \ldots, X_n$ given the event that satisfies all constraint in C. The result of the algorithm is an n-dimensional matrix M representing the conditional probability at each grid cell.

A grid is specified by the spacings of its dimensions, which are computed based on a specified number of partitions for each dimension. The input variable G is an n-dimentional array that defines the number or partitions considered at each dimention.

gridConditionalDistribution$(D, C, split, eligible, order, stop, G)$
Input: D: initial domains box; C: set of constraints; *eligible*, *stop*: predicate;
split, *order*: function; M: array of n integers
Output: M: n-dimentional matrix of intervals;
1 $S \leftarrow Branch\&Prune(D, C, split, eligible, order, stop)$;
2 $\forall_{1 \le idx_1 \le G[1]} \ldots \forall_{1 \le idx_n \le G[n]} \quad M[idx_1] \ldots [idx_n] \leftarrow 0$;
3 $P \leftarrow 0$;
4 **for each** $(B \in S)$ **do**
5 $\langle idx_1, \cdots, idx_n \rangle \leftarrow getIndex(\mathrm{B})$;
6 $M[idx_1] \ldots [idx_m] \leftarrow integrate(B)$;
7 $P \leftarrow P + M[idx_1] \ldots [idx_m]$;
8 **end**
9 $\forall_{1 \le idx_1 \le G[1]} \ldots \forall_{1 \le idx_n \le G[n]} \quad M[idx_1] \ldots [idx_n] \leftarrow \frac{M[idx_1] \ldots [idx_n]}{P}$
10 **return** M;

Algorithm 4.1 Computation of the probability distribution

The Algorithm 4.1 first computes a grid box cover S for the feasible space of the model constraint C (line 1). Function $Branch\&Prune$ is used with a grid oriented parameterization, i.e., $split$ splits the boxes in the grid and $eligible$ chooses boxes that are not yet inside a grid box. The $stop$ predicate requires that

the algorithm only stops when there are no more eligible boxes and $order$ induces a depth first search.

Then the matrix M of conditional probability distributions is initialized to zero (line 2) as well as the normalization factor P that will contain, in the end of the process, the overall sum of all non normalized parcels (line 3). For each box B in the cover S (lines 4-8), its corresponding index of the matrix cell is identified (line 5) and its probability is computed by function *integrate* and assigned to the value in that cell (line 6). The normalization factor is updated (line 7) and used in the end to normalize the computed probabilities (line 9).

4.3.1 Computing Probabilities with Monte Carlo integration

For the calculation of the probability (function *integrate* of Algorithm 4.1) we will use the Monte Carlo integration method that was discussed in the Chapter 3. We calculate the probability of robot being inside a box. Here the sampling will be implemented inside the box within a grid cell, which is defined by the coordinates $[x_0, x_1] \times [y_0, y_1] \times [\alpha_0, \alpha_1]$. Algorithm 4.2 calculates the probability over the box of the grid.

simpleMonteCarloProbability($x_0, x_1, y_0, y_1, \alpha_0, \alpha_1, N,$ $randomGenerator, getPdfValue$)
Input: x_0, x_1, y_0, y_1: coordinates; $\alpha_0\alpha_1$: angles; N: number of samples;
$randomGenerator$, $getPdfValue$: function;
Output: P: probability;
1 $P \leftarrow 0$;
2 **while** $(i < N)$ **do**
3 $x_i \leftarrow randomGenerator(x_0, x_1)$;
4 $y_i \leftarrow randomGenerator(y_0, y_1)$;
5 $\alpha_i \leftarrow randomGenerator(\alpha_0, \alpha_1)$;
6 $P \leftarrow P + getPdfValue(x_i, y_i, \alpha_i)$;
7 **end**
8 $volume \leftarrow (x_1 - x_0)(y_1 - y_0)(\alpha_1 - \alpha_0)$;
9 $P \leftarrow P \cdot volume/N$;
10 **return** P

Algorithm 4.2 Monte Carlo integration for probability calculation

For the defined number of samples $randomGenerator$ generates the points (lines 3-5). Function $getPdfValue$ calculates the value of the pdf for each point and accumulates its contribution (line 6) . The volume is calculated (line 8) and the probability estimate is obtained in line 9.

A common probability distribution for a single measurement error is the one-dimensional normal distribution with 0 mean and variance σ^2. In this case, the probability density function is the following Gaussian function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} \tag{4.3}$$

The above normal distribution assumes that x is a scalar value. Often, x will be a multidimensional vector resulting from several independent measurements. Normal distributions over vectors are called multivariate. Multivariate normal distributions with 0 mean and identical variance σ^2 for each dimension are characterized by density functions of the following form:

$$f(x) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_i {x_i}^2} \tag{4.4}$$

Algorithm 4.3 computes the value of probability density function for the current pose (x, y, α) of the robot and n ladar measurements.

getPdfValue(x, y, α)
Input: x, y: coordinates; α: angle; err: specified value of the error;
n: number of ladar measurements; $getDistance$: function;
Output: $pdfValue$: value of probability density function;
1 $\sigma \leftarrow err/3$;
2 $sum \leftarrow 0$; $i \leftarrow 1$:
3 **while** $(i \leq m)$ **do**
4 $\alpha_i \leftarrow \alpha + ladarAngle_i$;
5 $predicted_i \leftarrow getDistance(x, y, \alpha_i)$;
6 $observed_i \leftarrow ladarDistance_i$;
7 $sum \leftarrow sum + (predicted_i - observed_i)^2$;
8 **end**
9 $pdfValue \leftarrow e^{-\frac{sum}{2\sigma^2}} / \left(\sigma\sqrt{2\pi}\right)^n$;
10 **return** $pdfValue$;

Algorithm 4.3 Value of probability density function calculation

Here we consider n ladar measurements. The direction of the robot is described by the angle α, to which we add the angle $ladarAngle_i$ of the related measurement (line 4). The predicted value of the distance is calculated with the function $getDistance$ (line 5). The observed value is simply the value of the distance obtained by the ladar (line 6). The difference between predicted and observed value is the error committed in measurement i and its square is accumulated in line 7. The value of the corresponding probability density function is obtained in line 9.

Algorithm 4.4 represent the function $getDistance$, it basically computes the distance from the robot pose to the closest object in the direction α.

getDistance(x, y, α)
Input: x, y: coordinates; α: angle; $intersectWall$: function;
Output: D : distance;
1 $P_0 \leftarrow (x, y)$;
2 $P_1 \leftarrow (x + maxD \cdot \cos\alpha\,, y + maxD \cdot \sin\alpha)$;
3 $D \leftarrow intersectWall(P_0, P_1)$;
4 **return** D

Algorithm 4.4 Distance from robot to the closest object

The coordinates of the robot are assigned to the point P_0 (line 1). The point P_1 is located at distance $maxD$ from the point P_0 along the direction α (line 2). This *maxD* is the maximum range of the ladar. The function $intersectWall$ finds the distance D to the closest wall returning *maxD* if it could not intersect any wall (line 3).

The representation is illustrated in the Figure 4.5. The robot is located in the point P_0. In the simulation we obtain the vector of measurements. If the wall is not detected, then measurement has value $maxD$, otherwise the distance from robot till obstacle is calculated.

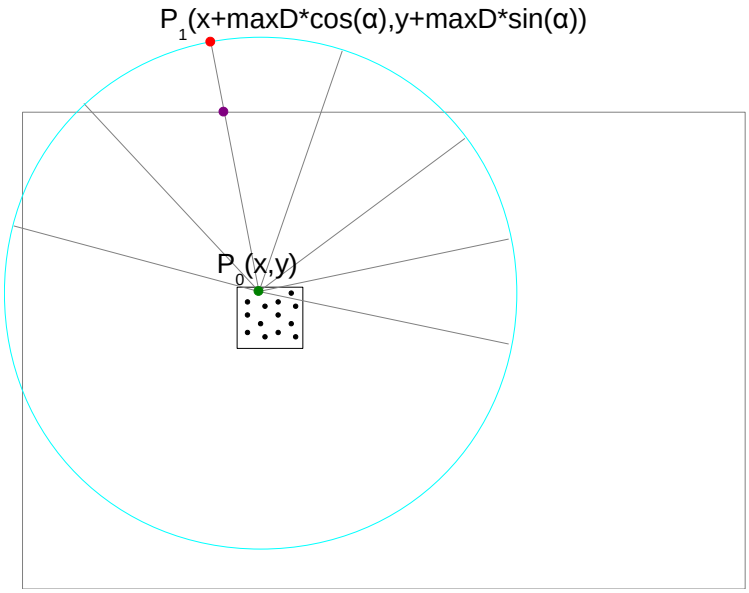

Figure 4.5 Robot pose and measurements.

4.3.2 Pruning Domains with Constraint Programming

Function *Branch&Prune* is used in Algorithm 4.1 (line 1) to reduce the search space, avoiding the application of the Monte Carlo integration over all grid cells and eventually restricting it to a small number of boxes consistent with the ladar measurements and the map of the environment. This function, generally represented in Algorithm 3.1 requires a specialized constraint propagation algorithm to process adequately the information gathered by the sensors in robot localization problems. Algorithm 4.5 *robotCPA* prunes a domains box accordingly to the ladar measurements.

robotCPA$([x_0, x_1] \times [y_0, y_1] \times [\alpha_0, \alpha_1])$
Input: x_0, x_1, y_0, y_1: coordinates; α_0, α_1: angles; $canSeeWalls$: function;
Output: B: box;
1 $C \leftarrow \emptyset$; $i \leftarrow 1$;
2 **while** $(i \leq n)$ **do**
3 $[\alpha_i] \leftarrow [\alpha_0, \alpha_1] + ladarAngle_i$;
4 $S \leftarrow canSeeWalls([x_0, x_1] \times [y_0, y_1] \times [\alpha_i], maxD)$;
5 **if** $(S=\emptyset)$

6 $C \leftarrow C \cup \{\|maxD - ladarDistance_i\| \leq err\}$;
7 **end**
8 **if** (S=$\{\langle(x_0, y_0), (x_1, y_1)\rangle\}$) and $x_0 \neq x_1$
9 $m \leftarrow \frac{y_1 - y_0}{x_1 - x_0}$; $l_i \leftarrow ladarAngle_i \times \pi/180$;
10 $C \leftarrow C \cup \left\{\left\|\frac{\boldsymbol{y} - m\boldsymbol{x} + m(x_0 - y_0)}{m \times cos(l_i + \boldsymbol{\alpha}\pi/180) - sin(l_i + \boldsymbol{\alpha}\pi/180)} - ladarDistance_i\right\| \leq err\right\}$;
11 **end**
12 **if** (S=$\{\langle(x_0, y_0), (x_1, y_1)\rangle\}$) and $y_0 \neq y_1$
13 $m \leftarrow \frac{x_1 - x_0}{y_1 - y_0}$; $l_i \leftarrow ladarAngle_i \times \pi/180$;
14 $C \leftarrow C \cup \left\{\left\|\frac{\boldsymbol{x} - m\boldsymbol{y} + m(y_0 - x_0)}{m \times sin(l_i + \boldsymbol{\alpha}\pi/180) - cos(l_i + \boldsymbol{\alpha}\pi/180)} - ladarDistance_i\right\| \leq err\right\}$;
15 **end**
16 **end**
17 $B \leftarrow CPA([x_0, x_1] \times [y_0, y_1] \times [\alpha_0, \alpha_1], C)$;
18 **return** B

Algorithm 4.5 *robotCPA* for pruning a domains box accordingly to the ladar measurements

It computes a set of numerical constraints C (initialized in line 1) that can be enforced over the variables of the problem $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha})$ and then calls the CPA function (line 17) to reduce the box domains. The constraints may result from each one of the n ladar measurement processed in the while loop (lines 2 to 16). Firstly an enclosure $[\alpha_i]$ for the possible angle of vision (in global coordinates) associated with the ladar measurement i is computed (line 3). Next, a specialized geometric function $canSeeWalls$ is used to determine which of the walls in the map can eventually be seen by a robot positioned in $(x, y) \in [x_0, x_1] \times [y_0, y_1]$ with an angle of vision $\alpha \in [\alpha_i]$ and a vision range up to $maxD$.

If no wall can be seen (line 5) the predicted distance is always $maxD$ and a constraint is added (line 6) to enforce the error between the predicted and the ladar measurement not to exceed err.

If it is only possible to see a single non vertical wall (line 8), represented by the segment $\langle(x_0, y_0), (x_1, y_1)\rangle$, an adequate numerical constraint is enforced (line 10) to restrain the error between the ladar measurement and the predicted

distance for a pose $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha})$. This cannot be done in the case of a vertical wall since it would induce a numerical exception (a division by zero in line 9). A similar procedure is adopted to add a new constraint when is only possible to see a single non horizontal wall (lines 12 to 15).

Notice that whenever there is the possibility of seeing more that one wall it cannot be decided which constraint to enforce, and the algorithm proceeds without associating any constraint to the ladar measurement.

Summarizing, the idea of constraint reasoning with Monte Carlo integration in application to robot localization is to define the grid over the map, apply the propagation technique in order to localize the possible solutions, and then apply the Monte Carlo method for obtaining the probability of each grid. As a result we obtain the probability distribution that shows the robot's probability of being located in the exact position.

4.4 Simulation results

The following experiments were carried out on an Intel Core i7-4700MQ at 2.40 GHz. The Algorithms 4.1-4.4 were implemented in C++.

For the simulation we define the simple case of the room with the size 1000x1000 and 2 objects inside the room, defined as the segments. Let pose the robot in the middle of the room with the bearing equal to 0 (robot "looks" to the right). Then we define the size of the measurement vector that is equal to 7, so we obtain the measurements $z = \{z_1, .., z_7\}$. We define the error of the measurements equal to 5. As we can see from the Figure 4.6 some of the measurements intersect the objects.

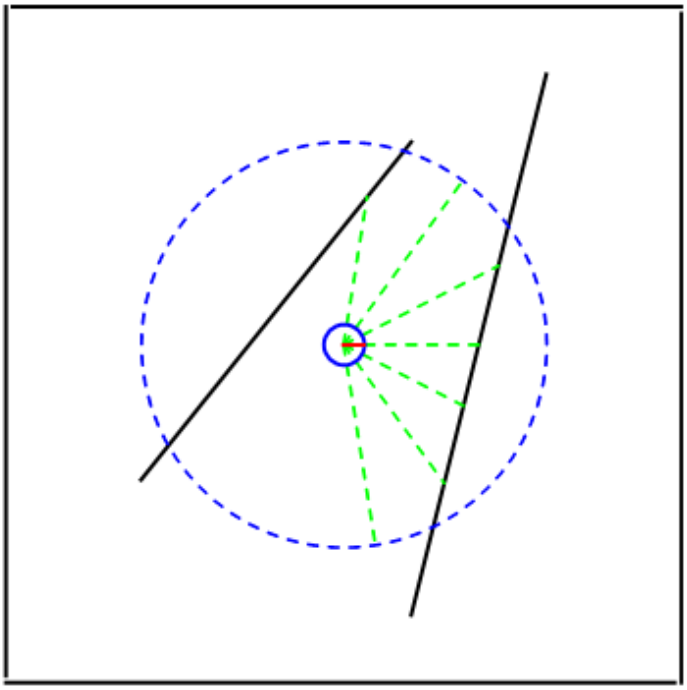

Figure 4.6 Simulation of the robot position

Then we define the grid. For our case it will be equal 100x100 for the x, y coordinates and 360 for the angle (each box has dimension 10x10x1°).

Algorithm 4.1 divides the space into the grid and applies constraint propagation techniques to reduce the space. The resulting boxes are shown on the Figure 4.7. Those boxes represent the possible position of the robot.

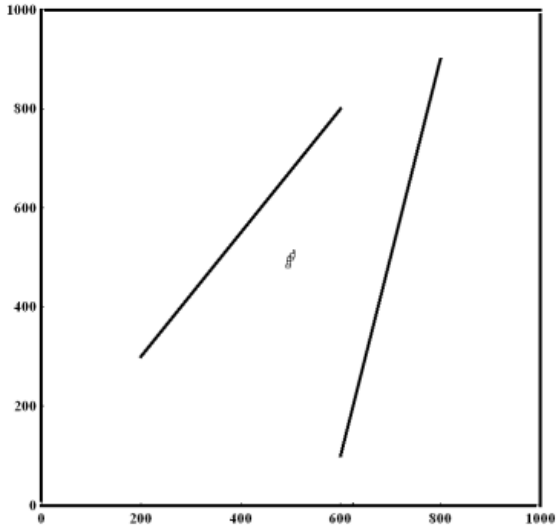

Figure 4.7 Simulation results. Reduced space

The Monte Carlo integration sets random points inside the box as it shown on Figure 4.5. Algorithms 4.2-4.4 are implemented in order to obtain the probability over the box. The obtained probability is shown in the Figure 4.8. The calculations take 8 seconds.

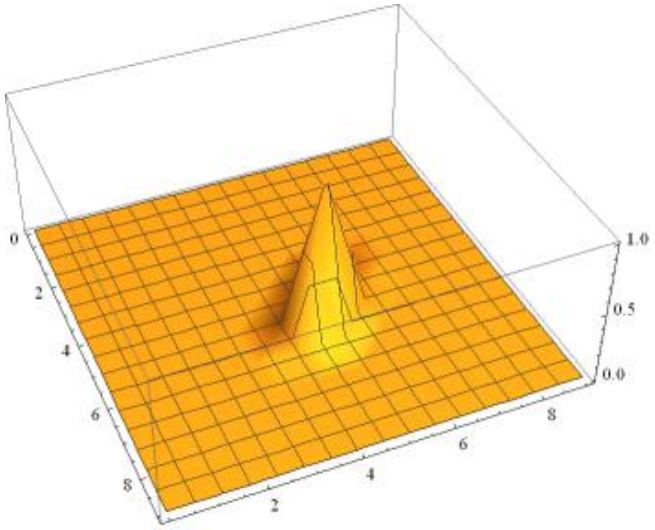

Figure 4.8 Simulation results. Probability distribution

Further we present several experiments with different configurations. The simulation results are presented in Figure 4.9. This is the simple case. The boxes are obtained on the small region and the probabilities are calculated for that boxes.

Figure 4.9 Simulation example 1. Constraint reasoning with Monte Carlo.

The example from the Figure 4.10 shows not that trivial result. We can see that robot can be positioned in different spaces in the room and the probability if being in the different stop is different. So we obtain higher probability along the right wall of the room.

Figure 4.10 Simulation example 2. Constraint reasoning with Monte Carlo.

As we can see from the figure above our approach of probabilistic constraint reasoning with Monte Carlo integration gives the accurate results.

In the following experiment we assume a map of the environment such as the one shown in Figure 4.11. The robot is placed in one of the four rooms and detects the objects. The algorithm that uses constraint reasoning with Monte Carlo method computes probability of the robot pose. As we can see from the figure the probability of the robot pose is equal for each room.

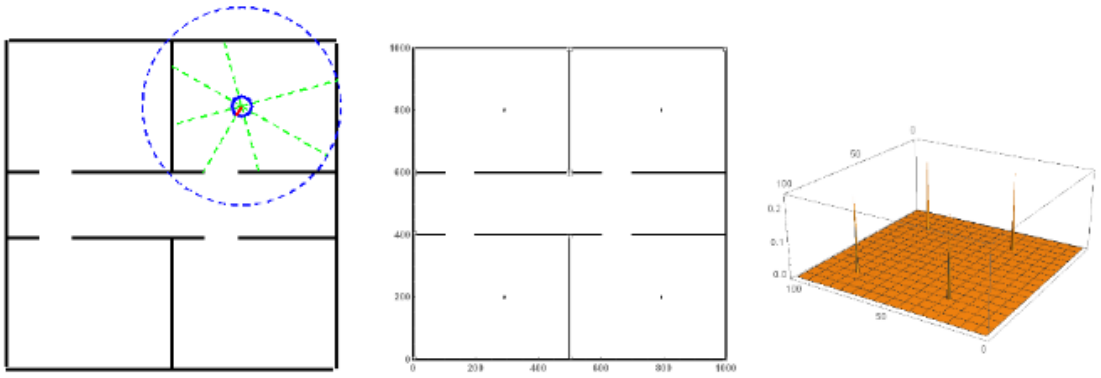

Figure 4.11 Simulation example 3. Constraint reasoning with Monte Carlo.

Placing the robot in the corridor gives the results shown on Figure 4.12. The algorithm calculates the possible poses of the robot as well as the probability of this poses.

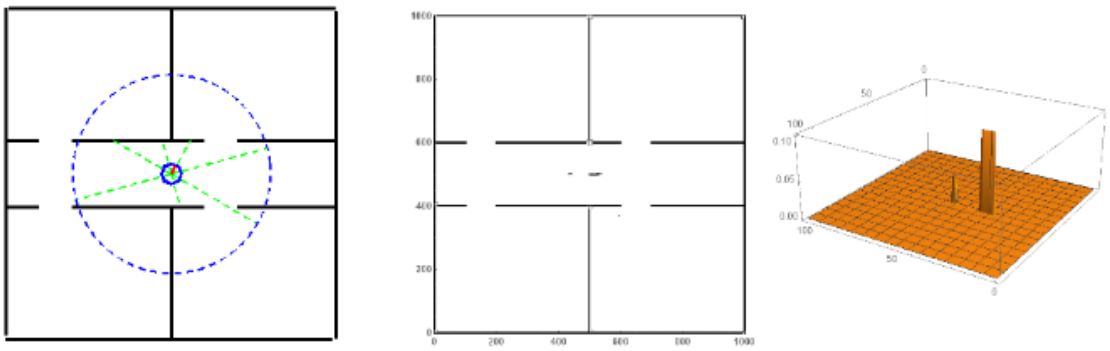

Figure 4.12 Simulation example 4. Constraint reasoning with Monte Carlo.

More complicated environment is shown in Figure 4.13. We simulate the robot robot placing it in the middle of the room. The measurements define the unique pose of the robot. Different options of the robot pose are shown in Figures 4.13-4.18.

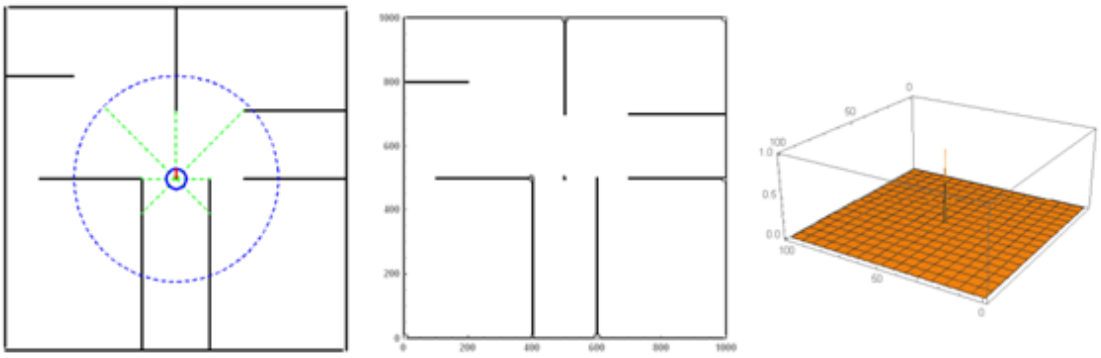

Figure 4.13 Simulation example 5. Constraint reasoning with Monte Carlo.

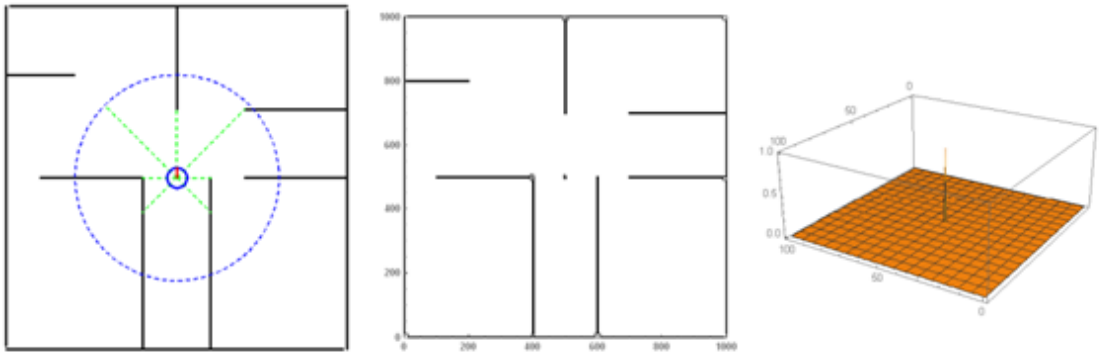

Figure 4.13 Simulation example 5. Constraint reasoning with Monte Carlo.

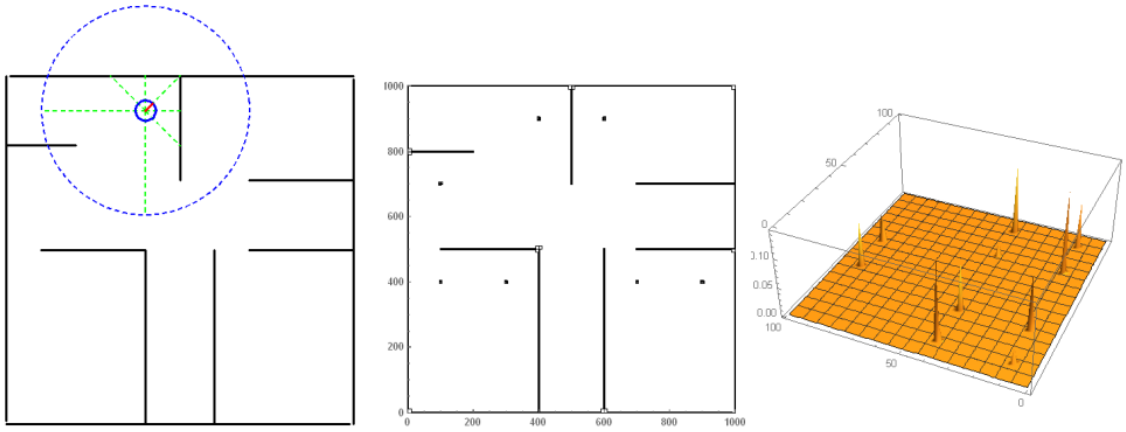

Figure 4.14 Simulation example 6. Constraint reasoning with Monte Carlo.

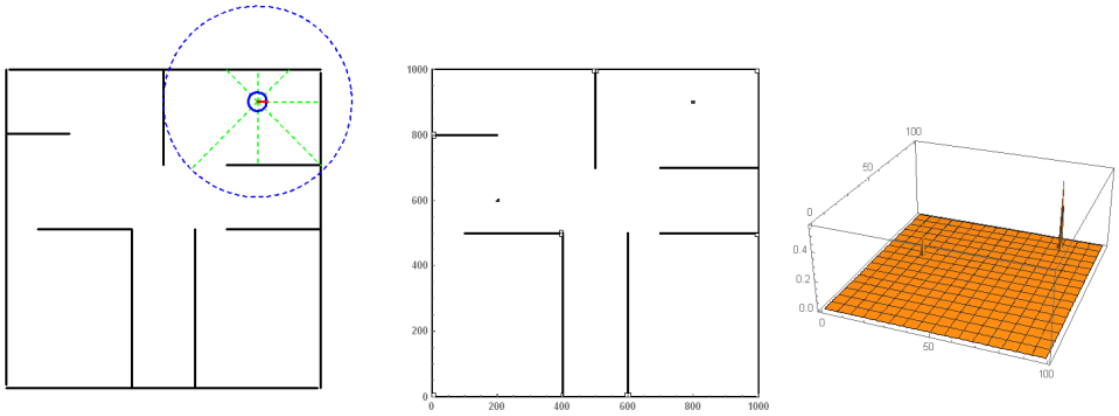

Figure 4.15 Simulation example 7. Constraint reasoning with Monte Carlo.

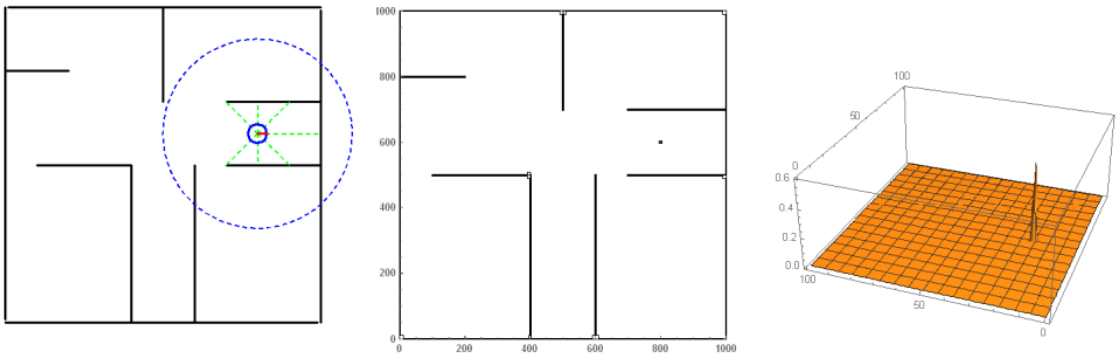

Figure 4.16 Simulation example 8. Constraint reasoning with Monte Carlo.

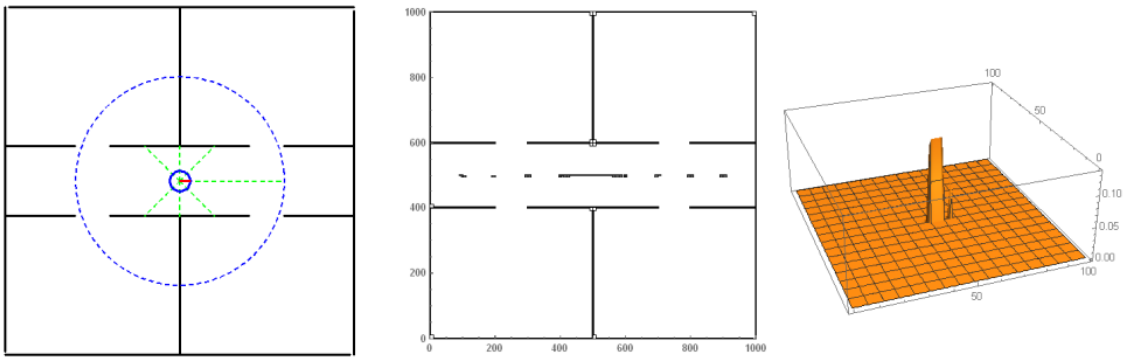

Figure 4.17 Simulation example 9. Constraint reasoning with Monte Carlo.

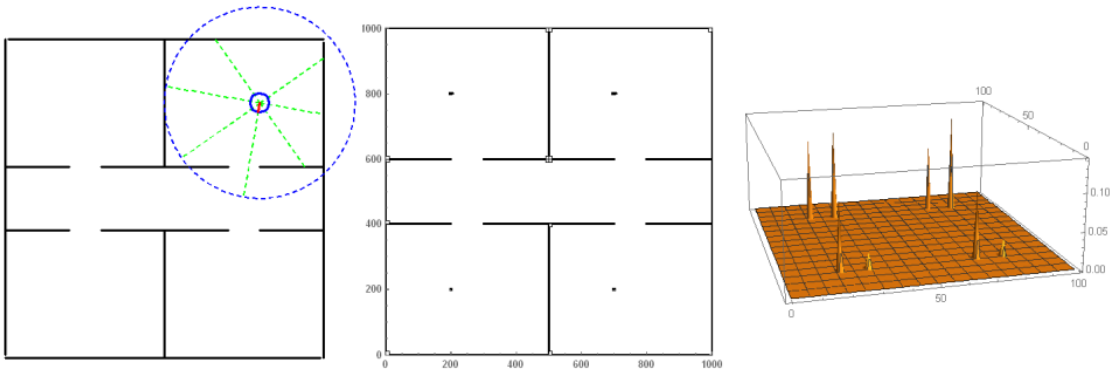

Figure 4.18 Simulation example 10. Constraint reasoning with Monte Carlo.

From the following tests we conclude that proposed technique allow solve the simple localization problem. The propagation method accelerates the calculation of the robot coordinates and the Monte Carlo integration gives the probability of the robot pose. The experiments show the efficiency and

convenience of the method. However, time spend each example is 4-12 seconds that requires improvements of algorithms performance.

4.5 Conclusion

Mobile robots can change their location thought locomotion. For a mobile robot one of the most important capabilities is ability to navigate. We study the localization problem and use the developed in the previous Chapter techniques for developing the algorithms of mobile robot localization with probabilistic constraint reasoning.

We discussed several aspects of the localization problem and methods for robot localization. Then we defined the type of the map representation that is more convenient for our task. Further, we define the type of sensors and representation of the data from them.

We developed the algorithm for robot localization using Probabilistic constraint reasoning with Monte Carlo integration. This method allows to determine the position of robot and with its probability. Experiments shows, that robot can easily determine its position on the map.

5

Conclusions and Future Work

5.1 Conclusions

In this dissertation the probabilistic constraint reasoning tool with Monte Carlo integration and its application to robot localization problem were proposed. The probabilistic constraint framework can be used as an effective tool for dealing with uncertainty in the localization problem. We used the approach that bridges the gap between pure safe reasoning and pure probabilistic reasoning.

First, we introduced the basic notions of Continuous Constraint Programming and probability theory, and quadrature methods and presented the basic algorithms. Continuous Constraint programming is used to solve a wide range of problems, including integration. We studied the efficiency of quadrature methods, such as Monte Carlo and its modifications.

Second, we proposed technique for integration over constraint region that computes close estimates of the correct integral value. For this task we apply Monte Carlo integration techniques combined with constraint programming. In

comparison with pure Monte Carlo integration, our method is faster and requires less sampling, due to constraint programming techniques that reduce the sample space into a enclosure of the integration region.

Third, we described various techniques for integrating using Monte Carlo methods and its modifications combined with constraint propagation techniques.

Fourth, we studied the mobile robots, their technical characteristics, types of sensors, motors. We defined the reasons of arising of uncertainty in robotics, talked about probabilistic robotics and denoted several approaches for Simultaneous Localization and Mapping.

Finally, we developed the algorithm for robot localization that is based on continuous constraint programming with Monte Carlo integration method. The simulation shows that proposed approach is efficient and can be applied to the real systems. The probabilistic reasoning with Monte Carlo integration minimizes the uncertainty and can be successfully applied for robot localization problem.

5.2 Future work

There are many possibilities for the development of the proposed technique to robotics and related areas. One of the possible directions of the future work is the development Simultaneous Localization and Mapping algorithm based on the proposed techniques.

Moreover there is possibility to develop the modifications of the algorithm in order to improve efficiency.

Bibliography

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. 2005, pp. 1999–2000.

[2] J. Cruz, *Constraint Reasoning for Differential Models*. IOSPress, 2005, p. 244.

[3] E. Carvalho, J. Cruz, and P. Barahona, "Reasoning with Uncertainty in Continuous Domains," pp. 1–12.

[4] T. W. F. Rossi, P. van Beek, "Handbook of Constraint Programming," vol. 2, 2006.

[5] P. Van Hentenryck and V. Saraswat, "Constraint programming: Strategic directions," *Constraints*, vol. 33, pp. 7–33, 1997.

[6] E. Carvalho, J. Cruz, and P. Barahona, "Probabilistic constraints for nonlinear inverse problems," *Constraints*, vol. 18, no. 3, pp. 344–376, Feb. 2013.

[7] E. Davis, "Constraint propagation with interval labels," *Artif. Intell.*, vol. 32, no. 3, pp. 281–331, 1987.

[8] J. C. Cleary, "Logical arithmetic," *Futur. Comput. Syst.*, vol. 2, pp. 125–149, 1987.

[9] E. Hyvonen, "Constraint Reasoning Based on Interval Arithmetic," *Proc. IJCAI-89 (Morgan Kaufmann, Los Altos, USA)*, pp. 1193–1198, 1989.

[10] T. Sunaga, "Theory of an interval algebra and its application to numerical analysis," *RAAG Mem. 2*, pp. 29 – 46, 1958.

[11] R. Moore, *Methods and applications of interval analysis*. Philadelphia: SIAM, 1979.

[12] J. Rohn, "A handbook of results on interval linear problems," *Inst. Comput. Sci. Acad. Sci. Czech Republic, Prague, 2005-2012. – Tech. Rep. No. V-1163.*

[13] G. Walster and E. Hansen, *Global Optimization Using Interval Analysis*. New York: Marcel Dekker, 2004.

[14] E. Carvalho, J. Cruz, and P. Barahona, "Probabilistic Continuous Constraint Satisfaction Problems," *2008 20th IEEE Int. Conf. Tools with Artif. Intell.*, vol. 2, 2008.

[15] A. Goldsztejn, J. Cruz, and E. Carvalho, "Convergence analysis and adaptive strategy for the certified quadrature over a set defined by inequalities," *J. Comput. Appl. Math.*, Sep. 2013.

[16] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, 2nd ed. New York: Academic Press, 1984, p. 612.

[17] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, "Numerical recipes in C: the art of scientific computing, 1992," *Cité en*, 1992.

[18] T. Gerstner and M. Griebel., "Sparse grids," *Encycl. Quant. Financ. R. Cont (ed.), Wiley*, 2008.

[19] T. Gerstner and M. Griebel, "Numerical integration using sparse grids," *Numer. algorithms*, pp. 1–26, 1998.

[20] H. Niederreiter, *Random number generation and Quasi-Monte Carlo methods*. Society for Industrial Mathematics, 1992.

[21] M. Colbert, S. Premoze, and G. François, "Importance Sampling for Production Rendering," *SIGGRAPH 2010 Course …*, 2010.

[22] Y. Zhang and A. K. Mackworth, "Specification and verification of constraint-based dynamic systems," *Princ. Pract. Constraint Program. number. Springer-Verlag*, vol. 874 in LNC, pp. 229 – 242, 1994.

[23] A. K. Mackworth and Y. Zhang, "Constraint programming in constraint nets," *Princ. Pract. Constraint Program. Newport Pap. MIT Press. Cambridge, MA.*, pp. 49–68., 1995.

[24] D. K. Pai, "Least constraint: A framework for the control of complex mechanical systems," *Proc. Am. Control Conf. Boston, MA*, pp. 1615–1621, 1991.

[25] Y. Zhang and A. K. Mackworth, "Synthesis of hybrid constraint-based controllers," *Hybrid Syst. II. Springer Verlag*, vol. 999 in LNC, pp. 552 – 567, 1995.

[26] A. Mackworth, "Constraint-based design of embedded intelligent systems," *Constraints*, vol. 86, pp. 83–86, 1997.

[27] H. Zhang, "Interval importance sampling method for finite element-based structural reliability assessment under parameter uncertainties," *Struct. Saf.*, vol. 38, pp. 1–10, Sep. 2012.

[28] V. A. Saraswat, R. Jagadeesan, and V. Gupta., "Timed Default Concurrent Constraint Programming," *J. Symb. Comput. To Appear. Ext. Abstr. Appear. Proc. ACM Symp. Princ. Program. Lang. San Fr.*, 1995.

[29] R. S. Stansbury and A. Agah, "A robot decision making framework using constraint programming," *Artif. Intell. Rev.*, vol. 38, no. 1, pp. 67–83, May 2011.

[30] N. Berger, R. Soto, A. Goldsztejn, and P. Cardou, "Finding the Maximal Pose Error in Robotic Mechanical Systems Using Constraint Programming," 2010.

[31] A. Otero, P. Félix, C. Regueiro, M. Rodríguez, and S. Barro, "Fuzzy constraint satisfaction approach for landmark recognition in mobile robotics," *AI Commun.*, 2006.

[32] M. Fromherz and T. Hogg, "Modular robot control and continuous constraint satisfaction," *… Probl. with Constraints*, 2001.

[33] D. Chablat, P. Wenger, F. Majou, and J.-P. Merlet, "An Interval Analysis Based Study for the Design and the Comparison of Three-Degrees-of-Freedom Parallel Kinematic Machines," *Int. J. Rob. Res.*, vol. 23, no. 6, pp. 615–624, Jun. 2004.

[34] J.-P. Merlet, "Solving the Forward Kinematics of a Gough-Type Parallel Manipulator with Interval Analysis," *Int. J. Rob. Res.*, vol. 23, no. 3, pp. 221–235, Mar. 2004.

[35] D. Daney, "Interval methods for certification of the kinematic calibration of parallel robots," *Robot. Autom. …*, no. April, pp. 1913–1918, 2004.

[36] T. Braünl, "Embedded robotics," *Springer, ISBN*, p. 458, 2006.

[37] V. N. Meshcheryakov and O. V. Meshcheryakova, "Mathematical vector model of induction motor and structural-topological analysis of the model," *News high Educ. institutes Chernozemya*, vol. 2, 2014.

[38] V. N. Meshcheryakov, O. V. Meshcheryakova, and P. V. Saraev, "Mathematical modeling and process control in industrial automation systems using complex-valued neural networks," *Probl. Upr.*, vol. 6, pp. 71–75, 2013.

[39] T. Bailey and H. Durrant-Whyte, "Simultaneous localisation and mapping (slam) part 2: State of the art," *Robot. …*, pp. 1–10, 2006.

[40] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, "The SLAM problem: a survey.," *CCIA*, 2008.

[41] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots Dieter Fox, Wolfram Burgard," *Proc. Natl. Conf. Artif. Intell.*, vol. 113, p. 114, 1999.