Algorithmic and Architectural Optimization Techniques in Particle Filtering for FPGA-Based Navigation Applications

Biruk Getachew Sileshi

Universitat Autònoma de Barcelona

Departament de Microelectrònica i Sistemes Electrònics

Algorithmic and Architectural Optimization Techniques in Particle Filtering for FPGA-Based Navigation Applications

Biruk Getachew Sileshi

PhD Advisor: Dr. Joan Oliver Malagelada

A Thesis Submitted for the Degree of

PhD in Microelectronics and Electronic Systems

Bellaterra (Barcelona), June 3, 2016

UNIVERSITAT AUTÒNOMA DE BARCELONA

Departament de Microelectrònica i Sistemes Electrònics

DECLARATION

I declare that I am the sole author of this thesis entitled **"Algorithmic and Architectural Optimization Techniques in Particle Filtering for FPGA-Based Navigation Applications"** and that the work contained therein is original, where explicitly stated otherwise in the text.

Biruk Getachew Sileshi

Bellaterra (Barcelona), June 3, 2016

UNIVERSITAT AUTÒNOMA DE BARCELONA

Departament de Microelectrònica i Sistemes Electrònics

ALGORITHMIC AND ARCHITECTURAL OPTIMIZATION TECHNIQUES IN PARTICLE FILTERING FOR FPGA-BASED NAVIGATION APPLICATIONS

Thesis presented to obtain the PhD in Microelectronics and Electronic Systems

Author: BIRUK GETACHEW SILESHI

PhD Advisor: DR. JOAN OLIVER MALAGELADA

Dr. Joan Oliver Malagelada, professor of the Universitat Autònoma de Barcelona

CERTIFY:

that the dissertation **"Algorithmic and Architectural Optimization Techniques in Particle Filtering for FPGA-Based Navigation Applications"** presented by Mr. Biruk Getachew Sileshi to obtain the PhD in Microelectronics and Electronic Systems, has been done under their direction at the Universitat Autònoma de Barcelona.

Dr. Joan Oliver Malagelada

Bellaterra (Barcelona), June 3, 2016

To My Family

Acknowledgements

The work presented in this PhD thesis could not have been done without the supports of many people to whom i would like to express my sincere gratitude.

First and for most i would like to express my deepest gratitude to my supervisor Dr. Joan Oliver for introducing me to this field of research, for all his continuous support and guidance during my PhD study. I would also like to thank Prof. Carles Ferrer for the opportunity to join the Integrated Circuits and Systems Design group and for his financial supports. I am also deeply thankful to Dr. Ricardo Toledo in assisting me with conducting some of the experiments with robots and for the finical support with the publication of some of my work. I am also thankful to all members of the examination committee of the Micro-electronics and Electronic Systems department of UAB for all their useful and helpful comments during my annual PhD evaluations. I would also like to thank the UAB university for the Personal Investigador en Formación (PIF) pre-doctoral grant financial support to conduct this research work.

My sincere thanks also goes to Dr. Jose Gonçalves for all his guidance and support during my stay at the Polytechnic Institute of Bragança (IPB). I would also like to thank all my work colleagues and friends at IPB for all their friendship and good times.

On a personal note, i would like to thank all my work colleagues, friends and the secretarial staffs in the Micro-electronics and Electronic Systems department of UAB. It is my great pleasure to get to know you all: Adriana, Andres, Alex, David, Lu Wang, Natalie, Marc, Raul, Roger and Sadiel. I would like to give my special thanks to Jordi Carrabina for all his technical supports he provided to me.

Finally my special words goes to all my family, specially to my mom, sister and brother for all their unconditional support and encouragement throughout my PhD study.

Abstract

Particle filters (PFs) are a class of Bayesian estimation techniques based on Monte Carlo simulations that are among the estimation frameworks that offer superior performance and flexibility on addressing non-linear and non Gaussian estimation problems. However, such superior performance and flexibility of PFs comes at the cost of higher computational complexity that has so far limited their applications in real time problems.

Most real time applications, in particular in the field of mobile robotics, such as tracking, simultaneous localization and mapping (SLAM) and navigation, have constraints on performance, area, cost, flexibility and power consumption. Software implementation of the PFs on sequential platforms for such applications is often prohibitive for real time applications. Thus to make PFs more feasible to such real-time applications, the acceleration of PFs computations using hardware circuitry is essential. As most of the operations in PFs can be performed independently, pipelining and parallel processing can be effectively exploited by use of an appropriate hardware platform, like field programmable gate arrays (FPGAs), which offer the flexibility to introduce parallelization and lead to a wide range of applications of PFs in real time systems. Thus the focus of this PhD thesis is to address the challenge to deal with the computational complexity of PFs introducing FPGA hardware acceleration for improving their real time performance and make their use feasible in these applications.

For a high throughput hardware realization of the PFs, some of the issues addressed in this thesis include: the identification in the computational bottlenecks of the PFs and the proposal and design of PF hardware acceleration techniques. Based on the PF hardware acceleration techniques, the design and implementation of a PF HW/SW architecture is presented. In addition, a new approach for full parallelization of the PFs is presented which leads to a distributed particle filtering implementation with simplified parallel architecture. Finally, the design of a fully hardware PF processor is provided where the whole particle filtering steps applied to the SLAM problem are proposed for an implementation on FPGA. As part of the PF processor design, important problems for PF in SLAM are also solved. Also, the design and implementation of a parallel laser scanner as a PF co-processor using a Bresenham line drawing algorithm is realized. The proposed hardware architecture has led to the development of the first fully hardware (FPGA) prototype for the PF applied to the SLAM problem.

Resum

Els filtres de partícules (FPs) són una tipologia de tècniques d'estimació bayesiana basades en simulacions Monte Carlo que es troben entre els sistemes d'estimació que ofereixen millors rendiments i major flexibilitat en la resolució de problemes d'estimació no lineals i no gaussians. No obstant això, aquest millor rendiment i major flexibilitat es contraposa amb la major complexitat computacional del sistema, motiu pel que fins ara la seva aplicació a problemes de temps real ha estat limitada.

La majoria de les aplicacions en temps real, en particular en el camp de la robòtica mòbil, com ara el seguiment, la localització i mapatge simultani (SLAM) i la navegació, tenen limitacions en el rendiment, l'àrea, el cost, la flexibilitat i el consum d'energia. La implementació software de FPs en plataformes d'execució seqüencial en aquestes aplicacions és sovint prohibitiu per l'elevat cost computacional. Per tant per aproximar els FPs a aplicacions en temps real és necessària l'acceleració de les operacions de còmput utilitzant plataformes hardware. Donat que la major part de les operacions es poden realitzar de forma independent, el pipeline i el processament en paral · lel poden ser explotats de manera efectiva mitjançant l'ús de hardware apropiat, com ara utilitzant Field Programmable Gate Arrays (FPGAs). La flexibilitat que tenen per introduir la paral · lelització fa que puguin ser emprades en aplicacions de temps real. Amb aquest enfocament, aquesta tesis doctoral s'endinsa en el difícil repte d'atacar la complexitat computacional dels filtres de partícules introduint tècniques d'acceleració hardware i implementació sobre FPGAs, amb l'objectiu d'incrementar el seu rendiment en aplicacions de temps real.

Per tal d'implementar filtres de partícules d'alt rendiment en hardware,aquesta tesis ataca la identificació dels colls d'ampolla computacionals en FPs i proposa, dissenya i implementa tècniques d'acceleració hardware per a FPs. Emprant tècniques d'acceleració hardware per a filtres de partícules primer es dissenya i implementa una arquitectura HW/SW per a FPs. Després, es dissenya un processador hardware per a FPs en el que es detallen totes les etapes del FP aplicant-lo a un algorisme de mapatge i localització simultània i implementant-lo sobre FPGA. També es dissenya i implementa un co-processador paral · lel per a un escàner làser emprat en FPs emprant un algorisme de Bresenham. Aquesta proposta hardware ha conduit al desenvolupament del primer prototip totalment hardware (sobre FPGA) per a filtres de partícules emprats en SLAM.

Contents

List of Figures

List of Tables

List of Abbreviations

AXI4 . . . . . Advanced eXtensible Interface 4

ASIC . . . . . Application Specific Integrated Circuit

ASSP . . . . . Application Specific Standard Product

ALU . . . . . Arithmetic Logic Unit

CORDIC . . . COordinate Rotation DIgital Computer

CPU . . . . . Central Processing Unit

CDF . . . . . Cumulative Distribution Function

DE . . . . . Differential Evolution

DSP . . . . . Digital Signal Processor

EDK . . . . . Embedded Development Kit

EKF . . . . . Extended Kalman Filter

FPGA . . . . . Field Programmable Gate Arrays

FPU . . . . . Floating Point Unit

FSL . . . . . Fast Simplex Link

GPGPU . . . . General Purpose Graphic Processing Unit

GRNG . . . . Gaussian Random Number Generator

HW/SW . . . Hardware/Software

IW . . . . . Importance Weight

ICDF . . . . . Inverse Cumulative Distribution Function

IMH . . . . . Independent Metropolis-Hastings

IP . . . . . Intellectual Property

ISE . . . . . Integrated Software Environment

KLD . . . . . Kullback-Leibler Distance

KF . . . . . Kalman Filter

LRF . . . . . Laser Range Finder

LFSR . . . . . Linear Feedback Shift Register

LMB . . . . . Local Memory Bus

LUTs . . . . . Lookup Tables

MCMC . . . . Markov Chain Monte Carlo

$(MC)^3$ . . . . . Metropolis Coupled Markov Chain Monte Carlo

MCL . . . . . Monte Carlo Localization

MEM . . . . . Memory

OGM . . . . . Occupancy Grid Map

PDF . . . . . Probability Density Function

PE . . . . . Processing Element

PFs . . . . . Particle Filters

PLB . . . . . Processor Local Bus

RMSE . . . . . Root Mean Square Error

RNG . . . . . Random Number Generator

RNA . . . . . Resampling with Non-proportional Allocation

RPA . . . . . Resampling with Proportional Allocation

RPF . . . . . Regularized Particle Filter

S . . . . . Sampling

SDK . . . . . Software Development Kit

SIR . . . . . Sampling Importance Resampling

SLAM . . . . . Simultaneous Localization and Mapping

SMG-SLAM . Scan-Matching Genetic Simultaneous Localization and Mapping

SoC . . . . . Systems on Chip

URNG . . . . Uniform Random Number Generator

UART . . . . . Universal Asynchronous Receiver Transmitter

VHDL . . . . . Very High Speed Integrated Circuit Hardware Description Language

XCL . . . . . Xilinx Cache Link

XPS . . . . . Xilinx Platform Studio

1 Introduction

1.1 Background

Estimating the state of a dynamic system has diverse applications in engineering and scientific areas including communications, machine learning, radar and sonar target tracking, satellite navigation, neuroscience, economics, finance, political science, and many others. In the history of solving the state estimation problems, the introduction of a Kalman filtering (KF) method in the year 1960 was a major breakthrough for estimating the state of linear Gaussian problems [1]. As the basic KF is restricted to linear Gaussian systems, an Extended Kalman filter (EKF) version was introduced in 1980 in order to handle systems with nonlinear model and non-gaussian noise [2]. However, the EKF approximates the nonlinear model by a linear model and then utilise KF to obtain an optimal solution. As a result, the degree of accuracy of the EKF relies on the validity of the linear approximation and is not suitable for highly non-Gaussian conditional probability density functions (PDFs). If the non-linearities are significant, or the noise is non-Gaussian, it diverges and results in poor performance, and the references contained therein [3, 4]. Bearing in mind that KF method is limited by its assumptions, the provision of optimal estimation problems for non-linear/non-Gaussian systems that do not typically rely on analytic solutions is essential. This is where the sequential Monte Carlo

methods called *particle filters* (PFs) come into play, where they approximate the posterior probability of the state through generation of a weighted state samples, called *particles* using Monte Carlo Methods [5].

Since their introduction in 1993, PFs methods have become a very popular class of algorithms to solve the state and the parameter estimation for nonlinear and non-Gaussian system in diverse fields as computer vision, econometrics, robotics and navigation. Their popularity is due to the fact that for any nonlinear/non-Gaussian dynamic estimation problem one can design an accurate and reliable recursive Bayesian filter. Their principal advantage is that they do not rely on any local linearization technique or any crude functional approximation and provide enhanced performance with respect to EKF approximation method [3, 5, 6].

PFs estimate the state based on the principle of importance sampling whereby the particles are sampled from an importance density function (*sampling step*). This sampling step is followed by the *importance weight* computation step where weights are assigned to the individual particles based on the received observation to form the weighted set of particles. The weighted set of particles represent the posterior density of the state and is used to find various estimates of the state. The posterior is then recursively updated in time as new observations become available. *Resampling* is another step of the PF required to avoid the *weight degeneracy* problem [7, 8], which causes the weights of many particles to be negligible after a few time instances.

The non-parametric representation of the posterior densities makes PFs suitable candidate for highly non-linear and / or non-Gaussian estimation problems. Such greater flexibility and estimation accuracy, however, comes at the cost of *large computational complexity*. Due to such computational complexity, the conventional software implementations of PFs for an accurate result in most real time applications is prohibitive and lack enough computational power. As such, the practical use of PFs has been limited especially in hard real-time systems in which a failure to meet the deadline is potentially catastrophic [9]. Therefore, in order to overcome such limitations and make PFs amenable for real-time applications, the speedup of the intensive computations in PFs with an efficient implementation using flexible

hardware / software (HW / SW) platforms is required. As most of the operations in PFs can be performed independently, parallel processing and pipelining can be exploited by use of an appropriate hardware platform, like field programmable gate arrays (FPGAs), considering their suitability for low power, pipelined and parallel hardware designs. However, there are some challenges for a high throughput hardware implementation of the PFs.

At first, due to the wide range of applications to which PF techniques can be applied, the design and implementation of a generic and highly optimized architecture for all PF-based systems is difficult. This is mainly due to the dependency of the implementation upon the properties of the model. Thus it is required to develop an efficient hardware architecture focusing on the high-level data and control flow with the objective to easily incorporate the architecture to other application domains. Second, particle filtering is both computation intensive and memory access intensive. It is characterized as an iterative processing of particles where the number of particles can vary from a few tens to thousands depending on the applications. Of course, a large number of particles increase the precision of the system, however when the number is large also the overall hardware complexity grows rapidly and satisfying real-time constraint becomes very difficult. Third, as the algorithm is a mixture of parallel and sequential operations the direct mapping from algorithm to architecture results in bad performance. The sequential operation results from the resampling step of the particle filter, which inhibits the ability for full parallelization of the PF computations. Resampling is a computationally expensive step that requires a complicated hardware design. Thus, it can greatly impact the speed and resource utilization in PFs hardware realization. In particular, traditional resampling algorithms pose a significant challenge for a pipelined implementation and lead to increased latency of the whole implementation [10–14]. However, resampling method based on Markov Chain Monte Carlo (MCMC)[15] helps to avoid the expensive resampling step that traditional resampling based implementations would need. The Independent Metropolis-Hastings (IMH) resampling method which is adopted in this work, is

a MCMC based method that does not suffer from bottlenecks in pipelining and is suitable for parallelization [16].

1.2 State of the Art on Real Time PFs

The implementation of the PFs for real time applications has so far been a major obstacle due to the PFs high computational requirements. In this sense, there exists several studies aimed at accelerating the intensive computations in PFs and make them amenable to real-time problems. These studies, in general, can be classified under three major topics (Fig. 1.1):

1. Algorithmic modifications
2. Adaptive particle filtering
3. Hardware implementations
    - FPGA based implementations
    - Non-FPGA based implementations

The discussion on literature studies related to the above three major topics are explained in the following subsections.

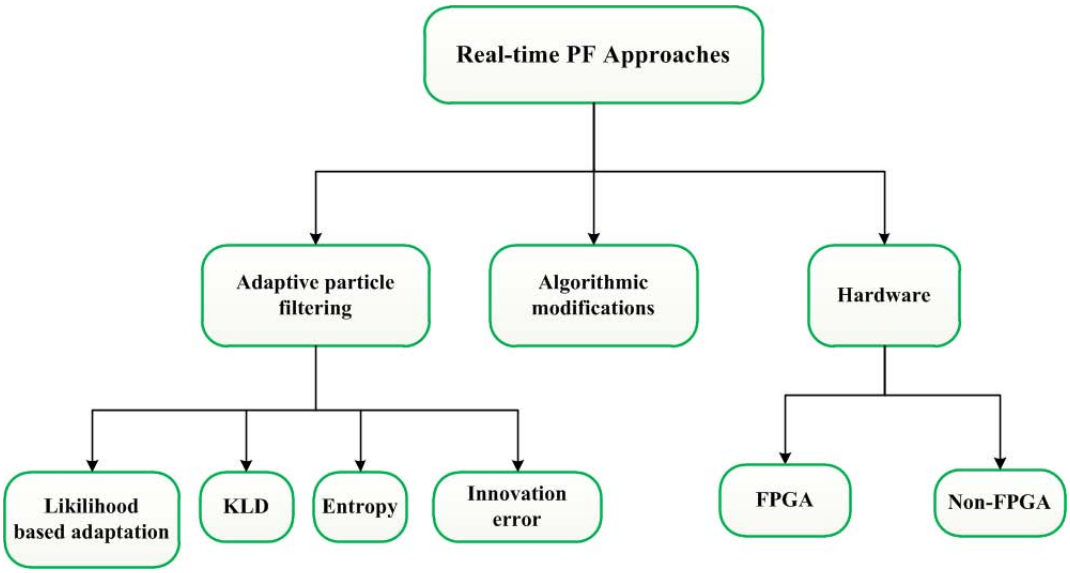


Figure 1.1: Real time particle filter approaches

1.2.1 Adaptive Particle Filtering

The number of particles is the key parameter of PFs, were its square root is inversely proportional to the rate of convergence of the approximate probability distribution towards the true posterior [17]. This implies that the filter perfectly approximates the posterior distribution when the number of particles tends to infinity. However, since the computational cost grows with the number of particles, for practical use a specific number of particles has to be chosen in the design. In many practical applications, the observations arrive sequentially, and there is a strict deadline for processing each new observation. Then, the best solution in terms of filter performance is to increase the number of particles as much as possible and keep it fixed. Also, in some hardware implementations, the number of particles is a design parameter that cannot be modified during implementation. Nevertheless, in many other applications where resources are scarce or are shared with a dynamical allocation and/or with energy restrictions, one might be interested in adapting the number of particles in a smart way.

However, the selection of the number of particles, is often a delicate subject because, the performance of the filter cannot usually be described in advance as a function of the number of particles, and the mismatch between the approximation provided by the filter and the unknown posterior distribution is obviously also unknown. Therefore, although there is a clear trade-off between performance and computational cost, this relation is not straightforward. Increasing the number of particles over a certain value may not significantly improve the quality of the approximation while decreasing the number of particles below some other value can dramatically affect the performance of the filter. For example, in a robot localization problem choosing not enough samples results in a poor approximation of the underlying posterior and the robot frequently fails to localize itself. On the other hand, if we choose too many samples, each update of the algorithm takes several seconds and valuable sensor data could be lost or not processed.

In the adaptive approaches different kind of metrics such as Kullback-Leibler distance (KLD), innovation error, likelihood and entropy are used to track the

estimation accuracy which helps to tune the number of particles at run time. The existing techniques based on these metrics for adapting the number of particles are:

KLD:- The work in [18, 19] presents a KLD-sampling method that helps to choose a small number of particles while the density is focused on a small part of the sate space, and a large number of particles if the state uncertainty is large. In this study the authors particularly, applied the procedure to a mobile robot localization problem. The application of this approach to a real time robot localization problem is presented in [20]. In this study the issue of a significantly higher sensor data rate than the PF update rate is considered to avoid the lose of useful sensor data. In their approach, for an efficient use of sensor information, particle sets are divided between all available observations and the state is represented as a mixture of particle sets. And this leads to an increase in the performance of the PF compared to the original PF approach. An improvement to the KLD sampling is conducted in [21], by adjusting the variance and gradient data to generate particles near the high likelihood region. A similar recent study in [22] uses the adaptive PF approach to a mobile robot localization application for reducing the run-time computational complexity in PF. With a similar reasoning, the study in [23] suggests to adaptively vary the number of particles based on the estimation quality for reducing the complexity in PF.

Likelihood:- In the likelihood adaptation approach the number of samples is determined while the sum of non-normalized likelihoods (importance weights) exceeds a prespecified threshold. Such an approach has been applied to a mobile robot localization in [24]. In a robot localization context, the main idea in this approach is that the required particle set is small while the particle set is well in tune with the sensor measurement and the individual importance weights of the particles are large. However, the particle set is large while it is not well in tuned with the sensor reading and the individual weight of the particles is small. From the study presented in [25] the likelihood adaptation approach has showed lower estimation performance compared to the KLD approach.

Innovation:- The approach presented in [26] uses the innovation error metric to modify the number of particles where the authors particularly applied it to a target tracking problem. This approach has recently been applied to a visual tracking in [27]. The adaptation of the particles in this approach is controlled by the relative distance among the particles. While the distance between two neighboring particles is below a predefined threshold, the particle with the largest weight is kept and the one with the smaller weight is rejected.

1.2.2 Algorithmic Modifications

Besides the adaptive particle size approach for handling the real time computational challenges with PFs, there have been also studies aimed at modifying the basic PF algorithm to achieve the same objective. For example, the work in [28] incorporates MCMC steps to improve the quality of the sample based posterior approximation. Another approach, Auxiliary PF, applies a one step lookahead to minimize the mismatch between the proposal and the target distributions thereby minimizing the variability of the importance weights, which in turn determines the efficiency of the importance sampler [29]. The application of this approach to the simultaneous localization and mapping (SLAM) problem is presented in [30], were a hybrid auxiliary PF and differential evolution (DE) method is used. The auxiliary PF and the DE are applied for the estimation of the pose (position and orientation) and the robots environment respectively. For systems containing a linear substructure subjected to Gaussian noise, with a combination of the standard PF and the KF (Marginalized PF) it is possible to obtain better estimates with reduced complexity compared to using the standard PF only approach [31]. This approach is applied in the two common SLAM variants, Fast-SLAM 1.0 [32] and Fast-SLAM 2.0 [33]. In general, the disadvantages with the adaptive PF and modification of the basic PF approaches is that they introduce extra computational steps to the basic particle filtering algorithm that increases the complexity of the PF algorithm.

1.2.3 FPGA Based Implementations

PFs have been applied in the field of mobile robotics to perform tracking, localization, mapping and navigation tasks to deal with the uncertainties and / or noise generated by the sensors as well as with the intrinsic uncertainties of the environment. Most real-time robotics applications have constraints on performance, area, cost, flexibility and power consumption. Therefore, the development of hardware architectures focused on speeding up the PF execution, using an appropriate hardware platform like the FPGA is of importance in the real-time applicability of the PFs. With this context, PFs hardware implementations have been previously proposed to tackle the intensive computation.

A parallel hardware architecture for the PF applied to target tacking in wireless network is suggested for an implementation on an FPGA or on a fixed-point digital signal processor in [34]. From this study a massive increase in the processing speed is shown by dividing the PF process into twenty parallel PFs. The authors have also presented an algorithm that facilitates the mixing among the parallel processing units and reduces memory bandwidth. The work presented in[35] discusses on different hardware architectures for PF algorithm for an FPGA implementation to target tracking application. Their proposed architecture allowed to perform all the three steps of the PF concurrently. Regarding to the implementation of the importance weight step this study used piecewise linear function instead of the exponential function to reduce the complexity of the hardware architecture showing small degradation in performance.

The real time execution of the PFs becomes further difficult in certain applications due to the complexity of models or the large number of particles requirements. This happens when the rate of incoming measurement data is higher than the update rate of the PFs. In [20] this issue has been addressed by distributing the particles among different observations within an estimation interval. A similar study to Monte Carlo Localization algorithm for a real time mobile robot application is presented in [36]. This work focuses on the sampling step of the PF by emphasizing the importance of providing a good quality pseudo random number generator. The

study presented an embedded implementation based on the FPGA customized to compute the complete Monte Carlo Localization algorithm for a real time mobile robot application. In addition, the authors in [37] considered a mixture PF algorithm for a real time realization of integrated 3D navigation system for a land vehicle, due to its requirement of less number of particle compared to the standard sampling importance resampling (SIR) PF. They considered further optimization for their embedded system implementation to run the algorithm at higher rate. For the optimization they considered a method called fast median-cut clustering to reduce the computation time needed in the importance weighting step.

For bridging the gap between PF theory and its practical use it is a well known fact to reduce the execution time of the algorithms by implementing them using multiple parallel processing elements. However, the main challenge to such an implementation is the parallelization of the resampling step with the other steps of the PF. As a result of this, most of the works on parallel PFs focus on designing a resampling step suitable for parallel implementation. There are some studies that address this in hardware. The work in [38] presents hardware architectures and memory schemes to the resampling and sampling steps of the SIR PF algorithm. The presented architectures are based on systematic resampling and residual systematic resampling algorithms. The implementation of the proposed architectures on the FPGA resulted in 32 times faster speed than the same problem on a DSP. In [39] a compact threshold based resampling algorithm and architecture for efficient hardware implementation is presented. The use of such simple threshold-based scheme in the resampling algorithm resulted in a reduction in the complexity of hardware implementation. Their algorithm showed approximately equal performance with the traditional systematic resampling algorithm. A similar work is also presented in [40]. In[41] an improvement over the basic residual resampling algorithm is done and its hardware implementation is proposed. The proposed algorithm helps to avoid the resampling of the residuals as a result of which it has only one loop compared to the basic residual resampling algorithm. The modified algorithm resulted in approximately equal performance with the traditional systematic resampling and

residual systematic algorithms. Distributed resampling algorithms with proportional allocation (RPA) and non-proportional allocation (RNA) of particles are considered for real time FPGA implementation in [42]. The main advantage considered with such resampling schemes is that the communication between the parallel particle processing units is reduced and made deterministic.

1.2.4 Non-GPGA Based Implementations

The hardware implementations of the PFs, besides FPGA, on many-core architectures (GPGPUs and multi-core CPUs) has been also one of the approaches taken for making the PFs more feasible for real-time applications. This section is dedicated to the literature review of such non-FPGA based hardware implementation of the PFs.

A comprehensive study on the design and implementation of high-performance PFs on many-core architectures is presented in [43]. The study considered a complex robotic arm application with nine state variables for pushing the estimation rates and accuracy to new levels. For such application utilizing over one million particles, they claimed on achieving estimation rates of 100-200 Hz on GPGPUs. A similar work in [44] presented a parallel implementation of the PF and KF on a multi-core platform. The study presented a comparison of the speed up and performances for four existing parallel PFs (globally distributed PF, resampling with proportional allocation PF, resampling with non-proportional allocation PF and the Gaussian PF) implementation on a shared memory multicore computer using up to eight cores.

The work in [45] developed parallel algorithm for PF specifically designed for GPUs and similar parallel computing devices. In this study all the PF steps are executed in a massively parallel manner. Their approach helps to consider the limited bandwidth for data transfer between the GPUs and the CPUs memory by performing all the numerical computations or memory access solely inside the GPUs. In [46] a multicore processor based parallel implementation and performance analysis of the PF for a vehicle localization and map-matching algorithm is presented. In this study a speedup of up to 75 times is reported on parallel GPU implementation compared to the sequential implementation.

To solve the serial nature of the basic systematic resampling, which resulted in a major bottleneck to the full parallelization of the PF, the study [13] proposed a shared-memory systematic resampling algorithm for an implementation on GPU. This work aims to tackle the problems of the distributed resampling approach considered by several authors. These problems include: the responsibility of the central processing unit for generating and distributing random numbers, the use of single processor for weight summation and normalization, the centralized resampling of particles and the requirement of huge communication between central processor and other processors. So in their approach random numbers are generated in parallel, the sum of weights are obtained through a parallel reduction algorithm and the resampling step is parallelized. The study presented in [47] showed the implementation for different resampling methods including the Metroplois and rejection sampling on a GPU. It is suggested that Metropolis and rejection resampling methods resulted in a significant increase in speed compared to the most common resampling methods like multinomial, stratified and systematic resampling.

In summary, several works present hardware realizations of the PF algorithm in many applications based on different hardware platforms. However, there are few studies of hardware realization applying PFs to more complex real-time problems like SLAM in order to accelerate the underlying intensive computations and achieve real-time performance. Among the very few attempts to SLAM implementation in hardware platforms is the work presented in [48]. However, this work presents an FPGA implementation of a variant SLAM based on a genetic algorithm called SMG-SLAM (Scan-Matching Genetic SLAM). Other studies are based on KF algorithm [49–51]. This fact suggest the necessity of designing hardware architectures for PF applied to SLAM.

1.3 Objectives

Even though several methods for accelerating the intensive computations of PFs have been proposed in the literature, still the practical use of PFs has been limited for a vast range of hard real-time applications such as robotics, target tracking,

positioning, and navigation as they cannot meet the requirements of real-time processing. Furthermore, in the literature no significant effort has been considered for the development of efficient HW/SW architectures for PFs specially tailored to complex practical real-time applications, where real time processing and power consumption optimizations are crucial. On the other hand, PFs are potentially useful tools in real time navigation and robotics applications to provide precise and accurate estimations [37, 52, 53]. Taking the advantages of the FPGAs flexibility in the realization of PFs implementation, improved performance for PFs in complex real-time problem can be achieved. In this sense, the evaluation on the possibility of the PFs for a complex real-time applications, in particular to the SLAM problem, is considered in this thesis. SLAM is one of the core applications in most robotic systems, where a mobile robot is required to localize its position while at the same time building the map of its environment.

According to the review presented in the aforementioned paragraphs, the overall purpose of this doctoral thesis is the speedup of the intensive computations in PFs with the aim to make them amenable to real time applications. In order to achieve this overall objective, the following specific objectives are set.

Thesis Objectives

- To determine the critical computational challenges involved in the different operational steps of the PF algorithm for hardware implementation.
- To design and develop efficient HW / SW architectures for high speed PF, with the goal to bring the PF closer to complex practical real-time applications. It aims to reduce the computational complexity of the PF algorithm and achieve fast computations using less hardware resources so as to be implemented in real-time robotic system. To achieve this objective, hardware architectures for PF are proposed and implemented on an FPGA platform.
- To apply the developed HW / SW architectures to real time SLAM problem.

1.4 Publications

This doctoral thesis is mainly based on the following previous publications and submitted materials.

- **Sileshi, B.G**, Oliver, J., Toledo, R., Gonçalves, J., Costa,P., *On the Behaviour of Low Cost Laser Scanners in HW/SW Particle Filter SLAM Applications*, Robotics and Autonomous Systems, Volume 80, June 2016, Pages 11-23, ISSN 0921-8890, (http://www.sciencedirect.com/science/article/pii/S0921889015303201)

- **Sileshi, B.G.**; Ferrer, C.; Oliver, J., *Accelerating Techniques for Particle Filter Implementations on FPGA*, In Emerging Trends in Computer Science and Applied Computing, edited by Quoc Nam Tran and Hamid Arabnia, Morgan Kaufmann, Boston, 2015, Pages 19-37, Emerging Trends in Computational Biology, Bioinformatics, and Systems Biology, ISBN 9780128025086, (http://www.sciencedirect.com/science/article/pii/B9780128025086000028)

- **Sileshi, B.G.**; Ferrer,C.; Oliver, J., *Accelerating Particle Filter on FPGA*, 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, 2016, Accepted.

- **Sileshi, B.G.**; Ferrer, C.; Oliver, J., *Computational Speedup Hardware Architectures for Particle Filter*, to be submitted to a journal.

- **Sileshi, B.G.**; Ferrer, C.; Oliver, J., *Particle Filter Parallelization Using Metropolis Coupled Markov Chain Monte Carlo*, Submitted to Journal of Circuits, Systems & Signal Processing.

- **Sileshi, B.G.**; Oliver.J,Toledo.R, Gonçalves, J., Costa, P., *Particle Filter SLAM on FPGA: A Case Study on Robot@ Factory Competition*, In Robot 2015: Second Iberian Robotics Conference, pp. 411-423, 2016.

- **Sileshi, B.G.**; Ferrer, C.; Oliver, J., *Hardware/Software Co-design of Particle Filter in Grid Based Fast-SLAM Algorithm*, International Conference on Embedded Systems and Applications (ESA), 2014 Conference on , 24-27 Jul. 2014.

- **Sileshi, B.G.**; Ferrer, C.; Oliver, J., *Accelerating Hardware Gaussian Random Number Generation Using Ziggurat and CORDIC Algorithms*, In SENSORS, 2014 IEEE. IEEE, 2014. p. 2122-2125.

- **Sileshi, B.G.**; Ferrer,C.; Oliver, J., *Particle Filters and Resampling Techniques: Importance in Computational Complexity Analysis*, Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on , vol., no., pp.319,325, 8-10 Oct. 2013

1.5 Thesis Outline

This thesis is composed of six chapters including this introduction which are organized as follow:

- Chapter 2- **Particle Filters (PFs) and SLAM Background**: is dedicated to an in depth explanation on the theoretical foundations of the PFs algorithm evaluated during this thesis. Detailed explanations on the computations involved in each step of the PF is presented. In addition, a discussion on some of the most commonly used variants of the basic PF algorithm and the different techniques for performing the resampling operation is provided. Finally, the explanation on the application of the PF to the SLAM problem is presented, where the theory of the SLAM problem is described based on the PF algorithm.

- Chapter 3- **PFs Complexity Analysis and HW Acceleration Methods**: presents the analysis on the identification of the computational bottlenecks of the PFs. Based on the computational complexity analysis, this chapter provides the explanation on the proposed PFs acceleration techniques, namely

the CORDIC and Ziggurat algorithms. The discussion on the CORDIC algorithm for a hardware evaluation of different complex functions involved in PF computations is given. Furthermore, a discussion on uniform and Gaussian hardware random number generator based on Ziggurat and Thausworth algorithms respectively is provided.

- Chapter 4- **HW / SW Approach for PF-SLAM FPGA Architecture and Implementation**: explains the FPGA hardware design of the PF based on a HW/SW co-design approach. Based on the PF acceleration techniques provided in Chapter 3, a discussion on the proposed hardware architecture of a PF hardware acceleration module is presented. The PF hardware acceleration module is interfaced with the soft-core MicroBlaze processor of the FPGA resulting in the PF HW/SW architecture. The proposed PF HW/SW co-design architecture is evaluated for the SLAM application based on different real data sets. Finally, with an effort to parallelize the PF computations this chapter presents a parallel PF architecture with the application a Metroplis coupled Markov Chain Monte Carlo approach.

- Chapter 5- **HW Approach for PF-SLAM Processing Element Design**: provides the fully hardware architecture design for the different steps of the PF applied to the SLAM problem. The details in the design and evaluation of hardware architectures of the different computational modules is given. In addition, the design and implementation of a parallel laser scanner co-processor is explained with the objective of accelerating the processing of large measurement data from a laser scanner sensor in the update step of the PF applied to the SLAM problem.

- Chapter 6- **Conclusions**: gives the general conclusions and future outlooks regarding to the efforts made in this thesis to achieve real time PF computations.

References

[1] R. E. Kalman. "A new approach to linear filtering and prediction problems". In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[2] B. Anderson and J. Moore. "Optimal filtering". In: *Information and system sciences series* (1979).

[3] R. J. Meinhold and N. D. Singpurwalla. "Robustification of Kalman filter models". In: *Journal of the American Statistical Association* 84.406 (1989), pp. 479–486.

[4] G. A. Einicke and L. B. White. "Robust extended Kalman filtering". In: *IEEE Transactions on Signal Processing* 47.9 (1999), pp. 2596–2599.

[5] N. J. Gordon, D. J. Salmond, and A. F. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *Radar and Signal Processing, IEE Proceedings F*. Vol. 140. 2. IET. 1993, pp. 107–113.

[6] N. Gordon, B Ristic, and S Arulampalam. "Beyond the kalman filter: Particle filters for tracking applications". In: *Artech House, London* (2004).

[7] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *Signal Processing, IEEE Transactions on* 50.2 (2002), pp. 174–188.

[8] A. Smith, A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo methods in practice*. Springer Science & Business Media, 2013.

[9] A. Doucet and A. M. Johansen. "A tutorial on particle filtering and smoothing: Fifteen years later". In: *Handbook of Nonlinear Filtering* 12.656-704 (2009), p. 3.

[10] R. Douc and O. Cappé. "Comparison of resampling schemes for particle filtering". In: *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*. IEEE. 2005, pp. 64–69.

[11] J. D. Hol, T. B. Schon, and F. Gustafsson. "On resampling algorithms for particle filters". In: *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*. IEEE. 2006, pp. 79–82.

[12] M. Bolić, P. M. Djurić, and S. Hong. "New resampling algorithms for particle filters". In: *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. II–589.

[13] P. Gong, Y. O. Basciftci, and F. Ozguner. "A parallel resampling algorithm for particle filtering on shared-memory architectures". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE. 2012, pp. 1477–1483.

[14] S. Hong, M. Bolić, and P. M. Djuric. "An efficient fixed-point implementation of residual resampling scheme for high-speed particle filters". In: *Signal Processing Letters, IEEE* 11.5 (2004), pp. 482–485.

[15] C. M. Carlo. "Markov Chain Monte Carlo and Gibbs Sampling". In: *Notes* (2004).

[16] A. C. Sankaranarayanan, A. Srivastava, and R. Chellappa. "Algorithmic and architectural optimizations for computationally efficient particle filtering". In: *Image Processing, IEEE Transactions on* 17.5 (2008), pp. 737–748.

[17] A. Bain and D. Crisan. *Fundamentals of stochastic filtering*. Vol. 3. Springer, 2009.

[18] D. Fox. "Adapting the sample size in particle filters through KLD-sampling". In: *The international Journal of robotics research* 22.12 (2003), pp. 985–1003.

[19] T. Li, S. Sun, and T. P. Sattar. "Adapting sample size in particle filters through KLD-resampling". In: *Electronics Letters* 49.12 (2013), pp. 740–742.

[20] C. Kwok, D. Fox, and M. Meila. "Real-time particle filters". In: *Proceedings of the IEEE* 92.3 (2004), pp. 469–484.

[21] S.-H. Park, Y.-J. Kim, and M.-T. Lim. "Novel adaptive particle filter using adjusted variance and its application". In: *International Journal of Control, Automation and Systems* 8.4 (2010), pp. 801–807.

[22] T. C. Chau, W. Luk, P. Y. Cheung, A. Eele, and J. Maciejowski. "Adaptive sequential monte carlo approach for real-time applications". In: *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE. 2012, pp. 527–530.

[23] M. Bolić, S. Hong, and P. M. Djurić. "Performance and complexity analysis of adaptive particle filtering for tracking applications". In: *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*. Vol. 1. IEEE. 2002, pp. 853–857.

[24] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. "Monte carlo localization: Efficient position estimation for mobile robots". In: *AAAI/IAAI* 1999 (1999), pp. 343–349.

[25] D. Fox. "KLD-sampling: Adaptive particle filters". In: *Advances in neural information processing systems*. 2001, pp. 713–720.

[26] P. Closas and C. Fernández-Prades. "Particle filtering with adaptive number of particles". In: *Aerospace Conference, 2011 IEEE*. IEEE. 2011, pp. 1–7.

[27] D. Forte and A. Srivastava. "Resource-aware architectures for adaptive particle filter based visual target tracking". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18.2 (2013), p. 22.

[28] W. R. Gilks and C. Berzuini. "Following a moving target—Monte Carlo inference for dynamic Bayesian models". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.1 (2001), pp. 127–146.

[29] M. K. Pitt and N. Shephard. "Filtering via simulation: Auxiliary particle filters". In: *Journal of the American statistical association* 94.446 (1999), pp. 590–599.

[30] R. Havangi, M. A. Nekoui, M. Teshnehlab, and H. D. Taghirad. "A SLAM based on auxiliary marginalised particle filter and differential evolution". In: *International Journal of Systems Science* 45.9 (2014), pp. 1913–1926.

[31] R Karlsson, T Schon, and F Gustafsson. "Complexity analysis of the marginalized particle filter". In: *IEEE Transactions on Signal Processing* 53 (2005), pp. 4408–4411.

[32] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai*. 2002, pp. 593–598.

[33] D Roller, M. Montemerlo, S Thrun, and B. Wegbreit. "Fastslam 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 2003.

[34] Y. Zhang, T. Sathyan, M. Hedley, P. H. Leong, and A. Pasha. "Hardware efficient parallel particle filter for tracking in wireless networks". In: *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*. IEEE. 2012, pp. 1734–1739.

[35] H. A. A. El-Halym, I. I. Mahmoud, and S. Habib. "Proposed hardware architectures of particle filter for object tracking". In: *EURASIP Journal on Advances in Signal Processing* 2012.1 (2012), pp. 1–19.

[36] V. Bonato, B. F. Mazzotti, M. M. Fernandes, and E. Marques. "A Mersenne twister hardware implementation for the Monte Carlo localization algorithm". In: *Journal of Signal Processing Systems* 70.1 (2013), pp. 75–85.

[37] M. M. Atia, J. Georgy, M. Korenberg, and A. Noureldin. "Real-time implementation of mixture particle filter for 3D RISS/GPS integrated navigation solution". In: *Electronics letters* 46.15 (2010), pp. 1083–1084.

[38] A. Athalye, M. Bolic, S. Hong, and P. M. Djuric. "Architectures and memory schemes for sampling and resampling in particle filters". In: *Digital Signal Processing Workshop, 2004 and the 3rd IEEE Signal Processing Education Workshop. 2004 IEEE 11th*. IEEE. 2004, pp. 92–96.

[39] S.-H. Hong, Z.-G. Shi, J.-M. Chen, and K.-S. Chen. "A low-power memory-efficient resampling architecture for particle filters". In: *Circuits, Systems and Signal Processing* 29.1 (2010), pp. 155–167.

[40] Z.-G. Shi, Y. Zheng, X. Bian, and Z. Yu. "Threshold-based resampling for high-speed particle PHD filter". In: *Progress In Electromagnetics Research* 136 (2013), pp. 369–383.

[41] S. Hong, J. Jiang, and L. Wang. "Improved residual resampling algorithm and hardware implementation for particle filters". In: *Wireless Communications & Signal Processing (WCSP), 2012 International Conference on*. IEEE. 2012, pp. 1–5.

[42] M. Bolić, P. M. Djurić, and S. Hong. "Resampling algorithms and architectures for distributed particle filters". In: *Signal Processing, IEEE Transactions on* 53.7 (2005), pp. 2442–2450.

[43] M. Chitchian, A. S. van Amesfoort, A. Simonetto, T. Keviczky, and H. J. Sips. "Adapting particle filter algorithms to many-core architectures". In: *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE. 2013, pp. 427–438.

[44] O. Rosén and A. Medvedev. "Efficient parallel implementation of state estimation algorithms on multicore platforms". In: *Control Systems Technology, IEEE Transactions on* 21.1 (2013), pp. 107–120.

[45] K. McAlinn, H. Katsura, and T. Nakatsuma. "Fully Parallel Particle Learning for GPGPUs and Other Parallel Devices". In: *arXiv preprint arXiv:1212.1639* (2012).

[46] O. Tosun et al. "Parallelization of particle filter based localization and map matching algorithms on multicore/manycore architectures". In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 820–826.

[47] L. M. Murray, A. Lee, and P. E. Jacob. "Rethinking resampling in the particle filter on graphics processing units". In: *arXiv preprint arXiv:1301.4019* (2013).

[48] G. Mingas, E. Tsardoulias, and L. Petrou. "An FPGA implementation of the SMG-SLAM algorithm". In: *Microprocessors and Microsystems* 36.3 (2012), pp. 190–204.

[49] V. Bonato, E. Marques, and G. A. Constantinides. "A floating-point extended kalman filter implementation for autonomous mobile robots". In: *Journal of Signal Processing Systems* 56.1 (2009), pp. 41–50.

[50] V Bonato. "Proposal of an FPGA hardware architecture for SLAM using multi-cameras and applied to mobile robotics". PhD thesis. Ph. D. dissertation, Institute of Computer Science and Computational Mathematics-University of Sao Paulo, Sao Carlos, SP, Brazil, 2008.

[51] S. Cruz, D. M. Munoz, M. E. Conde, C. H. Llanos, and G. A. Borges. "A hardware approach for solving the robot localization problem using a sequential EKF". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE. 2013, pp. 306–313.

[52] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund. "Particle filters for positioning, navigation, and tracking". In: *Signal Processing, IEEE Transactions on* 50.2 (2002), pp. 425–437.

[53] J. Georgy, A. Noureldin, and C. Goodall. “Vehicle navigator using a mixture particle filter for inertial sensors/odometer/map data/GPS integration”. In: *Consumer Electronics, IEEE Transactions on* 58.2 (2012), pp. 544–552.

2

Particle Filters (PFs) and SLAM Background

2.1 Introduction

This chapter examines the relevant background in the PFs theory, where detailed explanation on typically used particle filtering methods and their respective computational steps is given. In addition, a discussion on the application of the PFs to one of the fundamental problem in the field of mobile robotics, namely the simultaneous localization and mapping (SLAM) problem, is provided. As PFs complexity and performance depends on a specific application model nonlinearity, model dynamics and dimension of the state, detailed explanation behind the theoretical aspect of the SLAM problem is also given. First the robot localization problem is explained including the different motion and measurement models required to solve the problem. Then an explanation on the probabilistic robotic mapping is given, where the different robot mapping techniques are explained. Finally the explanation on the mathematical foundation to the solution of the SLAM problem based on PF approach is given.

The chapter provides a discussion of theoretical techniques presented in the recent literature in respect to PFs and SLAM. All these techniques present the recent evolution in the field that forms the basis of this thesis research.

2.2 Dynamic State Space Models

PFs are applied to many real world systems for state estimations that are formulated as dynamic state space models. A dynamic state space model assumes two fundamental mathematical models: the state dynamics and the measurement equation. The state dynamics model describes the evolution of the state of interest $x_t \in R^{d_x}$ with time and is given by:

$$x_t = f_t(x_{t-1}, v_{t-1}) \tag{2.1}$$

Here x_t is the state vector to be estimated, t denotes the time step and f_t is a known possibly non-linear function, v_{t-1} is the process noise sequences, d_x is the dimensions of the state. The measurement (observation) equation relates the received measurements to the state vector and is given by:

$$z_t = h_t(x_t, w_t) \tag{2.2}$$

where $z_t \in R^{d_z}$ is the vector of received measurements at time step t, d_z is the dimensions of the measurement vector, h_t is the known measurement function and w_t is the measurement noise sequences. The process v_{t-1} and measurement w_t noise

Table 2.1: Dynamic state space model notations

Variable	Description
t	Time index
x_t	State vector at time step t
d_x	Dimension of the state vector
v	Process noise sequence
f_t	Nonlinear function of the state and process noise sequence
z_t	Measurement vector at time step t
d_z	Dimension of the measurement vector
w	Measurement noise sequence
h_t	Nonlinear function of the state and measurement noise sequence
$p(x_0)$	Prior state probability density
$p(x_t\|x_{t-1})$	Sate transition probability density
$p(z_t\|x_t)$	Measurement probability density

sequences are assumed to be white, with known probability density functions and mutually independent.

Equations 2.1 and 2.2 define a first order Markov process, where their equivalent probabilistic descriptions for the state and measurement evolution are given by $p(x_t|x_{t-1})$ and $p(z_t|x_t)$ respectively. A process is a first order Markov process if the conditional probability distribution of future states, given the present state and all past states, depends only upon the present state and not on any past states (Fig. 2.1). Fig. 2.1 thus implies that the distribution of the state x_{t+1} conditional on the history of the states, $x_0, ..., x_t$, is determined by the value taken by the preceding one, x_t; this is called the *Markov property*. Similarly, the distribution of z_t conditionally on the past observations $z_o, ..., z_{t-1}$ and the past values of the state, $x_0, ..., x_t$, is determined by the x_t only. Therefore, for a first order Markov process the true states are conditionally independent of all earlier states given the immediately previous state (Equation 2.3). Similarly, the measurement at the tth time step is dependent only upon the current state and is conditionally independent of all other states given the current state (Equation 2.4).

$$p(x_t|x_0, ..., x_{t-1}) = p(x_t|x_{t-1}) \tag{2.3}$$

$$p(z_t|x_0, ..., x_t) = p(z_t|x_t) \tag{2.4}$$

The final piece of information to complete the specification of the estimation problem is the initial conditions. This is the prior probability density function (PDF) $p(x_0)$ of the state vector at time $t = 0$, before any measurements have been

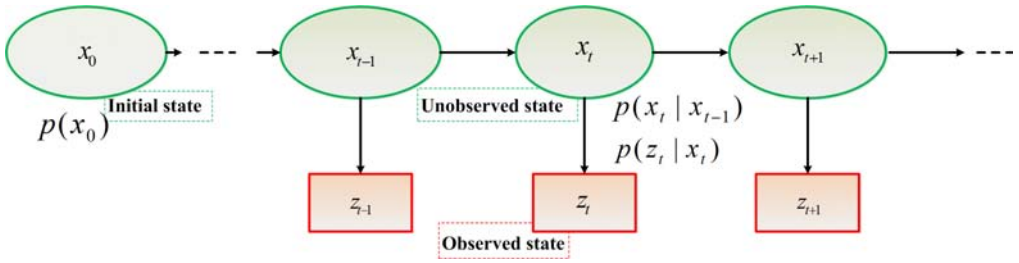


Figure 2.1: Graphical representation of the dependence structure of a first order hidden Markov Model, where x_t and z_t are the hidden and observed states respectively.

received. The state-space model can equivalently be formulated as a hidden Markov model (HMM), as summarized in Equation 2.5 where the stochastic states x_t evolve according to a Markov chain with transition probabilities $p(x_t|x_{t-1})$. In other words, the chain is defined by the initial PDF, $p(x_0)$, combined with the likelihood of the states at some time given the states at the previous time, $p(x_t|x_{t-1})$. The observations z_t are distributed according to $p(z_t|x_t)$.

$$\begin{aligned} &p(x_0) \\ &p(x_t|x_{t-1}) \\ &p(z_t|x_t) \end{aligned} \tag{2.5}$$

The solution to the estimation of the states x_t at time t based on history of measurements up to time t ($z_{0:t}$), is given by the filtering density $p(x_t|z_{0:t})$. PFs provide approximations to the filtering density for dynamic systems described by Equations 2.1, 2.2 and 2.5. As these equations describe a general, nonlinear and non-Gaussian systems, the assumptions of linear systems with Gaussian noise, required by the classical Kalman filter can be ignored. Such a capability to handle nonlinear, non-Gaussian systems allows PFs to achieve improved accuracy over Kalman filter-based estimation methods.

2.3 Recursive Bayesian Filters

Bayesian filtering allows the online estimation of the unknown state x_t of a dynamic system based on the sequence of all available measurement up to time t which is given by the posterior distribution $p(x_t|z_{1:t})$. In recursive Bayesian estimation, the true states are assumed to be a hidden Markov process, and the measurements are the observed states of a HMM. The posterior distribution, $p(x_t|z_{1:t})$, is obtained by a recursive estimation that consists of a *prediction* and an *update* steps. In the prediction step the state estimate from the previous time step is used to predict the current state. This estimate is known as the *prior estimate*, as it does not incorporates any measurement from the current time step. In the update step, the state estimate from the previous step is updated according to the actual

measurements done on the system. This estimate is referred to as the *a posteriori estimate*. The measurements of the state follows the probability distribution which is often called the *likelihood probability* given by the distribution $p(x_t|z_t)$.

The recursive estimation of the posterior distribution with the *prediction* and *update* steps is given as follows

Prediction: This step involves using the system model $p(x_t|x_{t-1})$ and the available PDF $p(x_{t-1}|z_{1:t-1})$ at time $t-1$, to obtain the prediction density of the state at time t via the Chapman Kolmogorov equation [6]:

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1} \tag{2.6}$$

Where the probabilistic model of the state evolution (transitional density), $p(x_t|x_{t-1})$, is defined by the system equation 2.1.

Update: The update step is carried out while the measurement z_t becomes available at time step t. This involves an update of the prediction (or prior) PDF via the Bayes rule:

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{2.7}$$

where the normalizing constant $p(z_t|z_{1:t-1})$ is :

$$p(z_t|z_{1:t-1}) = \int p(z_t|x_t)p(x_t|z_{1:t-1})dx_t \tag{2.8}$$

Fig. 2.2 illustrates the above two steps of the recursive Bayesian filtering approach to the posterior estimation.

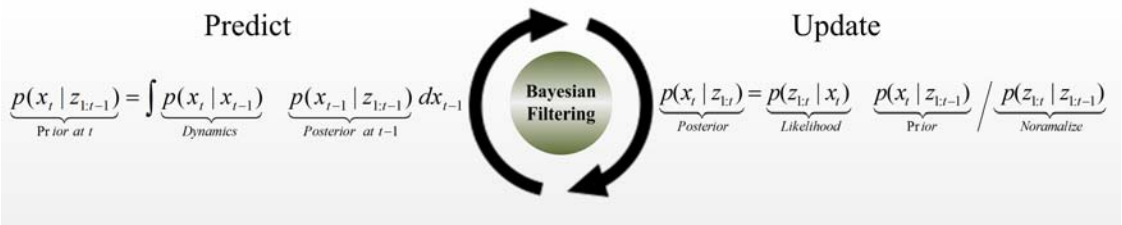


Figure 2.2: Illustration of a Bayesian filtering scheme

2.4 Principles of Importance Sampling

If the solutions to the prediction and update, equations 2.6 and 2.7 respectively, of the optimal Bayesian filtering are not analytically tractable, approximate solutions based on Monte Carlo sampling is normally used. Monte Carlo integration is the basis of Monte Carlo sampling method. It is used in the approximation of integrals of the form $\int f(x)\mu(x)dx$, where f is real-valued measurable function and $\mu(x)$ denotes a probability measure of interest referred to as the *target distribution*. The Monte Carlo sampling approach for the approximation of the integral $\int f(x)\mu(x)dx$ consists of drawing $N \gg 1$ identically independent (i.i.d.) samples $\{x^i\}_{i=1}^N$ from the probability density $\mu(x)$ and then evaluating the sample mean $\frac{1}{N}\sum_{i=1}^N f(x^i)$. Of course, this technique is applicable only when it is possible and reasonably simple to sample from the target distribution. Unfortunately, it is usually not possible to sample effectively from the target distribution, being multivariate, nonstandard, and only known up to a proportionality constant. A possible solution is to apply the importance sampling technique, which is a general Monte Carlo integration method.

In importance sampling, for the estimate of the integral $\int f(x)\mu(x)dx$, samples are generated from a density $q(x)$ which is similar to $\mu(x)$. The PDF $q(x)$ is referred to as the *importance* or *proposal* density. A Monte Carlo estimate of the integral is computed by generating $N \gg 1$ independent samples distributed according to $q(x)$ and forming the weighted sum given by:

$$\frac{1}{N}\sum_{i=1}^{N} f(x^i)\tilde{w}^i \tag{2.9}$$

where

$$\tilde{w}^i = \frac{\mu(x^i)}{q(x^i)} \tag{2.10}$$

are the un-normalized importance weights. As equation 2.9 is an estimator of the integral $\int f(x)\mu(x)dx$ with the sample mean of independent random variables, the law of large number implies that the estimate given by equation 2.9 converges to the integral $\int f(x)\mu(x)dx$ almost surely as N tends to infinity. In many situations the

target density $\mu(x)$ or the proposal density $q(x)$ are known only up to a normalizing factor, particularly when applying importance sampling ideas to HMMs and more generally in Bayesian statistics [1, 2]. If the normalizing factor of the target density $\mu(x)$ is unknown, the normalization of the importance weights is required and the estimate of the integral is given by:

$$\frac{1}{N}\sum_{i=1}^{N} f(x^i)w^i \tag{2.11}$$

where the normalized importance weights are given by:

$$w^i = \frac{\tilde{w}^i}{\sum_{i=1}^{N} \tilde{w}^i} \tag{2.12}$$

Fig. 2.3 illustrates the importance sampling principle, where a weighted set of samples that are drawn from a proposal distribution (solid line) is used for approximating a target distribution (dashed line). In Fig. 2.3, the samples drawn from the proposal distribution are shown with vertical lines and the weighted samples are shown with the lengths of the lines being proportional to their importance weights.

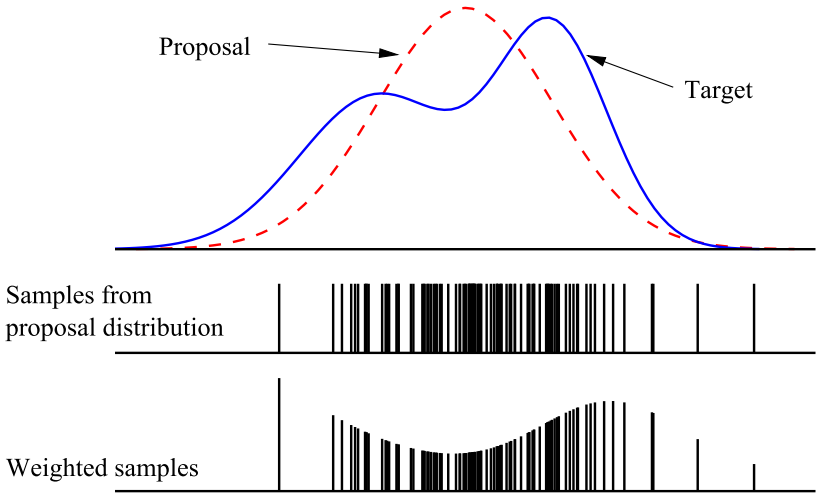


Figure 2.3: Principle of Importance Sampling [3]. The target distribution is approximated by samples that are drawn from the proposal distribution. After weighting the resulting sample set is an approximation of the target distribution.

2.5 Sequential Importance Sampling (SIS)

Although importance sampling is primarily intended to overcome difficulties with direct sampling from $\mu(x)$ when approximating integrals of the form $\int f(x)\mu(x)dx$, it can also be used for sampling from the distribution $\mu(x)$. The latter can be achieved by the Sequential Importance Sampling (SIS) method [4, 5], which is the most basic Monte Carlo method for implementing a recursive Bayesian filter based on the importance sampling principle [6–8]. It is known variously as particle filtering, interacting particle approximation, bootstrap, survival of the fittest and condensation [9–13].

SIS PF applies a Monte Carlo integral approximation method for the representation of the posterior PDF with a random measure composed of discrete set of samples (particles) $\{x_t^i\}_{i=1}^N$ drawn from an easy to sample proposal distribution $q(x_{0:t}|z_{1:t})$ with their corresponding weights $\{w_t^i\}_{i=1}^N$ suitably normalized so that $\sum_{i=1}^N w_t^i = 1$. A discrete weighted approximation to the true posterior density, $p(x_{0:t}|z_{1:t})$ is given by:

$$p(x_{0:t}|z_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta(x_{0:t} - x_{0:t}^i) \tag{2.13}$$

Where $\delta(.)$ is a Dirac Delta function and w_t^i is the importance weight. For samples $\{x_t^i\}_{i=1}^N$ drawn from an importance density $q(x_{0:t}|z_{1:t})$, as per equation 2.10 the normalized importance weights are given by:

$$w_t^i \propto \frac{p(x_{0:t}^i|z_{1:t})}{q(x_{0:t}^i|z_{1:t})} \tag{2.14}$$

SIS algorithm consists of a recursive procedure for the propagation of the samples and their associated weights as successive observations become available. For recursive estimation, an approximation to the posterior $p(x_{0:t}|z_{1:t})$ at time t is obtained based on samples that approximate the posterior $p(x_{0:t-1}|z_{1:t-1})$ from the previous time $t-1$. If the importance density is chosen to factorize as given by Equation 2.15, then one can obtain samples $x_{0:t}^i \sim q(x_{0:t}|z_{1:t})$ by augmenting each of the existing samples $x_{0:t-1}^i \sim q(x_{0:t-1}|z_{1:t-1})$ with new state $x_t^i \sim q(x_t|x_{0:t-1}, z_{1:t})$.

$$q(x_{0:t}|z_{1:t}) \triangleq q(x_t|x_{0:t-1}, z_{1:t})q(x_{0:t-1}|z_{1:t-1}) \tag{2.15}$$

For the derivation of the weight update equation, $p(x_{0:t}|z_{1:t})$ is first expressed in terms of $p(x_{0:t-1}|z_{1:t-1})$, $p(z_t|x_t)$, and $p(x_t|x_{t-1})$.

$$\begin{aligned} p(x_{0:t}|z_{1:t}) &= \frac{p(z_t|x_{0:t}, z_{1:t-1})p(x_{0:t}|z_{1:t-1})}{p(z_t|z_{1:t-1})} \\ &= \frac{p(z_t|x_{0:t}, z_{1:t-1})p(x_t|x_{0:t-1}, z_{1:t-1})}{p(z_t|z_{1:t-1})} \times p(x_{0:t-1}|z_{1:t-1}) \\ &= \frac{p(z_t|x_t)p(x_t|x_{t-1})}{p(z_t|z_{1:t-1})} \times p(x_{0:t-1}|z_{1:t-1}) \\ &\propto p(z_t|x_t)p(x_t|x_{t-1})p(x_{0:t-1}|z_{1:t-1}) \end{aligned} \tag{2.16}$$

By substituting equations 2.15 and 2.16 into equation 2.14, the weight update equation can then be shown to be:

$$\begin{aligned} w_t^i &\propto \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)p(x_{0:t-1}^i|z_{1:t-1})}{q(x_t^i|x_{0:t-1}^i, z_{1:t})q(x_{0:t-1}^i|z_{1:t-1})} \\ &= w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{0:t-1}^i, z_{1:t})} \end{aligned} \tag{2.17}$$

If $q(x_t|x_{0:t-1}, z_{1:t}) = q(x_t|x_{t-1}, z_t)$ then the importance density becomes only dependent on x_{t-1} and z_t. This is particularly useful in the common case when only a filtered estimate of $p(x_t|z_{1:t})$ is required at each time step. With such an assumption , only x_t^i need to be stored; therefore, one can discard the path $x_{0:t-1}^i$ and history of observations $z_{1:t-1}$. The modified weight is then given by:

$$w_t^i \propto w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, z_t)} \tag{2.18}$$

The posterior filter density $p(x_t|z_{1:t})$ is approximated as:

$$p(x_t|z_{1:t}) \approx \sum_{i=1}^{N} w_t^i \delta(x_t - x_t^i) \tag{2.19}$$

where the weights are defined by equation 2.18. It can be shown that $N \to \infty$ the approximation given by equation 2.19 approached the true posterior density $p(x_t|z_{1:t})$. In summary, SIS filtering consists of recursive propagation of the particle

set $S_t = \{x_t^i, w_t^i\}_{i=1}^N$ at time t, from the set $S_{t-1} = \{x_{t-1}^i, w_{t-1}^i\}_{i=1}^N$ at the previous time $t-1$. The initial particle set S_0 is obtained from samples drawn from the prior density $p(x_0)$. The pseudo code description of the SIS method is given in Algorithm 1.

The major problem of the SIS algorithm is that after a certain number of iterations, the *weight degeneracy* problem starts to reveal [14–16]. In other words, a small number of normalized importance weights tend to one while the remaining weights are negligible. This is not a desirable phenomenon, as only few particles contribute in the approximation of the posterior density and lots of computation is wasted to the particles with negligible weight. A common measure of the degeneracy of the algorithm is the effective sample size N_{eff} given by [15, 17]:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N}(w_t^i)^2} \tag{2.20}$$

The degeneracy problem in SIS method can be reduced by use of very large value of samples N but, since it is an impractical approach other alternative techniques are commonly used. Which includes; use of *resampling* and an optimal importance density. In resampling, particles are resampled N times with replacement from the discrete approximation of the posterior density. In this process particles with low importance weight are eliminated and particles with high importance weight are multiplied. This step is performed while N_{eff} falls below certain predefined threshold N_T. The different methods for performing the resampling operation are explained

Algorithm 1 SIS PF

Input: $\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^N, z_t$

1: **for** $i = 1 : N$ **do**
2: Draw $x_t \sim q(x_t|x_{t-1}, z_t)$
3: Calculate the non normalized importance weight $\tilde{w}_t^i = w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, z_t)}$
4: **end for**
5: Calculate sum of weights $W = \sum_{i=1}^N \tilde{w}_t^i$
6: **for** $i = 1 : N$ **do**
7: Normalize: $w_t^i = W^{-1}\tilde{w}_t^i$
8: **end for**
9: **return** $\{x_t^i, w_t^i\}_{i=1}^N$

in section 2.8. The inclusion of a resampling step in the SIS algorithm results in the formulation of the *generic* PF, where its pseudo code is given in Algorithm 2.

In the design of a PF the choice of the importance density $q(x_t|x^i_{t-1}, z_t)$ is the most critical issue. The optimal importance density function that minimizes the variance of importance weights, conditioned upon x^i_{t-1} and z_t is shown to be [5]:

$$q(x_t|x^i_{t-1}, z_t)_{opt} = \frac{p(z_t|x_t, x^i_{t-1})p(x_t|x^i_{t-1})}{p(z_t|x^i_{t-1})} \tag{2.21}$$

Substituting equation 2.21 in 2.18 results in equation 2.22, which states that importance weights at time step t can be computed *before* the particle are propagated to time t. In order to use the optimal importance function, it is necessary to be able to sample from $p(x_t|x^i_{t-1}, z_t)$ and evaluate the integral given by equation 2.23.

$$w^i_t \propto w^i_{t-1} p(z_t|x^i_{t-1}) \tag{2.22}$$

$$p(z_t|x^i_{t-1}) = \int p(z_t|x_t)p(x_t|x^i_{t-1})dx_t \tag{2.23}$$

In general, as sampling from the distribution $p(x_t|x^i_{t-1}, z_t)$ and the evaluation of the integral in equation 2.23 may not be straightforward, commonly a suboptimal choice of the importance density by using the prior density is considered as shown in equation 2.24.

$$q(x_t|x^i_{t-1}, z_t) = p(x_t|x^i_{t-1}) \tag{2.24}$$

Substituting 2.24 in 2.18 results in:

Algorithm 2 Generic PF

Input: $\{x^i_{t-1}, w^i_{t-1}\}^N_{i=1}, z_t$
1: SIS filtering (Algorithm 1)
2: Calculate N_{eff} using equation 2.20
3: **if** $N_{eff} \leq N_T$ **then**
4: RESAMPLE
5: Input:$\{x^i_{t-1}, w^i_{t-1}\}^N_{i=1}$ Output:$\{x^i_{t-1}, w^i_{t-1}, -\}^N_{i=1}$
6: **end if**
7: **return** $\{x^i_t, w^i_t\}^N_{i=1}$

$$w_t^i \propto w_{t-1}^i p(z_t|x_t^i) \tag{2.25}$$

2.6 Sampling Importance Resampling (SIR)

The SIR PF shown in Algorithm 3 is one variant of the SIS algorithm and it is derived by:

1. Using the prior density $p(x_t^i|x_{t-1}^i)$ as the importance density $q(x_t^i|x_{t-1}^i, z_t)$
2. Performing the resampling at every time step.

For this particular choice of the importance density and performing the resampling every time step, the weights are given by:

$$w_t^i \propto p(z_t|x_t^i) \tag{2.26}$$

The resampling step in the SIR and generic PFs is intended to reduce the degeneracy problem. However, it leads to the loss of diversity among the particles. The loss of diversity among the particles may lead to the occupancy of the same point in the state space by all N particles, giving poor representation of the posterior density. The sample diversity in the PF can be improved either via regularization

Algorithm 3 SIR PF

Input: $\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^N, z_t$
1: **for** $i = 1 : N$ **do**
2: Draw $x_t^i \sim p(x_t|x_{t-1}^i)$
3: Calculate the non normalized importance weight $\tilde{w}_t^i = p(z_t|x_t^i)$
4: **end for**
5: Calculate sum of weights $W = \sum_{i=1}^N \tilde{w}_t^i$
6: **for** $i = 1 : N$ **do**
7: Normalize: $w_t^i = W^{-1}\tilde{w}_t^i$
8: **end for**
9: RESAMPLE
Input :$\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^N$ Output:$\{x_{t-1}^i, w_{t-1}^i, -\}_{i=1}^N$
10: **return** $\{x_t^i, w_t^i\}_{i=1}^N$

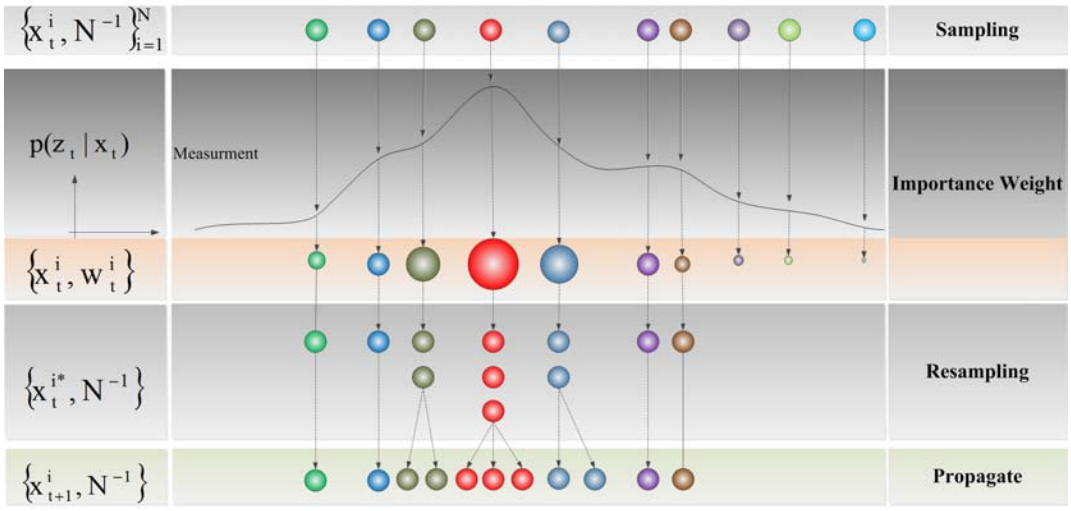


Figure 2.4: Principle of sampling importance resampling (SIR). The particles with weights are represented along the horizontal axis by bullets, the radii of which being proportional to the normalized weight of the particle.

step (Regularized PF) or by the Markov Chain Monte Carlo (MCMC) move step based on the Metropolis-Hasting algorithm [14].

Fig. 2.4 illustrates a graphical representation for a single cycle of the SIR PF operation. The operation starts with a uniformly weighted particle set $\{x_t^i, \frac{1}{N}\}$ which provides an approximation to the prediction density $p(x_t|z_{1:t-1})$. While the measurement z_t becomes available at time t, the importance weight of the individual particles are computed by using the measurement likelihood $p(z_t|x_t)$. This results in a weighted measure $\{x_t^i, w_t^i\}$ for the approximation of the density $p(x_t|z_{1:t})$. In the resampling step, particles are selected more often in proportion to their importance weights to obtain the unweighted measure $\{x_t^{i*}, \frac{1}{N}\}$ which is still an approximation of the density $p(x_t|z_{1:t})$. Finally, the resampled particle set is propagated to the next time step $t+1$ resulting in the measure $\{x_{t+1}^i, \frac{1}{N}\}$ for the approximation of the density $p(x_{t+1}^i|z_{1:t})$.

2.7 Regularized PF (RPF)

The RPF resamples from a continuous approximation of the posterior density (equation 2.13) in contrast to the SIR filter by replacing the Dirac-delta function with a kernel function.

$$p(x_t|Z_t) \approx \sum_{i=1}^{N} w_t^i K_h(x_t - x_t^i) \tag{2.27}$$

where

$$K_h(x) = \frac{1}{h^{d_x}} K(\frac{x}{h}) \tag{2.28}$$

is the rescaled kernel density $K(.)$, $h > 0$ is the kernel bandwidth, d_x is the dimension of the state vector x and $w_t^i, i = 1, ..., N$ are normalized weights.

In the special case of an equally weighted sample, the optimal choice of the kernel K_{opt} is the Epanechnikov kernel. But to reduce the computational cost, the samples can be drawn from the Gaussian kernel instead of the Epanechnikov kernel and the optimal choice for the bandwidth for a Gaussian kernel is given by [14]:

$$h_{opt} = A \times N^{-\frac{1}{d_x+4}} \text{ with } A = \left(\frac{4}{d_x+2}\right)^{\frac{1}{d_x+4}} \tag{2.29}$$

The main difference of the RPF to the generic PF is only in the additional regularization steps when conducting resampling. The RPF steps are summarized in Algorithm 4.

2.8 Resampling Operations

Resampling is an essential step for PF because without this step the particle weight degeneracy problem reveals [2]. The degeneracy of particles weights problem is the unbounded increase of the variance of the importance weights w_t^i of the particles with time which results in inaccurate estimates with unacceptably large variances. For preventing such a growth in the variance a resampling operation must be employed. Resampling aims to prevent the degeneracy of the propagated particles by replacing an old set $S_t = \{x_t^i, w_t^i\}_{i=1}^N$ of N particles by a new set $S_t^* = \{x_t^{i*}, w_t^{i*}\}_{i=1}^N$ based on their weights w_t^i. Thus the particles from S_t with large weights are more likely to dominate S_t^* than particles with small weights, and, consequently, in the next time step, more new particles will be generated in the region of large weights. This enables to focus on the exploration of the state space to the parts with large probability masses.

Algorithm 4 Regularized PF

Input: $\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^N, z_t$
1: **for** $i = 1 : N$ **do**
2: Draw $x_t \sim p(x_t|x_{t-1})$
3: Calculate the non normalized importance weight $\tilde{w}_t^i = p(z_t|x_t^i)$
4: **end for**
5: Calculate sum of weights $W = \sum_{i=1}^N \tilde{w}_t^i$
6: **for** $i = 1 : N$ **do**
7: Normalize: $w_t^i = W^{-1}\tilde{w}_t^i$
8: **end for**
9: Calculate N_{eff} using equation 2.20
10: **if** $N_{eff} \leq N_T$ **then**
11: Calculate the empirical covariance matrix E_t of $\{x_t^i, w_t^i\}_{i=1}^N$
12: Compute D_t such that $D_t D_t^T = E_t$
13: Resample $[\{x_t^i, w_t^i\}_{i=1}^N, -]$ =RESAMPLE$[\{x_t^i, w_t^i\}_{i=1}^N]$
14: **for** $i = 1 : N$ **do**
15: Draw $\epsilon^i \sim K$ from Epanechnikov / Gaussian kernel
16: $x_t^{i*} = x_t^i + h_{opt} D_k \epsilon^i$
17: **end for**
18: **end if**
19: **return** $\{x_t^i, w_t^i\}_{i=1}^N$

Even if resampling is intended to prevent the degeneracy problem it may introduce undesired effects such as *sample impoverishment* [11, 18, 19]. Sample impoverishment is the reduction in diversity among the particles which results due to replicating many instances of particles with large weights and removing those particles with low importance weights with the resampling operation. For example, if few particles of S_t have most of the weight, many of the resampled particles in the new set S_t^* will be the same and as a result the number of different particles in S_t^* will be small. To reduce the effect of sample impoverishment, resampling operation is normally done by measuring the variance of the particle weights with the effective sample size parameter N_{eff}. This parameter provides a measure of the variance of the particle weights. e.g. this parameter tends to 1 while one single particle carries the largest weight and the rest have negligible weights in comparison. In addition to the sample impoverishment problem, resampling also limits the full parallelization

of the PF computations due to the serial nature of most resampling method [20–25].

Due to the undesired effects of resampling, different advanced methods for resampling have been proposed, such as multinomial resampling, systematic resampling, residual resampling, stratified resampling and independent Metropolis Hastings. As the resampling is a computational intensive step, a justified decision regarding which resampling algorithm to use might result in a reduction of the overall computational effort of the PFs. Therefore, for facilitating on the decision of which resampling algorithm to adopt, this section describes the different strategies for performing the resampling operation.

2.8.1 Multinomial Resampling

Multinomial resampling [11] is the most straightforward method, where N independent random numbers u^j, $j = 1, ..., N$ are generated from a uniform distribution $(0, 1]$ in order to pick a particle from the approximate discrete posterior density. In the jth selection, a particle with index i is chosen while the condition in equation 2.30 is satisfied. Thus, the probability of selecting a particle with index i is the same as that of u^j being in the interval bounded by the cumulative sum of the normalized weights given in Equation 2.31.

$$C^{i-1} \leq u^j \leq C^i \tag{2.30}$$

where

$$C^i = \sum_{j=1}^{N} w_t^j \tag{2.31}$$

Multinomial resampling has a computational complexity in the order of $O(NM)$, where the M factor arises from the search of the required j in equation 2.30 which makes it an inefficient method [9]. The multinomial resampling method is summarized in Algorithm 5.

Algorithm 5 RESAMPLE: Multinomial

1: Generate N uniform independent random numbers $u^j \sim U(0,1)$
2: Calculate the cumulative sum of weights $C^i = \sum_{j=1}^{N} w_t^j$
3: Find C^i so that $C^{i-1} \leq u^j \leq C^i$, the particle with index i is chosen
4: Given $\{x_t^i, w_t^i\}$, for $j = 1, ..., N$, generate new samples x_t^j by duplicating x_t^i according to the associated w_t^i
5: Reset $w_t^i = \frac{1}{N}$

2.8.2 Stratified Resampling

In the stratified resampling [26], the N particles are divided into sub particles called strata. The $(0, 1]$ interval is partitioned into N disjoint sub-intervals, where the N random numbers u^i, $i = 1, ..., N$ are drawn independently in each of these sub-intervals given by Equation 2.32.

$$u^i = \frac{(i-1) + \tilde{u}_l}{N} \tag{2.32}$$

Then the bounding method based on the cumulative sum of normalized weights given by equation 2.33 is used.

$$max\Big(\Big\lfloor Nw^j \Big\rfloor - 1, 0\Big) \leq u^j \leq \Big\lfloor Nw^j \Big\rfloor + 2 \tag{2.33}$$

Algorithm 6 RESAMPLE: Stratified

1: Generate N ordered random numbers
$u_i = \frac{(i-1)+\tilde{u}_l}{N}$, with $\tilde{u}_l \sim U(0,1)$
2: Use u_i to select a particle with index i according to the multinomial distribution

2.8.3 Systematic Resampling

Systematic Resampling [9, 15] is the most popular method where it applies the idea of strata in a different way. Here the first random number u^1 is drawn from the interval $(0, 1/N]$ and the rest of the random numbers are obtained deterministically by applying Equation 2.34.

$$u^i = u^1 + \frac{i-1}{N}, i = 2, ..., N \tag{2.34}$$

The upper and lower limits of the times the jth particle is resampled in the systematic method is given by Equation 2.35, where $\lfloor\rfloor$ denotes the floor operation.

$$\left\lfloor Nw^j \right\rfloor \leq u^j \leq \left\lfloor Nw^j \right\rfloor + 1 \tag{2.35}$$

Both systematic and stratified resampling has complexity in the order of $O(N)$. However, the systematic method is computationally more efficient than the stratified method as it only requires the generation of a single random number.

Algorithm 7 RESAMPLE: Systematic

1: Sample $u_1 \sim U(0, \frac{1}{N})$ and define $u_i = u_1 + \frac{i-1}{N}$ for $i = 2, ..., N$
2: Use u_i to select a particle index i according to the multinomial distribution

2.8.4 Residual Resampling

Residual Resampling [17, 19] method involves two stages. First particles are resampled deterministically by picking $k_i = Nw^i$ copies of the $i'th$ particle followed by multinomial sampling on the residual weights as shown in Algorithm 8.

Algorithm 8 RESAMPLE: Residual

1: For $i = 1 : N$, retain $k_i = Nw^i$ copies of x_t^i
2: Let $N_r = N - k_1 - ... - k_N$, obtain N_r i.i.d. draws from $\{x_t^i\}$ with probabilities proportional to $Nw^i - k_i$ where $i = 1 : N$
3: Reset $w_t^i = \frac{1}{N}$

2.8.5 Independent Metropolis Hastings Algorithm (IMHA)

In PFs, the sampling and importance weight computations can be easily parallelized and pipelined as there is no data dependencies between them. However the resampling step, in particular with the traditional resampling methods explained so far, is hard to be pipelined with other steps as it requires the knowledge of all normalized weights of particles. Thus resampling with traditional methods create bottleneck in the full parallelization of the whole PF steps computations. In order to overcome such bottleneck, resampling methods such as the Metropolis Hasting Algorithm (MHA)[27–29] which requires only ratios between weights that do not need

to be normalized can be used. MHA is suitable for parallel processing as knowledge about all the particle weights is not required and the resampling computation can start as soon as the first particle weight becomes available [30]. In particular from hardware implementation point of view, the use of the MHA for resampling has the advantage of a bottleneck free operation compared to traditional resampling methods and reduces the latency of the whole PF implementation [31, 32].

MHA is a Markov Chain Monte Carlo (MCMC)[33–35] based sampling method used for generating samples from a desired posterior density $p(x_t)$ by generating samples from an easy to sample proposal distribution $q(x_t)$. Compared to traditional resampling methods MHA has the advantage that it does not require the knowledge about all the particles as it can produce a sequence of Markov chain states in which the current state x_t^{i+1} depends on the previous state x_t^i. The MHA steps for generating sequence of Markov chain states is described in Algorithm 9. The first step of Algorithm 9 is to initialize the chain with the first particle index, i.e. $(x_t^1, \tilde{w}_t^1)$. Then in the main loop three operations are performed: (1) Generate a proposal (or a candidate) particle x_t^* from the proposal distribution $q(x_t^*|x_t^i)$; (2) Compute the acceptance probability P_a via the acceptance function $\alpha\left(x_t^i, x_t^*\right)$ based upon the proposal distribution and the posterior density $p(.)$; (3) Accept the candidate particle with probability P_a, the acceptance probability, or reject it with probability $1 - P_a$.

Algorithm 9 The Metropolis Hasting algorithm (MHA)

1: Initialize the chain with $(x_t^1, \tilde{w}_t^1)$
2: **for** $i = 2 : N$ **do**
3: From x_t^i draw samples and compute corresponding weights $(x_t^*, \tilde{w}_t^*)$ from $q(x_t^*|x_t^i)$
4: Compute the acceptance probability, $P_a = \alpha\left(x_t^i, x_t^*\right)$

$$\alpha\left(x_t^i, x_t^*\right) = \min\left\{\frac{p\left(x_t^*\right)}{p\left(x_t^i\right)}\frac{q\left(x_t^i|x_t^*\right)}{q\left(x_t^*|x_t^i\right)}, 1\right\}$$

5:

$$\left(x_t^{i+1}, \tilde{w}_t^{i+1}\right) = \begin{cases}\left(x_t^*, \tilde{w}_t^*\right), & \text{with prob. } P_a \\ \left(x_t^i, \tilde{w}_t^i\right), & \text{with prob. } 1 - P_a\end{cases}$$

6: **end for**

Independent Metropolis Hastings algorithm (IMHA)[2, 32, 36, 37] is a special case of the general MHA where $q\left(x_t^*|x_t^i\right)$ is independent of x_t^i. This lead to an acceptance probability P_a, given by Equation 2.36.

$$P_a = \min\left\{\frac{p\left(x_t^i\right)}{p\left(x_t^*\right)}\frac{q\left(x_t^*\right)}{q\left(x_t^i\right)}, 1\right\} \tag{2.36}$$

The acceptance probability in Equation 2.36 can be evaluated using the non normalized importance weights $\tilde{w}_t^*$ and $\tilde{w}_t^i$ of a proposed particle x_t^* and previous particle x_t^i respectively by using Equation 2.37[32]. A proposed particle x_t^* is accepted or rejected by comparing its acceptance probability P_a with a uniform random number $u \sim (0, 1)$ as shown in Equation 2.38.

$$P_a = \min\left\{\frac{\tilde{w}_t^*}{\tilde{w}_t^i}, 1\right\} \tag{2.37}$$

$$\left(x_t^{i+1}, \tilde{w}_t^{i+1}\right) = \begin{cases} \left(x_t^*, \tilde{w}_t^*\right), & u \leq \min\left\{\frac{\tilde{w}_t^*}{\tilde{w}_t^i}, 1\right\} \\ \left(x_t^i, \tilde{w}_t^i\right), & u > \min\left\{\frac{\tilde{w}_t^*}{\tilde{w}_t^i}, 1\right\} \end{cases} \tag{2.38}$$

2.9 PFs Applications

PFs have been used for estimating the unknown states of a system from noisy measurements in many important applications of today's high technology. These applications include; robotics [38–42], positioning and navigation [43, 44], computer vision [45–47], communication [48–50], biomedical applications [51–53], financial time series analysis and econometrics [54, 55], whether forecasting [56] and many other challenging problems. Due to the different characteristics of each application, in our research, we mainly focus on the application of the PFs to one of the fundamental problems of mobile robotics field, the *Simultaneous localization and mapping* (SLAM) problem [57–61].

In mobile robotics, for autonomous navigation the knowledge of the environment and the position of the robot within the environment is mandatory. If the robot position is provided along with its trajectory, the map can be easily constructed

through the information coming from robot sensors [62]. Similarly if the true map of the environment were available, estimating the path of the robot would be a straightforward localization problem [63, 64]. However, as the map is not available in most of the practical applications, the robot must be able to build a map of the surroundings environment and to determine its location within the map. Such inter dependency between the position of the robot and the map poses a challenging problem. SLAM is one of the solution to solve such a problem through a probabilistic approach. The goal of SLAM is to reconstruct a map of the world and the path taken by the robot while the robot makes relative observations of its motion and of the objects in its environment, both corrupted by noise.

SLAM has diverse applications for generating maps of unknown environments which are dangerous or inaccessible to humans. For example, SLAM is used for deep sea explorations[65, 66], for autonomous terrain vehicles involved with tasks such as mining and construction[67, 68] and autonomous planetary and space exploration of the solar system [69–72]. SLAM is also used to obviate or assist navigation systems in indoor and outdoor environments where the GPS information is unavailable [73–78]. In addition, it can be also used to improve object-recognition systems [79, 80], which will be a vital component of future robots that have to manipulate the objects around them in arbitrary ways.

2.9.1 SLAM Principle

The basics of all SLAM algorithms is based on two alternating steps, *prediction* and *update*, for estimating the *pose* and *map* of a mobile robot. The pose of a robot moving on a 2D plane is described by $x_t = [x, y, \theta]^T$, where x and y represent the position of the robot in some fixed global coordinate system and θ is the angle between the bearing of the robot and the positive x-axis, as shown in Fig. 2.5. The pose and map estimates covers the *localization* and *mapping* part of the SLAM problem respectively (Fig. 2.6).

The prediction step is used to determine a new robot pose from the previous pose using a *motion model* and possibly taking motion sensor readings as inputs

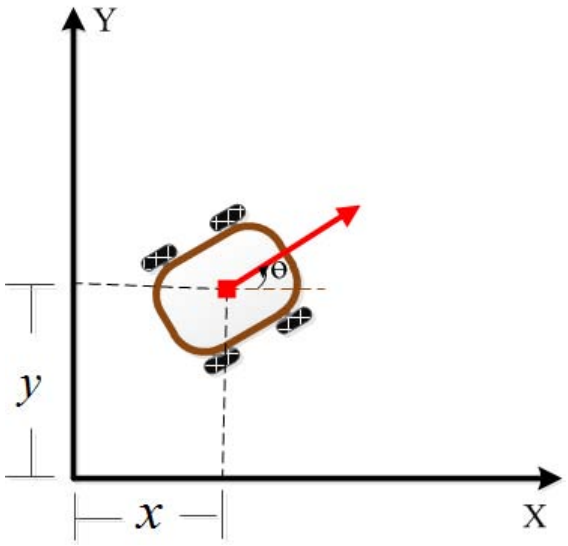


Figure 2.5: 2D parameters of a robot pose

(Fig. 2.6). Commonly used motion sensors include odometry, inertial measurement units (IMU) and GPS. The update step corrects the predicted robot pose based on observations of the environment and by applying an *observation model* (Fig. 2.6). Measurement sensors are used for gathering information about the structures of the surrounding environment and build a map. Typically robots are equipped with range sensors such as laser range finders, sonars, cameras, etc, that returns

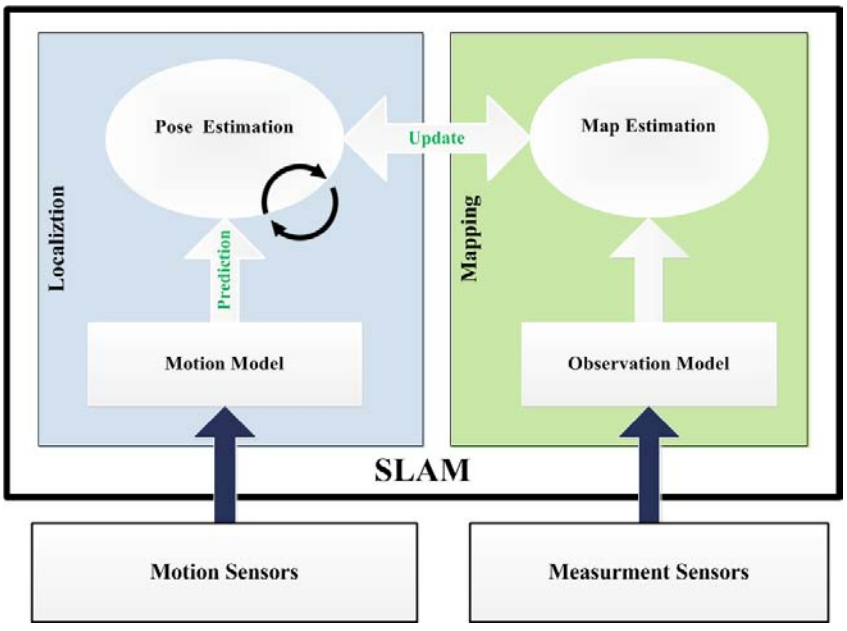


Figure 2.6: The SLAM principle

a set of distance measurements to obstacles in its field of view.

2.9.2 Localization

Localization is the process by which a mobile robot keeps track of its position as it moves around an environment and it is a key problem in making truly autonomous robots. For localization a robot rely on its control actions and sensors to determine its location as accurate as possible. However, the existence of uncertainty in both the control actions and the sensing of the robot makes the problem of localization difficult. Therefore, the uncertainty in the information needs to be combined in an optimal way. This naturally leads to the consideration of probabilistic methods, in which the spatial state of the robot is represented as a probability distribution over the space of possible robot poses. The problem of localization is then the problem of updating the distribution, based on robot motion and sensing, given a map of the environment that may be imperfect.

In robot localization, we are interested in estimating the robot pose at the current time step t, based on the knowledge about the initial state x_0 which is assumed to be available in the form of a probabilistic density $p(x_0)$ and all measurements $z_{1:t}$ up to the current time t. Mathematically, probabilistic robot localization consists of estimating the distribution $p(x_t|z_{1:t}, u_t, m)$ of the hidden dynamic variable x_t at time step t, given sensor observations $z_{1:t}$, a map of the environment m and robot actions u_t (normally, odometry increments). Fig. 2.7 shows a dynamic Bayesian network representation of the localization problem, where the sequence of robot poses constitute a Markov process (i.e. given a pose x_t, the pose at the next instant x_{t+1} is conditionally independent of all previous poses). The computation of the distribution $p(x_t|z_{1:t}, u_t, m)$ at each time step t is performed with the two (prediction and update) step recursive Bayesian estimation.

1. In the prediction step a *motion model* is used to predict the current position of the robot in the form of a predictive PDF $p(x_t|z_{1:t-1})$ taking only motion into account. It is assumed that the current state x_t is only dependent on the previous state x_{t-1} (Markov) and a known control input u_{t-1} and the

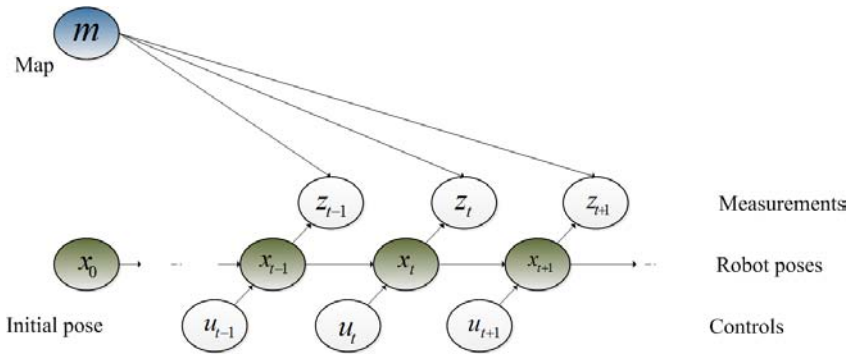


Figure 2.7: Mobile robot localization as dynamic Bayes network

motion model is specified as a conditional PDF $p(x_t|x_{t-1}, u_{t-1})$. The predictive density over x_t is then obtained by integration using equation 2.39.

$$p(x_t|z_{1:t-1}) = \int \underbrace{p(x_t|x_{t-1}, u_{t-1})}_{motion\ model} \underbrace{p(x_{t-1}|z_{1:t-1})}_{prior} dx_{t-1} \tag{2.39}$$

2. In the update step the posterior PDF $p(x_t|z_{1:t})$ is computed using a *measurement model* given in terms of a likelihood $p(z_t|x_t)$ to incorporate information from the sensors. The measurement likelihood expresses the likelihood that the robot is at location x_t given that z_t was observed. The measurement z_t is assumed conditionally independent of earlier measurements z_{t-1} given x_t. The posterior density over x_t is obtained using Bayes theorem:

$$p(x_t|z_{1:t}) = \frac{\overbrace{p(z_t|x_t)}^{observation\ model} \quad p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{2.40}$$

The solution to the robot localization problem is obtained by recursively solving the prediction and update equations 2.39 and 2.40 respectively. A PF based solution to robot localization problem is commonly know as *Monte Carlo localization* (MCL) [81–83]. MCL applies models of various sensors with the application of PFs for generating the belief state of the robot's location. In MCL, rather than approximating posteriors $p(x_t|z_{1:t})$ in parametric form, as is the case for Kalman filter [84, 85], MCL simply represents the posteriors by N random set of weighted

particles which approximates the desired distribution, i.e. $p(x_t|z_{1:t}) \approx \{x_t^i, w_t^i\}_{i=1}^N$. The use of PFs approach in MCL has the following advantages [81]:

- Accommodating arbitrary sensor characteristics, motion dynamics, and noise distributions.
- Focusing computational resources in areas that are most relevant by sampling in proportion to the posterior likelihood.
- Avoid the Gaussian restrictive assumptions on the posterior density compared to parametric approaches which makes PFs a universal density approximators.
- Flexible adaption of available computational resource by on-line monitoring of number of particles.

The recursive update of the particles set $S_t = \{x_t^i, w_t^i\}_{i=1}^N$ which approximates the posterior of PDF $p(x_t|z_{1:t})$ of the robot pose at time t is realized with the application of the SIR PF algorithm with the following steps:

1. *Prediction*: approximate the predictive density $p(x_t|z_{1:t-1})$ starting with set of particles S_{t-1} computed from a previous time step $t-1$ and applying the motion model to each particle x_{t-1}^i by sampling N new particles from the density $p(x_t|x_{t-1}^i, u_{t-1})$.
2. *Update*: taking the measurement z_t into account, evaluate the importance weight, w_t^i, of each particle x_t^i in proportion to the measurement likelihood i.e., $w_t^i \approx p(z_t|x_t^i)$.
3. *Resampling*: generate a new particle set from S_t by selecting particles having higher importance weights with higher probability. The newly generated particle set approximates the desired posterior PDF $p(x_t|z_{1:t})$ and it is used for the recursive update of the particles in the *Prediction* step.

2.9.3 Probabilistic Models

As per the prediction and update equations 2.39 and 2.40 respectively of the localization problem, the computation of the PDF $p(x_t|z_{1:t})$ requires two conditional densities: the probability $p(x_t|x_{t-1}, u_{t-1})$, which is often called motion model and the likelihood density $p(z_t|x_t)$ commonly called observation model or sensor model. The description of the specific probabilistic models used for motion and observation model are given below.

Motion Model

Modeling the inherent uncertainty in robot motion using PDF is by far the most widely used approach in robotics. The uncertainty in the pose x_t is best modeled by a probability distribution $p(x_t|x_{t-1}, u_{t-1})$ over possible poses that the robot might attain after executing the control u_{t-1} in x_{t-1}. The control u_t is normally obtained by integrating odometry data from wheel encoders and it is used as the basis for calculating the robot's motion over time. For example, an *odometry motion model* [81, 86] which is adopted in this work uses the relative information of the robot's internal odometry and transform it into a sequence rotation, followed by a straight line motion (translation) and another rotation. Fig. 2.8 illustrates an odometry model, where the robot motion in the time interval $t-1$ to t is approximated by a rotation δ_{rot1}, followed by a translation δ_{trans} and a second rotation δ_{rot2}. The odometry motion model assumes that these three parameters are corrupted by independent noise. For a small motion segment, these errors can be modeled by independent Gaussian noise variables, which are added to the commanded translation and rotation [87].

Observation Model

The observation model, physically models the inherent noise in the robot's sensors. In our case we use mainly laser-range finders as the robotic sensor, which are very common in robotics. The signal of a laser-range finder is emitted in a beam that provide the distance information to a nearest obstacle. By combining several of the

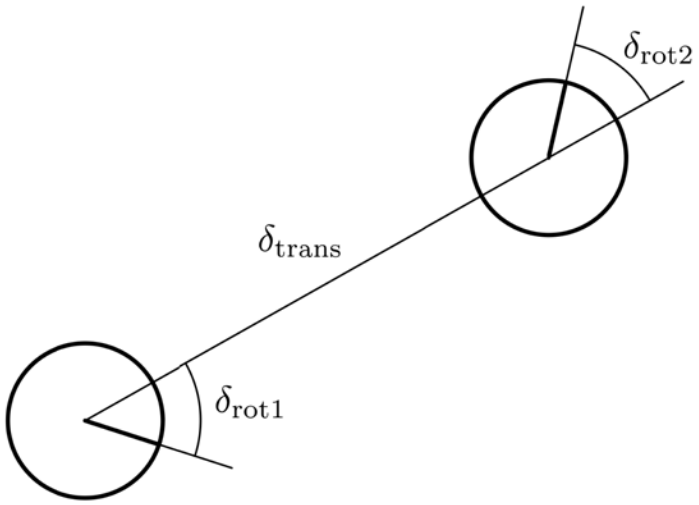


Figure 2.8: Odometry motion model [86].

beams, laser-range finders provide a two dimensional scan of the robot environment. At each time t a complete scan Z_t is provided, which is a combination of several distance measurements from the individual laser beams z_t^n, where n is the index of a beam. It is assumed the distance measurements z_t^n in the scan Z_t are independent and therefore the beams are considered individually.

The laser range finder measurement error is modeled by a Gaussian distribution with mean z_t^n of the actual laser measurement at time t, equation 2.41. Where m is the map of the robot environment represented by fine-grained metric grid cells, k represents the index of a gird cell and $\delta(z_t^n)$ is the standard deviation function which can be obtained from experiments for the specific laser scanner. Under the assumption that the map m and the robot pose x_t is known, the observation model specifies the probability that Z_t is measured. The probability that the range measurement with an index n is returned by a grid cell m_k along the path of the laser beam is given by:

$$p_o(m_k^n|z_t^n) = \frac{1}{\delta(z_t^n)\sqrt{2\pi}} e^{\frac{-(m_k^n - z_t^n)^2}{2\delta^2(z_t^n)}} \tag{2.41}$$

Beside the error model of the sensor, the observation model considered in this work takes into account a *scan matching* to determine how well the current scan

matches the occupied points in the map. It assumes that each scan induces a local map, which can be conceptually decomposed into three types of areas: free space, occupied space and occluded space. The same conceptual decomposition applies to the map. Each of the the measurements z_t^n in a range scan Z_t can thus fall into three different regions in the map. If it falls into the occluded (unknown) space, the probability is uniformly high. If it coincides with occupied space, the probability is high, too. While it falls into free space, the probability is low. The specific probability of the measurement is then given by a function that decreases monotonically with the distance to the nearest object. The probability of the complete laser path is computed by taking the product of the probability that the laser is passing through unoccupied grid cells, and the probability that it will be stopped by an obstacle in the grid cell that terminates the path. For grid cells along the path of the laser, the probability that the laser measurement will have traveled through unoccupied grid cells and stopped by an obstacle at grid cell m_k is given by [88]:

$$p(m_k = occupied) = p_o(m_k) \prod_{j=1}^{k-1} \Big(1 - p_o(m_j)\Big) \tag{2.42}$$

Considering a total of M laser beams in the current laser scan Z_t, the scan matching between Z_t and the occupied points in the map is obtained by the function $f_{sm}(Z_t)$ given by equation 2.43. In equation 2.43, those laser scans which fall in unoccupied or unexplored grid cells contribute small values to the final value of the scan matching function $f_{sm}(Z_t)$, when compared to the occupied points in the evaluation of scan matching. Therefore, this equation is maximized while the current laser scan points match well with the occupied points in the map. The scan-matching is applied per particle basis.

$$f_{sm}(Z_t) = \sum_{n=1}^{M} p(m_{t-1}|z_t^n) \tag{2.43}$$

The total probability of the laser scan Z_t at time t is obtained by taking the product of the $f_{sm}(Z_t)$ and sum of the $p(m_k = occupied)$, as shown in equation 2.44.

$$f_{sm}(Z_t) \sum_{n=1}^{M} p_o(m^k) \prod_{j=1}^{k-1} \Big(1 - p_o(m^j)\Big) \tag{2.44}$$

2.9.4 Robotic Mapping

Robotic mapping addresses the problem of acquiring spatial models of physical environments through mobile robots. Map is a data model of the environment which is used for autonomous navigation by robots [89]. There exists different types of models for the representation of the environment, where the two major categories are topological and metric maps[90–92].

Topological maps are a sparse representation of the environment where only key places for navigation are represented by using measurement data and connections between these places using position information data from sensors. In general, topological maps are represented as graphs, where nodes represent distinctive places and arcs that connect nodes represent path information between places. An example of a 2D topological map of an office environment is shown in Fig. 2.9 (right), where the nodes represent distinctive places such as corridors (C), doors (D) and rooms (R) and arcs that connect nodes represent path information between distinctive places. In Fig. 2.9 (right) the positions of two points P1 and P2 in the environment may be recognized as being part of the rooms. This makes it possible to infer that position P2 can be reached from position P1 via the different nodes.

In case of metric maps, the position of the obstacles encountered by the robot in the environment are stored in a common reference frame. In metric maps

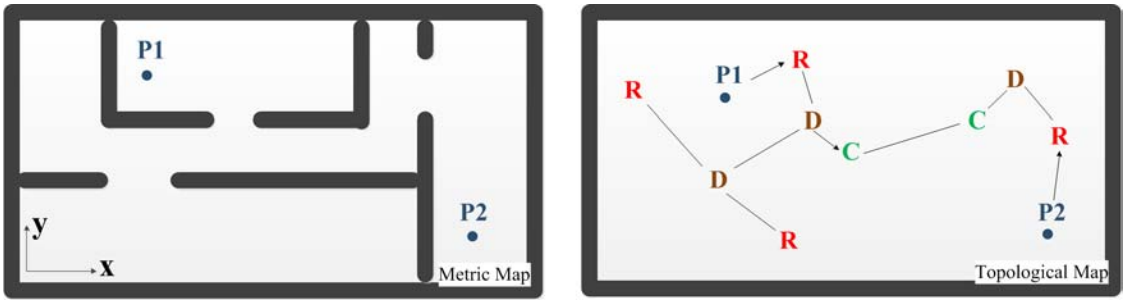


Figure 2.9: Environmental representations for robotic mapping.

measurement information is stored after transformation in the 2D space by means of a sensor model. This transformation results a set of objects or obstacles along with their positions relative to the robot. An example of a metric map is shown in Fig. 2.9 (left), where the position of two points P1 and P2 in the environment are represented by their coordinates in respect to a common reference frame and the distance of the two points can be easily inferred from their coordinates.

Metric maps are much more precise than the topological framework as the position estimate is continuous in the 2D space [93]. Moreover, metric maps display the layout of the environment in easy to read format which is independent of any given robot. This makes it easy for different robots to reuse such maps. Metric maps are also easier to build than topological maps due to the non-ambiguous definition of locations by their coordinates. However, metric map building often heavily depends on the quality of the position estimates from the odometry sensors. In respect to metric maps, topological maps have the advantage as do not require a metric sensor model to convert sensor measurement data in a common 2D reference frame. The only requirement is a method for storing place definitions and for recognizing places given sensor data. However, as the sensor data is available for places physically explored by the robot, topological maps require exhaustive exploration of the environment when higher precision for position estimation is required. In addition, in case of unreliable sensors and in dynamic environments, it introduces a difficulty for the definition of places.

Depending on the representation of obstacles in the environment, metric maps are further classified into feature, geometric, and grid maps[94–96]. A feature map stores a set of features such as lines and corners detected in the environment. For each feature, these maps store the feature information together with a coordinate and eventually an uncertainty measure. In geometric maps obstacles detected by the robot are represented by geometric objects, like circles or polygons. In the case of grid maps the environment is modeled by discretized grid cells, where each cell stores information about the area it covers. Most frequently used are occupancy grid maps that store for each cell a single value representing the probability that

this cell is occupied by an obstacle. The advantage of grids is that they do not rely on predefined features which need to be extracted from sensor data and doesn't suffer from data association problem [97]. Furthermore, they offer a constant time access to grid cells and provide the ability to model arbitrary environments [98]. However, it suffers from discretization errors and requires lot of memory resources.

Occupancy Grid Mapping (OGM)

The statistical dependency of the noise in different measurements, makes robotic mapping problem challenging [64]. This is because errors in control accumulate over time, and they affect the way future sensor measurements are interpreted. Probabilistic techniques are normally used for robotic mapping as robot mapping is characterized by uncertainty and sensor noise. Occupancy grid mapping algorithm is one of the probabilistic techniques used to address the problem of generating a consistent metric map from noisy sensor data. Occupancy grid maps normally generate two-dimensional probabilistic maps, which may also extend to three spatial dimensions [99–101], represented as grids.

The standard occupancy grid mapping algorithm is a version of Bayes filters used to calculate the posterior over the occupancy of each grid cell. For generating an occupancy grid map, it is required to determine the occupancy probability of each grid cell. The assumption of independence among the grid cells is often considered for an efficient generation of the occupancy grid map. Even if such an assumption is not accurate in particular while considering the fact that adjacent grid cells could represent the same object in the environment, it greatly facilitates and simplifies the mapping algorithm without the introduction of significant errors. With such an assumption, the probability of a particular map m can be factorized into the product of the probabilities of the individual grid cells. Therefore, the probabilistic representation of the map of the robot's environment is given by:

$$p(m|x_{0:t}, z_t) = \prod_{k=0}^{n-1} p\Big(m_i|x_{0:t}, z_t\Big) \tag{2.45}$$

In equation 2.45, $x_{0:t}$ represents the history on the position of the robot until a given time step t and z_t is the corresponding measurement data taken at each position of the robot. The computation of the k'th grid cell's probability $p(m_k|x_{0:t}, z_t)$ is conditioned on the position of the robot and measurement data. As a result it can be easily computed given these two information as it is determined by whether the robot sees it as occupied or free. The occupancy grid mapping evaluates the grid cells probabilities iterativly considering the position and measurements starting from time step $t = 0$ and to the most recent reading. Therefore, with occupancy grid maps, the mapping step determines the occupancy probability of each cell. The map cells are updated iteratively according to the position and measurement data which would require significant processing for updating the entire map at each step. However, as the processing of the entire map is unnecessary only those grid cells visible to the robots are updated. Each grid cell that is perceived by the robot's sensor at a given position is updated depending on whether the sensor reports it as occupied or free. The success of an accurate mapping depends on the position $x_{0:t}$ being correct. A typical 2D example of an occupancy grid map constructed from laser range finders and a 3D occupancy grid map of an outdoor environment are shown in Fig. 2.10.

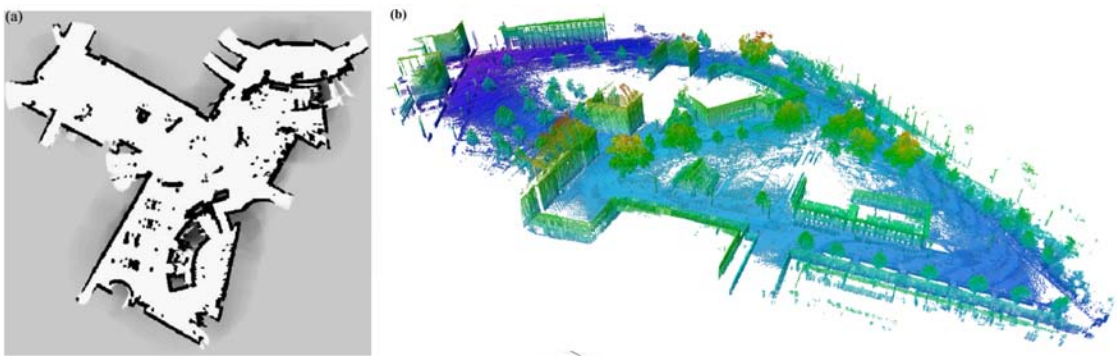


Figure 2.10: Typical 2D occupancy grid map (a) [64] and 3D Occupancy grid map (b) [99].

2.9.5 The SLAM Solution

It is natural to formulate the SLAM problem using probabilities to incorporate the uncertainties originating from the robot sensors, environmental factors and the control action given to the robot actuators. From a probabilistic perspective, there are two main forms of the SLAM problem: one is called the *full SLAM* problem and the other the *online SLAM* problem (Fig. 2.11).

The full SLAM problem seeks to calculate the posterior probability over the entire path, $x_{1:t}$, along with the map, i.e.

$$p(x_{1:t}, m|u_{1:t}, z_{1:t}) \tag{2.46}$$

where $x_{1:t}$ denotes the pose estimates from the first time instance up to time t, m is the map, $z_{1:t}$ and $u_{1:t}$ are the measurements and controls respectively. Since the full SLAM problem is formulated in such a way that it seeks an estimate over all the previous measurements and controls, it is often an offline algorithm concerned with finding out what the robot has already done.

The online SLAM problem requires the posterior over the current pose x_t along with the map, and is formulated as:

$$p(x_t, m|u_{1:t}, z_{1:t}) \tag{2.47}$$

This problem is called the online SLAM problem since it only involves the estimation of variables that persist at time t. The online SLAM problem is the

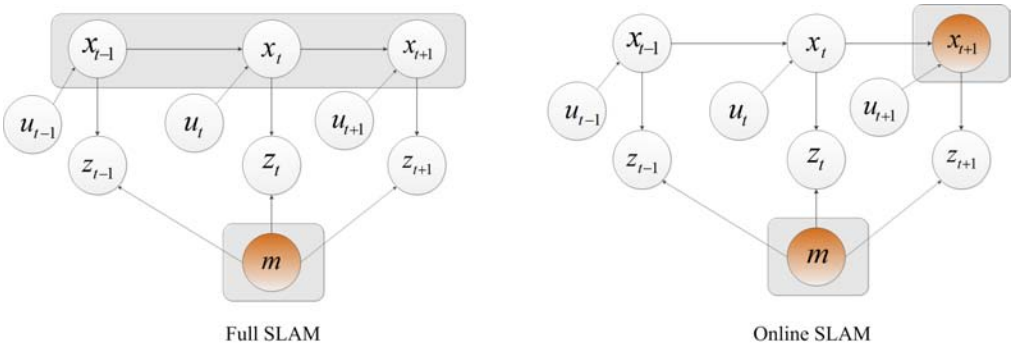


Figure 2.11: Graphical model of online and full SLAM problems, adopted from [86].

result of integrating out past poses from the full SLAM problem:

$$p(x_t, m|u_{1:t}, z_{1:t}) = \int\int ... \int p(x_{1:t}, m|z_{1:t}, u_{1:t})\, dx_1\, dx_2\, ...\, dx_{t-1} \tag{2.48}$$

Therefore, the solution to the online SLAM problem requires an incremental algorithm that discards previous measurements and controls after they have been processed. Because of the continuous nature of the pose estimation variables it is intractable to calculate the full posterior in equation 2.48. In practice, SLAM algorithms rely on approximations.

Taking into account the initial state of the robot, x_0, the online SLAM problem is given by a joint posterior probability density distribution $p(x_t, m|u_{1:t}, z_{1:t}, x_0)$. A recursive Bayes solution for the computation of the joint PDF is used by starting with an estimate for the distribution at time $t-1$, $p(x_{t-1}, m|z_{1:t-1}, u_{1:t-1})$, and following a control u_t and observation z_t. Therefore, the SLAM algorithm is implemented with the two standard recursive state update and measurement update steps. Where the recursion is a function of the robot motion model $p(x_t|x_{t-1}, u_t)$ and an observation model $p(z_t|x_t, m)$. The two recursive state and measurement updates are given by [102]:

SLAM state update:

$$p(x_t, m|z_{1:t-1}, u_{1:t-1}, x_0) = \int p(x_t|x_{t-1}, u_t) p(x_{t-1}, m|z_{1:t-1}, u_{1:t-1}, x_0)\, dx_{t-1} \tag{2.49}$$

SLAM measurement update:

$$p(x_t, m|z_{1:t}, u_{1:t}, x_0) = \frac{p(z_t|x_t, m) p(x_t, m|z_{1:t-1}, u_{1:t-1}, x_0)}{p(z_t|z_{t-1}, u_{1:t})} \tag{2.50}$$

For the analytical solution of the SLAM state update and measurement update, equations 2.49 and 2.50 respectively, a PF based approach based on MCL (Section 2.9.2) and OGM (Section 2.9.4) is often used [62, 86]. SLAM applies factorization

of the posterior by using a Rao-blackwellization technique [102], which allows first to estimate only the trajectory, using the SIR PF, and then to compute the map given that trajectory. The basic framework for PF-SLAM algorithms based on Rao-blackwellized SIR filter updates the particles following the steps shown in Algorithm 10. The Update map () procedure in Algorithm 10 applies the occupancy grid mapping methods and it is performed by tracing all the grid cells which lies along the path of the laser beams and updating their corresponding occupancy likelihood. However, the occupancy likelihood of those grid cells that are outside the laser's beam range of view remain unchanged. The trace of the grid cells along the path of the laser beam is performed based on the a Bresenham's line tracing algorithm [103] and the occupancy likelihood of each grid cell is read and updated accordingly. Those grid cells that lies between the starting point and end point of the laser beam corresponds to unoccupied points and their occupancy likelihoods are updated by a certain factor f_{free} and for the grid cell that corresponds to the end point of the laser beam, its occupancy likelihood is updated by incrementing its previous value by a factor of $f_{occupied}$. The value $f_{free} = 0.1$ and $f_{occupied} = 0.1$ are heuristic parameter determined from extensive tests.

Some variations of the basic PF-SLAM framework includes; Grid based Fast-SLAM [62, 86], Fast-SLAM [104], GridSLAM [105], DP-SLAM [88], and GMapping

Algorithm 10 Occupancy Grid FastSLAM

1: **for** $i = 1$ to N **do**
2: Sampling
$x_t^i \sim p(x_t|x_{t-1}^i, u_t, m)$
3: Importance weight
$w_t^i = p(z_t|x_t^i, m)$
4: Map estimation
m_t^i =Update map $\left(z_t, x_t^i, m_{t-1}^i\right)$
$\tilde{X}_t = \tilde{X}_t$ U $< x_t^i, w_t^i, m_t^i >$
5: Resampling
for $i = 1$ to N **do**
draw i from $\tilde{X}_t$ with probability $\propto w_t^i$
$X_t = X_t^i$ U $< x_t^i, m_t^i >$
end for
6: **end for**

[106]. In Fast-SLAM the map is represented with landmarks and uses a PF to estimate the robot's pose and particle has an associated set of independent Extended Kalman Filters (EKFs) that each tracks the parameters of one landmark. GridSLAM is an extension of FastSLAM where an occupancy grid is used and performs scan-matching as a way to maximize the likelihood of odometry measurements. GMapping addresses the problem of particle depletion that results due to resampling [14] by applying a selective resampling based on the measurement of particle variance. DP-SLAM is an occupancy grid map based SLAM technique where the cost of copying the map of particles during the resampling is addressed by maintaining a single map that is shared by all particles and applies a tree-like data structure to keep record of all updates done by all different particles. Grid based FastSLAM [86, 107] method which is adopted in this work and shown in Fig. 2.12 is performed by combining the MCL particle filtering with the occupancy grid mapping algorithm using Rao-Blackwellized PFs.

The use of occupancy grids in PF framework to solve the SLAM problem can be memory intensive. There exists two primary approaches to reduce the amount of memory required when applying PFs and occupancy grids in SLAM. The first approach [105] aimed at reducing the number of particles required to produce a

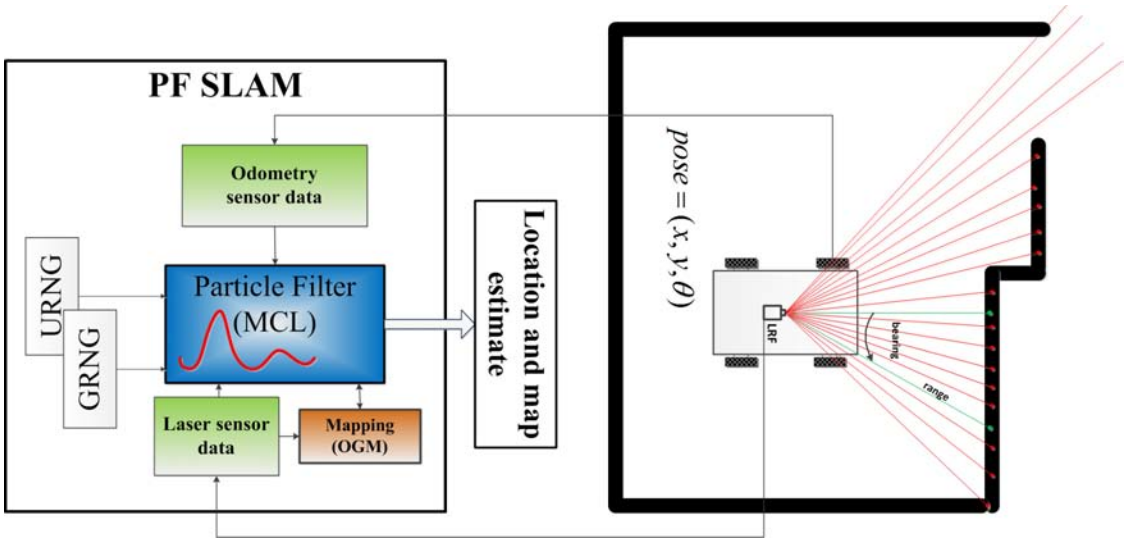


Figure 2.12: Illustration of 2D SLAM. The odometry and laser range finder (LRF) sensors data are used for performing SLAM based on MCL and OGM.

good estimate by improving the distribution from which potential particles are sampled. The approach considers the use of corrected odometry by applying scan matching procedure. However, for low cost laser range finders performing a proper scan matching procedure can be difficult due to noisy measurements (especially for large range measurements). The second approach is DP SLAM, where an alternative method of storing the occupancy grid for each particle is considered. In DP-SLAM approach, instead of each particle maintaining its own occupancy grid, a single occupancy grid map is maintained to all the particles. Each cell in the occupancy grid maintains a tree structure. Every time a particle makes an observation that affects a cell in the occupancy grid a node is added to the tree structure containing the ID of the particle that made the observation and value of the observation. This approach is elegant and has been shown to be computationally efficient, however it introduces additional complexity.

Besides the heavy memory requirement in Grid-based Fast SLAM due to the representation of robot's environment by a metric grid map, it could also lead to a heavy time cost at the resampling step, when the entire map has to be copied over from an old particle to a new one [88, 107]. To overcome both downfalls, in this thesis a different data structure is conducted for maintaining the particles map without additional complexity. The proposed approach maintains a single global occupancy grid map that is shared by all particles. However, instead of maintaining a tree structure at each grid cell like DP-SLAM, in the proposed method each grid of the global map stores an observation array that is indexed with the lists of all particles that have made observation on this grid and their observation values. While accessing a given particle's current estimate for a particular grid while computing its weight, one can simply retrieve the observation value at the index of the given particle. Initially, the map is initialized only with pointers at each grid cells. While a given grid cell is being observed by a particle a dynamic vector is associated with that grid cell and the occupancy likelihood of the particle is inserted at the index of the given particle in the dynamic vector. In general, if there are N particles and the map is $L \times L$, the proposed approach stores to the worst case N scalars

in all $H << L \times L$ observed grid cells, assuming that all the particles have made the observations in all the H observed grid cells. During resampling step, using the index of the replicated particles the map of a given particle is updated with the occupancy likelihood of its replication particle index only. Such an approach helps to avoid the requirement of copying the particle maps during the resampling step by maintaining the index of the particle in each grid cell.

References

[6] N. Gordon, B Ristic, and S Arulampalam. "Beyond the kalman filter: Particle filters for tracking applications". In: *Artech House, London* (2004).

[1] O. Cappé, S. J. Godsill, and E. Moulines. "An overview of existing methods and recent advances in sequential Monte Carlo". In: *Proceedings of the IEEE* 95.5 (2007), pp. 899–924.

[2] H. N. Nagaraja. "Inference in Hidden Markov Models". In: *Technometrics* 48.4 (2006), pp. 574–575.

[3] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. "Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association". In: *Journal of Machine Learning Research* 4.3 (2004), pp. 380–407.

[4] A. Doucet, N. De Freitas, and N. Gordon. "An introduction to sequential Monte Carlo methods". In: *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 3–14.

[5] A. Doucet, S. Godsill, and C. Andrieu. "On sequential Monte Carlo sampling methods for Bayesian filtering". In: *Statistics and computing* 10.3 (2000), pp. 197–208.

[6] N. Bergman, A. Doucet, and N. Gordon. "Optimal estimation and Cramér-Rao bounds for partial non-Gaussian state space models". In: *Annals of the Institute of Statistical Mathematics* 53.1 (2001), pp. 97–112.

[7] J. Harrison and M. West. *Bayesian Forecasting & Dynamic Models*. Springer, 1999.

[8] N. Bergman. "Recursive Bayesian Estimation". In: *Department of Electrical Engineering, Linköping University, Linköping Studies in Science and Technology. Doctoral dissertation* 579 (1999).

[9] J. Carpenter, P. Clifford, and P. Fearnhead. "Improved particle filter for nonlinear problems". In: *Radar, Sonar and Navigation, IEE Proceedings-*. Vol. 146. 1. IET. 1999, pp. 2–7.

[10] P. Del Moral. "Non-linear filtering: interacting particle resolution". In: *Markov processes and related fields* 2.4 (1996), pp. 555–581.

[11] N. J. Gordon, D. J. Salmond, and A. F. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *Radar and Signal Processing, IEE Proceedings F*. Vol. 140. 2. IET. 1993, pp. 107–113.

[12] K. Kanazawa, D. Koller, and S. Russell. "Stochastic simulation algorithms for dynamic probabilistic networks". In: *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1995, pp. 346–351.

[13] J. MacCormick and A. Blake. "A probabilistic exclusion principle for tracking multiple objects". In: *International Journal of Computer Vision* 39.1 (2000), pp. 57–71.

[14] N. Gordon, B Ristic, and S Arulampalam. "Beyond the kalman filter: Particle filters for tracking applications". In: *Artech House, London* (2004).

[15] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *Signal Processing, IEEE Transactions on* 50.2 (2002), pp. 174–188.

[16] C. Andrieu, A. Doucet, and E. Punskaya. "Sequential Monte Carlo methods for optimal filtering". In: *Sequential Monte Carlo Methods in Practice*. Springer, 2001, pp. 79–95.

[17] J. S. Liu and R. Chen. "Sequential Monte Carlo methods for dynamic systems". In: *Journal of the American statistical association* 93.443 (1998), pp. 1032–1044.

[18] M. Bolić, P. M. Djurić, and S. Hong. "Resampling algorithms for particle filters: A computational complexity perspective". In: *EURASIP Journal on Applied Signal Processing* 2004 (2004), pp. 2267–2277.

[19] E. R. Beadle and P. M. Djuric. "A fast-weighted Bayesian bootstrap filter for nonlinear model state estimation". In: *Aerospace and Electronic Systems, IEEE Transactions on* 33.1 (1997), pp. 338–343.

[20] T. Li, M. Bolic, and P. M. Djuric. "Resampling methods for particle filtering: classification, implementation, and strategies". In: *Signal Processing Magazine, IEEE* 32.3 (2015), pp. 70–86.

[21] J. D. Hol, T. B. Schon, and F. Gustafsson. "On resampling algorithms for particle filters". In: *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*. IEEE. 2006, pp. 79–82.

[22] P. Gong, Y. O. Basciftci, and F. Ozguner. "A parallel resampling algorithm for particle filtering on shared-memory architectures". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE. 2012, pp. 1477–1483.

[23] M. Bolić, P. M. Djurić, and S. Hong. "New resampling algorithms for particle filters". In: *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. II–589.

[24] M. Bolić, P. M. Djurić, and S. Hong. "Resampling algorithms and architectures for distributed particle filters". In: *Signal Processing, IEEE Transactions on* 53.7 (2005), pp. 2442–2450.

[25] A. Athalye, M. Bolic, S. Hong, and P. M. Djuric. "Architectures and memory schemes for sampling and resampling in particle filters". In: *Digital Signal Processing Workshop, 2004 and the 3rd IEEE Signal Processing Education Workshop. 2004 IEEE 11th*. IEEE. 2004, pp. 92–96.

[26] G. Kitagawa. "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models". In: *Journal of computational and graphical statistics* 5.1 (1996), pp. 1–25.

[27] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[28] S. Chib and E. Greenberg. "Understanding the metropolis-hastings algorithm". In: *The american statistician* 49.4 (1995), pp. 327–335.

[29] F. Liang, C. Liu, and R. J. Carroll. "The Metropolis-Hastings Algorithm". In: *Advanced Markov Chain Monte Carlo Methods: Learning from Past Samples* (2010), pp. 59–84.

[30] C. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[31] Y. Bao, J. Chen, Z. Shi, and K. Chen. "New real-time resampling algorithm for particle filters". In: *Wireless Communications & Signal Processing, 2009. WCSP 2009. International Conference on*. IEEE. 2009, pp. 1–5.

[32] A. C. Sankaranarayanan, A. Srivastava, and R. Chellappa. "Algorithmic and architectural optimizations for computationally efficient particle filtering". In: *Image Processing, IEEE Transactions on* 17.5 (2008), pp. 737–748.

[33] C. Geyer. "Introduction to Markov Chain Monte Carlo". In: *Handbook of Markov Chain Monte Carlo* (2011), pp. 3–48.

[34] C. J. Geyer. "Practical markov chain monte carlo". In: *Statistical Science* (1992), pp. 473–483.

[35] G. O. Roberts, J. S. Rosenthal, et al. "Optimal scaling for various Metropolis-Hastings algorithms". In: *Statistical science* 16.4 (2001), pp. 351–367.

[36] W. R. Gilks. *Markov chain monte carlo*. Wiley Online Library, 2005.

[37] W. K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (1970), pp. 97–109.

[38] S. Thrun. "Particle filters in robotics". In: *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2002, pp. 511–518.

[39] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. "Towards robotic assistants in nursing homes: Challenges and results". In: *Robotics and Autonomous Systems* 42.3 (2003), pp. 271–281.

[40] A Canedo-Rodríguez, V Álvarez-Santos, C. Regueiro, R Iglesias, S Barro, and J Presedo. "Particle filter robot localisation through robust fusion of laser, WiFi, compass, and a network of external cameras". In: *Information Fusion* 27 (2016), pp. 170–188.

[41] J. González, J.-L. Blanco, C. Galindo, A Ortiz-de Galisteo, J.-A. Fernandez-Madrigal, F. A. Moreno, and J. L. Martínez. "Mobile robot localization based on ultra-wide-band ranging: A particle filter approach". In: *Robotics and autonomous systems* 57.5 (2009), pp. 496–507.

[42] A. Canedo-Rodriguez, J. M. Rodriguez, V. Alvarez-Santos, R. Iglesias, and C. V. Regueiro. "Mobile Robot Positioning with 433-MHz Wireless Motes with Varying Transmission Powers and a Particle Filter". In: *Sensors* 15.5 (2015), pp. 10194–10220.

[43] H. M. Georges, D. Wang, Z. Xiao, and J. Chen. "Hybrid global navigation satellite systems, differential navigation satellite systems and time of arrival cooperative positioning based on iterative finite difference particle filter". In: *Communications, IET* 9.14 (2015), pp. 1699–1709.

[44] C. Boucher and J.-C. Noyer. "A hybrid particle approach for GNSS applications with partial GPS outages". In: *Instrumentation and Measurement, IEEE Transactions on* 59.3 (2010), pp. 498–505.

[45] S Sutharsan, T Kirubarajan, T. Lang, and M. McDonald. "An optimization-based parallel particle filter for multitarget tracking". In: *Aerospace and Electronic Systems, IEEE Transactions on* 48.2 (2012), pp. 1601–1618.

[46] M. S. Arulampalam, B. Ristic, N Gordon, and T Mansell. "Bearings-only tracking of manoeuvring targets using particle filters". In: *EURASIP Journal on Advances in Signal Processing* 2004.15 (2004), pp. 1–15.

[47] J.-R. Larocque, J. P. Reilly, and W. Ng. "Particle filters for tracking an unknown number of sources". In: *Signal Processing, IEEE Transactions on* 50.12 (2002), pp. 2926–2937.

[48] N. Ahmed, M. Rutten, T. Bessell, S. S. Kanhere, N. Gordon, and S. Jha. "Detection and tracking using particle-filter-based wireless sensor networks". In: *Mobile Computing, IEEE Transactions on* 9.9 (2010), pp. 1332–1345.

[49] F. Caballero, L. Merino, I. Maza, and A. Ollero. "A particle filtering method for wireless sensor network localization with an aerial robot beacon". In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 596–601.

[50] F. Zafari and I. Papapanagiotou. "Enhancing iBeacon Based Micro-Location with Particle Filtering". In: *2015 IEEE Global Communications Conference (GLOBECOM)*. 2015, pp. 1–7.

[51] Q. Wen, J. Gao, A. Kosaka, H. Iwaki, K. Luby-Phelps, and D. Mundy. "A particle filter framework using optimal importance function for protein molecules tracking". In: *Image Processing, 2005. ICIP 2005. IEEE International Conference on*. Vol. 1. IEEE. 2005, pp. I–1161.

[52] H. Cui, X. Xie, S. Xu, and Y. Hu. "Application of Particle Filter for Vertebral body Extraction: A Simulation Study". In: *Journal of Computer and Communications* 2.02 (2014), p. 48.

[53] G. Imai, H. Takahata, and M. Okada. "Particle filter assisted RFID tag location method for surgery support system". In: *Medical Information and Communication Technology (ISMICT), 2013 7th International Symposium on*. IEEE. 2013, pp. 135–138.

[54] H. F. Lopes and R. S. Tsay. "Particle filters and Bayesian inference in financial econometrics". In: *Journal of Forecasting* 30.1 (2011), pp. 168–209.

[55] D. Creal. "A survey of sequential Monte Carlo methods for economics and finance". In: *Econometric Reviews* 31.3 (2012), pp. 245–296.

[56] T. Bengtsson, P. Bickel, B. Li, et al. "Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems". In: *Probability and statistics: Essays in honor of David A. Freedman*. Institute of Mathematical Statistics, 2008, pp. 316–334.

[57] J. A. Castellanos, J. Montiel, J. Neira, and J. D. Tardós. "The SPmap: A probabilistic framework for simultaneous localization and map building". In: *Robotics and Automation, IEEE Transactions on* 15.5 (1999), pp. 948–952.

[58] S. Thrun. "A probabilistic on-line mapping algorithm for teams of mobile robots". In: *The International Journal of Robotics Research* 20.5 (2001), pp. 335–363.

[59] J. J. Leonard and H. J. S. Feder. "A computationally efficient method for large-scale concurrent mapping and localization". In: *ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-*. Vol. 9. Citeseer. 2000, pp. 169–178.

[60] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *Robotics & Automation Magazine, IEEE* 13.2 (2006), pp. 99–110.

[61] T. Bailey and H. Durrant-Whyte. "Simultaneous localization and mapping (SLAM): Part II". In: *IEEE Robotics & Automation Magazine* 13.3 (2006), pp. 108–117.

[62] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. "Monte carlo localization for mobile robots". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[63] H. P. Moravec. "Sensor fusion in certainty grids for mobile robots". In: *AI magazine* 9.2 (1988), p. 61.

[64] S. Thrun et al. "Robotic mapping: A survey". In: *Exploring artificial intelligence in the new millennium* 1 (2002), pp. 1–35.

[65] S. Williams, G. Dissanayake, and H. Durrant-Whyte. "Towards terrain-aided navigation for underwater robotics". In: *Advanced Robotics* 15.5 (2001), pp. 533–549.

[66] D. Ribas, P. Ridao, and J. Neira. *Underwater SLAM for structured environments using an imaging sonar*. Vol. 65. Springer, 2010.

[67] S Scheding, E. Nebot, M Stevens, H Durrant-Whyte, J Roberts, P Corke, J Cunningham, and B Cook. "Experiments in autonomous underground guidance". In: *Ultrasonics* 10 (1997), p. 4.

[68] S. Thrun, D. Hahnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. "A system for volumetric robotic mapping of abandoned mines". In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 3. IEEE. 2003, pp. 4270–4275.

[69] J. K. Uhlmann, S. J. Julier, B. Kamgar-Parsi, M. O. Lanzagorta, and H.-J. S. Shyu. "NASA Mars rover: a testbed for evaluating applications of covariance intersection". In: *AeroSense'99*. International Society for Optics and Photonics. 1999, pp. 140–149.

[70] R. Li, F Ma, F Xu, L Matthies, C Olson, and Y Xiong. "Large scale mars mapping and rover localization using escent and rover imagery". In: *International Archives of Photogrammetry and Remote Sensing* 33.B4/2; PART 4 (2000), pp. 579–586.

[71] C. H. Tong. "Laser-based 3D mapping and navigation in planetary worksite environments". PhD thesis. University of Toronto, 2013.

[72] A. Ellery. *Planetary Rovers: Robotic Exploration of the Solar System*. Springer, 2015.

[73] D. Holz and S. Behnke. "Mapping with micro aerial vehicles by registration of sparse 3D laser scans". In: *Intelligent Autonomous Systems 13*. Springer, 2016, pp. 1583–1599.

[74] S. Weiss, D. Scaramuzza, and R. Siegwart. "Monocular-SLAM–based navigation for autonomous micro helicopters in GPS-denied environments". In: *Journal of Field Robotics* 28.6 (2011), pp. 854–874.

[75] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. "Visual odometry and mapping for autonomous flight using an RGB-D camera". In: *International Symposium on Robotics Research (ISRR)*. Vol. 2. 2011.

[76] M Bosse, J Leonard, and S Teller. "Large-scale CML using a network of multiple local maps". In: *Workshop Notes of the ICRA Workshop on Concurrent Mapping and Localization for Autonomous Mobile Robots (W4), Washington, DC*. 2002.

[77] S. Thrun, W. Burgard, and D. Fox. "A probabilistic approach to concurrent mapping and localization for mobile robots". In: *Autonomous Robots* 5.3-4 (1998), pp. 253–271.

[78] T. Bailey. "Mobile robot localisation and mapping in extensive outdoor environments". PhD thesis. Citeseer, 2002.

[79] P. Jensfelt, S. Ekvall, D. Kragic, and D. Aarno. "Augmenting slam with object detection in a service robot framework". In: *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*. IEEE. 2006, pp. 741–746.

[80] S. Pillai and J. Leonard. "Monocular SLAM Supported Object Recognition". In: *arXiv preprint arXiv:1506.01732* (2015).

[81] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. "Robust Monte Carlo localization for mobile robots". In: *Artificial intelligence* 128.1 (2001), pp. 99–141.

[82] I. M. Rekleitis. "A particle filter tutorial for mobile robot localization". In: *Centre for Intelligent Machines, McGill University, Tech. Rep. TR-CIM-04-02* (2004).

[83] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. "Monte carlo localization for mobile robots". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[84] B. Schiele and J. L. Crowley. "A comparison of position estimation techniques using occupancy grids". In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE. 1994, pp. 1628–1634.

[85] M. Pinto, A. P. Moreira, and A. Matos. "Localization of mobile robots using an extended Kalman filter in a LEGO NXT". In: *Education, IEEE Transactions on* 55.1 (2012), pp. 135–144.

[86] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[87] D. Fox, W. Burgard, and S. Thrun. "Markov localization for mobile robots in dynamic environments". In: *Journal of Artificial Intelligence Research* (1999), pp. 391–427.

[88] A. I. Eliazar and R. Parr. "DP-SLAM 2.0". In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 2. IEEE. 2004, pp. 1314–1320.

[89] L. F. J. Bohrenstein, H. Everett, and L. Feng. "Navigating mobile Robots". In: *AK Peters* (1996).

[90] A. R. Siddiqui. "On Fundamental Elements of Visual Navigation Systems". In: (2014).

[91] D. Filliat and J.-A. Meyer. "Map-based navigation in mobile robots:: I. a review of localization strategies". In: *Cognitive Systems Research* 4.4 (2003), pp. 243–282.

[92] P. G. Zavlangas and S. G. Tzafestas. "Integration of topological and metric maps for indoor mobile robot path planning and navigation". In: *Methods and applications of artificial intelligence*. Springer, 2002, pp. 121–130.

[93] S. Thrun. "Learning metric-topological maps for indoor mobile robot navigation". In: *Artificial Intelligence* 99.1 (1998), pp. 21–71.

[94] C. Stachniss. “Spatial Modeling and Robot Navigation”. PhD thesis. Universitat Freiburg, 2009.

[95] K. O. Arras, J. A. Castellanos, M. Schilt, and R. Siegwart. “Feature-based multi-hypothesis localization and tracking using geometric constraints”. In: *Robotics and Autonomous Systems* 44.1 (2003), pp. 41–53.

[96] A. Corominas Murtra and J. M. Mirats Tur. “Map format for mobile robot map-based autonomous navigation”. In: (2007).

[97] A. Elfes. “Occupancy grids: a probabilistic framework for robot perception and navigation”. In: (1989).

[98] A. Milstein. *Occupancy grid maps for localization and mapping*. INTECH Open Access Publisher, 2008.

[99] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* (2013). Software available at `http://octomap.github.com`. URL: `http://octomap.github.com`.

[100] I. Dryanovski, W. Morris, and J. Xiao. “Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 1553–1559.

[101] M. Yguel, C. Tay, M. Keat, C. Braillon, C. Laugier, and O. Aycard. “Dense mapping for range sensors: Efficient algorithms and sparse representations”. In: (2007).

[102] A. Doucet, N. De Freitas, K. Murphy, and S. Russell. “Rao-Blackwellised particle filtering for dynamic Bayesian networks”. In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2000, pp. 176–183.

[103] J. E. Bresenham. “Algorithm for computer control of a digital plotter”. In: *IBM Systems journal* 4.1 (1965), pp. 25–30.

[104] D Roller, M. Montemerlo, S Thrun, and B. Wegbreit. “Fastslam 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges”. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. 2003.

[105] D. H. D. Fox, W Burgard, and S Thrun. “A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements”. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2003.

[106] G. Grisetti, C. Stachniss, and W. Burgard. “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 2432–2437.

[107] C. Stachniss, D. Hähnel, and W. Burgard. "Exploration with active loop-closing for FastSLAM". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on.* Vol. 2. IEEE. 2004, pp. 1505–1510.

3

PFs Complexity Analysis and HW Acceleration Methods

3.1 Introduction

The huge computational complexities involved with PFs is known to be one of the major constraint to their widespread use for different real time applications. To improve the computational performance of PFs and achieve real-time computations, the knowledge of their computational complexity is of paramount importance.

This chapter analyzes the computational complexities of the sampling, importance weight and resampling steps for the two most popular types of PFs, generic PF (SIR) and regularized PF (RPF). In the resampling step, the different types of resampling methods explained in Chapter 2 are considered. In addition, the evaluation on the performance of the PFs is considered by using the most common root mean squared error (RMSE) metrics. The results obtained are of importance in the study of accelerating the PF algorithm in a hardware based platform and to be applied in real time problems. Therefore, based on the analysis of the computational complexities, hardware acceleration techniques are proposed for the design and development of PF hardware acceleration architectures in the next chapters.

3.2 Comparison on PFs and Resampling Methods

The investigations on the computational complexities and performance of the SIR PF, RPF and the different types of resampling methods are based on their implementation on a computer with an Intel Core 2 Duo CPU 3.00 GHz and 4 Gb RAM, running Windows XP service Pack 3. The computational complexity study is conducted using MATLAB® software's Profiler functionality which is used to debug and optimize MATLAB® code files by tracking their execution time.

The computational complexity study is conducted by measuring the time required to perform one step of the iteration by varying the number of particles, N. The performance of the algorithms is studied using the RMSE metrics evaluated for 100 run steps. These investigations are performed based on the specific application of the PF adopted in this study, namely the Grid-based fast-SLAM algorithm. Real time odometry and laser scanner data collected from a mobile robot platform is used for post processing in the Grid-based fast-SLAM algorithm. The estimation is performed for the 2D positions and orientation of the robot, given by the Euclidian distance (range) and orientation (bearing) of the robot.

Considering that the sampling and importance weight steps are identical for both SIR and RPF, Table 3.1 provides the results of the execution time for these two steps of the PF applied to the Grid-based fast-SLAM algorithm. The results of Table 3.1 shows that the importance weight computation requires larger amount of computational time in comparison to the sampling step. This is attributed to the requirement of nonlinear function evaluations and map updating procedures

Table 3.1: Execution Time for Sampling and Importance Weight steps

N	**Sampling**	**Importance weight**
	Execution time (s)	
20	0.00224	0.330
50	0.00554	0.818
80	0.00902	1.340
110	0.01220	1.810

in the importance weight step. As expected theoretically, the execution times for both steps increase with the number of particles, N.

Table 3.2 provides the execution times and performance results of the different resampling methods with the SIR PF and RPF. For clarity the execution time results of the SIR and RPF with the different resampling methods is summarized in Fig. 3.1. The results in Fig. 3.1 show that the IMHA resampling algorithm has the least computational time followed by the systematic, stratified, residual and multinomial resampling algorithms. Except the IMHA, the other resampling methods differ only the way they generate the random numbers. Due to the complex operation in random number generation and the need to generate N independent random numbers in multinomial resampling compared to the generation of only one random number in the case of systematic and stratified resampling methods, resulted in its higher computational complexity. Regarding the computational complexity between the RPF and generic PF, RPF has a relatively higher execution time which can be accounted to the extra regularization step where most of it is accounted to the empirical covariance matrix calculation. The study on the comparison of sample impoverishment among the different resampling methods is shown in Fig. 3.2. Fig. 3.3 compares the percentage of computations for the sampling, importance weight and resampling steps for the SIR PF considering the IMHA resampling. Fig. 3.3 confirms the higher percentage in computation of the importance weight step in relative to the sampling and resampling steps.

3.3 Computational Bottlenecks Identifications and HW/SW Partitioning

The analysis and experimental results of Section 3.2 show that the SIR PF with the IMHA resampling has the lowest computational complexity compared to the RPF. As a result, the SIR PF with the IMHA resampling is considered in this study. Compared to the other resampling methods IMHA provides a bottleneck free operation as the resampling operation can be pipelined with the sampling and importance weight steps.

Table 3.2: Execution Time and RMSE for SIR/RPF with Different Resampling Methods

N	Execution Time	RMSE	
		Range (m)	**Bearing (rad)**
		Multinomial	
20	2.00E-4/2.86E-4	0.0045/0.0051	0.0335/0.0263
50	9.30E-4/1.02E-3	0.0044/0.0137	0.0351/0.0329
80	2.26E-3/3.14E-3	0.0056/0.0046	0.0315/0.0310
110	4.06E-3/4.26E-3	0.0073/0.0086	0.0310/0.0299
		Average	
		0.0055/0.0080	0.0328/0.0890
		Systematic	
20	1.30E-4/2.16E-4	0.0082/0.0036	0.0296/0.0332
50	1.40E-4/2.28E-4	0.0077/0.0022	0.0328/0.0294
80	1.60E-4/2.48E-4	0.0027/0.0033	0.0372/0.0310
110	1.60E-4/2.38E-4	0.0029/0.0034	0.0342/0.0341
		Average	
		0.0054/0.0031	0.0335/0.0319
		Residual	
20	1.20E-4/2.16E-4	0.0040/0.0131	0.0400/0.0338
50	4.50E-4/5.58E-4	0.0061/0.0046	0.0338/0.0325
80	1.10E-3/1.09E-3	0.0035/0.0029	0.0360/0.0292
110	1.92E-3/1.99E-3	0.0037/0.0079	0.0382/0.0333
		Average	
		0.0043/0.0071	0.0370/0.0322
		Stratified	
20	1.30E-4/2.16E-4	0.0142/0.0141	0.0314/0.0289
50	2.50E-4/3.38E-4	0.0069/0.0092	0.0315/0.0296
80	3.60E-4/4.48E-4	0.0038/0.0061	0.0344/0.0328
110	4.70E-4/6.88E-4	0.0090/0.0052	0.0348/0.0342
		Average	
		0.0085/0.0087	0.0330/0.0314
		IMHA	
20	5.00E-5/1.36E-4	0.1001/0.1047	0.2082/0.2117
50	6.00E-5/1.48E-4	0.1097/0.1147	0.1948/0.1971
80	7.00E-5/1.58E-4	0.1021/0.1017	0.1984/0.1998
110	9.00E-5/1.68E-4	0.1118/0.1082	0.2138/0.2091
		Average	
		0.1059/ 0.1073	0.2038/0.2044

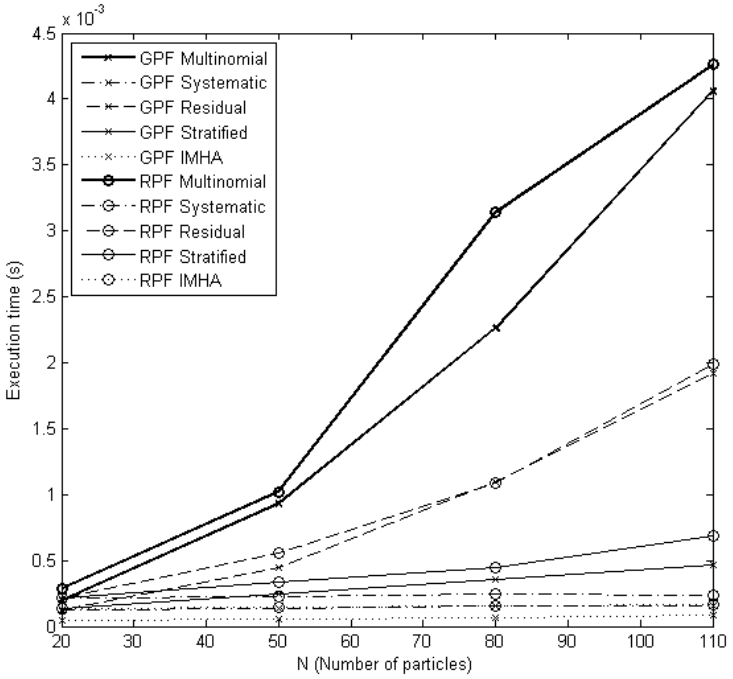


Figure 3.1: Comparison of the execution time for generic PF (GPF) and RPF with different resampling methods

A preliminary study on the identification of the critical bottlenecks of the SIR PF is crucial for the design of hardware modules in order to accelerate the computational

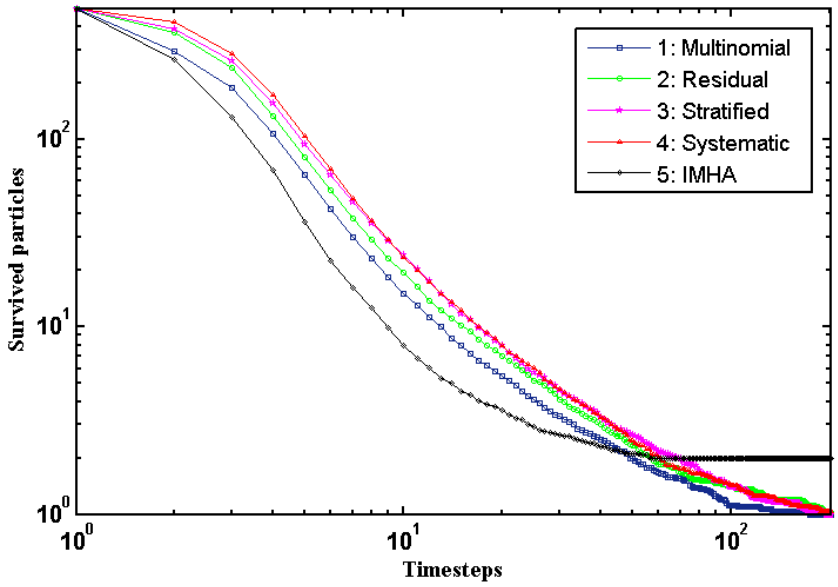


Figure 3.2: Comparison on sample impoverishment among the different resampling methods

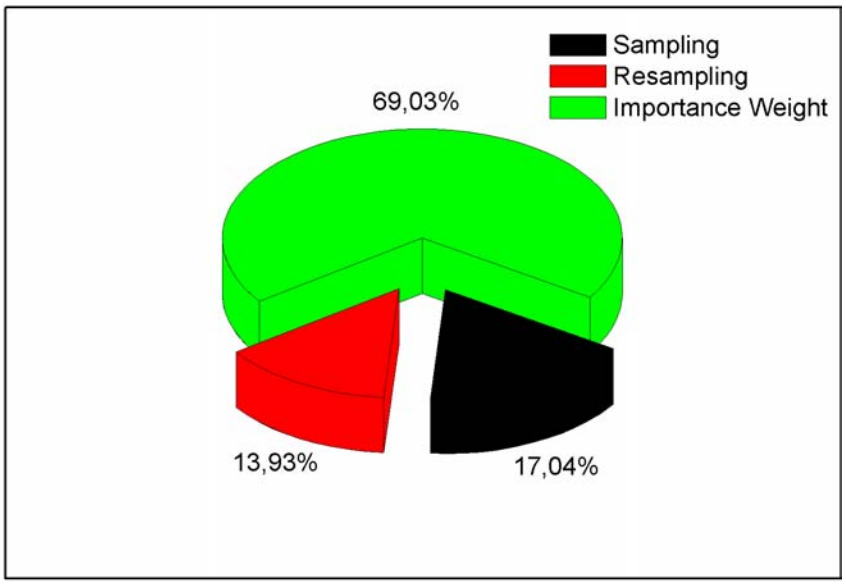


Figure 3.3: Percentage of computations for sampling, importance and resampling steps for SIR PF

bottleneck steps and consequently the overall computation. Based on an embedded implementation of the SIR PF (applied to a Grid-based Fast SLAM algorithm) on an FPGA soft-core processor (MicroBlaze), profiling is conducting on each step of the SIR PF using a hardware timer. Fig. 3.4 summarizes the critical computational bottlenecks obtained from such a study. In the sampling step, trigonometric function computations (sine, cosine and atan2) and Gaussian random number generation accounts to 45.91% and 53.62% of the sampling step execution time respectively. In the importance weight step, the computation of sine and cosine, and exponential functions contribute 75.34% and 7.65% of the execution time respectively. For the resampling step, the generation of uniform random numbers accounts for most of the execution time (60.71%). For each step of the SIR PF computations shown in Fig. 3.4, 'others' corresponds to other related computations involved in each step.

The profiling information of the PF shown in Fig. 3.4 can be used in the hardware/software partitioning i.e. which parts of the algorithm should be implemented in hardware and which ones can be kept in software (running on the FPGA's embedded processor). It is clear from Fig. 3.4 that the trigonometric functions and random number generation, are the critical bottlenecks of the Grid-based Fast SLAM algorithm, thus requiring acceleration with a hardware implementation.

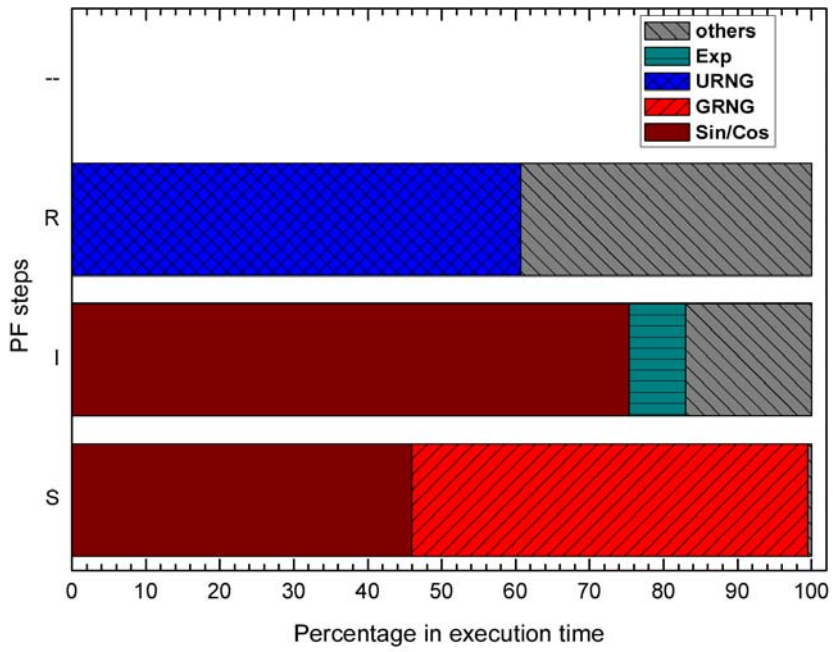


Figure 3.4: Computational bottlenecks identifications in the sampling (s), importance weight (I) and resampling (R) steps of the PF in the Grid-based Fast SLAM application.

3.4 PF Acceleration Techniques

The techniques used in the speedup of the computational bottleneck steps of the PF explained in section 3.3 are based on the use of COordinate Rotation DIgital Computer (CORDIC) algorithm [1] for the evaluations of the trigonometric and exponential functions, and the Ziggurat [2] and Tausworthe [3] algorithms for generations of Gaussian and uniform random numbers respectively. The details of these techniques and the respective hardware designs are provided in this section.

3.4.1 CORDIC Acceleration Technique

CORDIC is an iterative algorithm for the calculation of the rotation of a two-dimensional vector $v = (x, y)^T$ in linear, circular and hyperbolic coordinate systems. The rotation is performed iteratively using a series of specific incremental rotation angles selected so that each iteration is performed by shift and add operation. The resulting trajectories for the vector v_i from the successive CORDIC iterations in the linear, hyperbolic and circular coordinate systems is shown in Fig. 3.5. The norm of the vector $(x, y)^T$ in these coordinate systems is defined as $\sqrt{x^2 + my^2}$,

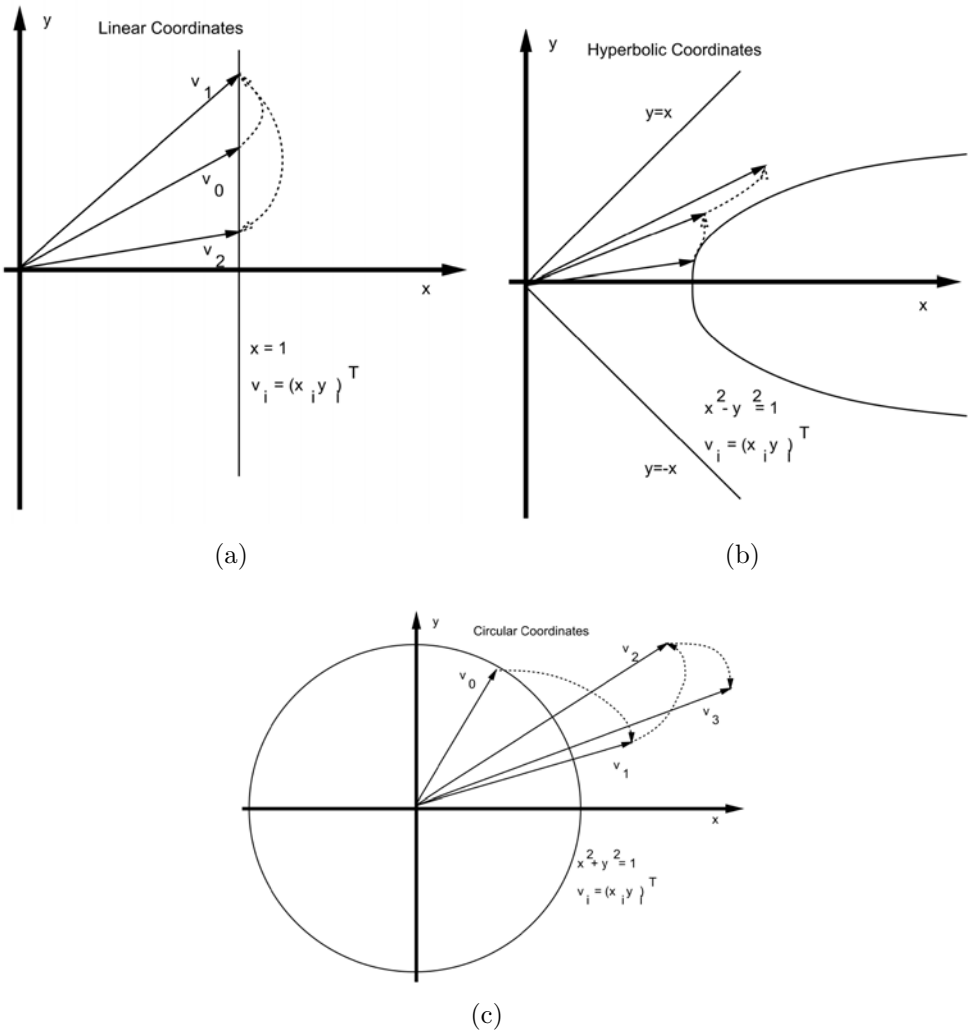


Figure 3.5: CORDIC rotation trajectories for the linear (a), hyperbolic (b) and circular (c) coordinate systems[4]

where $m \in \{1, 0, -1\}$ is a coordinate parameter specifying whether the rotation is in a circular, linear or hyperbolic coordinate respectively. The norm preserving rotation trajectory is a circle defined by $x^2 + y^2 = 1$ in the circular coordinate system, while in the hyperbolic coordinate system the rotation trajectory is a hyperbolic function defined by $x^2 + y^2 = -1$ and in the linear coordinate system the trajectory is a simple line $x = 1$.

The unified CORDIC algorithm for the computation of the rotation of the two-dimensional vector $v = (x, y)^T$ is given by the following iterative equations [1]:

$$\begin{aligned} x_{n+1} &= x_n - m\mu_n y_i \delta_{m,n} \\ y_{n+1} &= y_n + \mu_n x_i \delta_{m,n} \\ z_{n+1} &= z_n - \mu_n \alpha_{m,n} \end{aligned} \tag{3.1}$$

where $\alpha_{m,n}$ defines the angle of the vector $v_n = (x_n, y_n)^T$, the variable z_n is used to keep track of the rotation angle $\alpha_{m,n}$, $\mu_n \in \{1, -1\}$ represents either clockwise or counter clockwise direction of rotation. The variable $\delta_{m,n}$ is defined as $\delta_{m,n} = 2^{-s_{m,n}}$, where $s_{m,n}$ defines a non decreasing integer shift sequence.

The first two equations in the unified CORDIC equation 3.1 can be represented as $v_{n+1} = R_{m,n} v_n$, where $R_{m,n}$ is a rotation matrix given by equation 3.2. For $m \in \{-1, 1\}$, $R_{m,n}$ is an unnormalized rotation matrix by scale factor of $K_{m,n} = \sqrt{1 + tan^2(\sqrt{m}\alpha_{m,n})}$ describing a rotation with scaling rather than a pure rotation. However, for $m = 0$, $R_{m,n}$ is a normalized rotation matrix with no scaling is involved.

$$\begin{aligned} R_{m,n} &= K_{m,n} \begin{pmatrix} cos(\sqrt{m}\alpha_{m,n}) & -\mu_n \sqrt{m} sin(\sqrt{m}\alpha_{m,n}) \\ \frac{\mu_n}{\sqrt{m}} \sin(\sqrt{m}\alpha_{m,n}) & cos(\sqrt{m}\alpha_{m,n} \end{pmatrix} \\ &= K_{m,n} R^*_{m,n} \end{aligned} \tag{3.2}$$

After k successive iterations the resulting vector v_k is given by equation 3.3, which is a rotation by an angle $\theta = \sum_{n=0}^{k-1} \mu_n \alpha_{m,n}$ is performed with an overall scaling factor of $K_m(k) = \prod_{n=0}^{k-1} K_{m,n}$.

$$v_k = \prod_{n=0}^{k-1} K_{m,n} \prod_{n=0}^{k-1} R^*_{m,n} v_0 \tag{3.3}$$

The third iteration component of the unified CORDIC equation (i.e. $z_{n+1} = z_n - \mu_n \alpha_{m,n}$) simply keeps track of the overall rotation angle accumulated during successive pseudo-rotations. After k iterations it results in z_k given by equation 3.4, which is the difference of the start value z_0 and the total accumulated rotation angle.

$$z_k = z_0 - \sum_{n=0}^{k-1} \mu_n \alpha_{m,n} \tag{3.4}$$

In rotation mode it performs a general rotation by a given angle z and in vectoring mode it computes unknown angle z of a vector by performing a finite number of micro rotations.

CORDIC Modes of Operation

The CORDIC algorithm is formulated by using a shift sequence $s_{m,n}$ defining an angle sequence $\alpha_{m,n} = \frac{1}{\sqrt{m}} tan^{-1}(\sqrt{m}2^{-s_{m,n}})$ and a control scheme generating a sign sequence μ_n which guarantees convergence. The shift sequences which guarantees convergence for the linear ($m = 0$), circular ($m = 1$) and hyperbolic ($m = -1$) coordinates are given by equation 3.5. For $m = -1$ convergence is not satisfied for $\alpha_{-1,i} = tanh^{-1}(2^{-i})$, but it is satisfied if the integers $(4, 13, 40, ..., n, 3n + 1, ...)$ are repeated in the shift sequence [5, 6]. For any implementation, the angle sequence $\alpha_{m,n}$ resulting from the chosen shift sequence can be calculated in advance, quantized according to a chosen quantization scheme and retrieved from storage during execution of the CORDIC algorithm.

$$s_{m,n} = \begin{cases} 0, 1, 2, 3, 4, ..., n, ... & m = 0 \\ 1, 2, 3, 4, 5..., n + 1, ... & m = 1 \\ 1, 2, 3, 4, 4, 5... & m = -1 \end{cases} \tag{3.5}$$

In CORDIC algorithm the direction of the rotation angle has to be chosen such that the absolute value of the remaining rotation angle, $|\beta_{n+1}| = ||\beta_n| - \alpha_{m,n}|$, after rotation n eventually becomes smaller during successive iterations. Two control schemes, namely *rotation mode* and *vectoring mode* are used for this purpose.

In rotation mode the desired rotation angle θ is given for an input vector $(x, y)^T$. Setting $x_0 = x$, $y_0 = y$, and $z_0 = \theta$, after k iterations results in:

$$z_k = \theta - \sum_{n=0}^{k-1} \mu_n \alpha_{m,n} \tag{3.6}$$

If z_k holds, then $\theta = \sum_{n=0}^{k-1} \mu_n \alpha_{m,n}$, i.e. the total accumulated rotation angle is equal to θ. In order to drive z_k to zero, $\mu_n = sign(z_n)$ is used leading to

$$\begin{aligned} x_{n+1} &= x_n - m\ \text{sign}(z_n)\ y_n 2^{-s_{m,n}} \\ y_{n+1} &= y_n + \text{sign}(z_n)\ x_n 2^{-s_{m,n}} \\ z_{n+1} &= z_n - \text{sign}(z_n)\ \alpha_{m,n} \end{aligned} \tag{3.7}$$

The finally computed scaled rotated vector is given by $(x^k, y^k)^T$.

In vectoring mode the objective is to rotate the given input vector $(x, y)^T$ with magnitude $\sqrt{x^2 + my^2}$ and angle $\phi = \frac{1}{\sqrt{m}} tan^{-1}(\sqrt{m}\frac{y}{x})$ towards the x-axis. The initial values are set as $x_0 = x$, $y_0 = y$ and $z_0 = 0$. The control scheme is such that during the k iterations y_k is driven to zero and $\mu_n = \text{sign}(x_n)\ \text{sign}(y_n)$. Depending on the sign of x_0 the vector is then rotated towards the positive ($x \geq 0$) or negative ($x_0 < 0$) x-axis. If $y_k = 0$ holds, z_k contains the negative total accumulated rotation angle after k iterations which is equal to ϕ and x_k contains the scaled and eventually (for $x_0 < 0$) signed magnitude of the input vector.

$$z_n = \phi = -\sum_{n=0}^{k-1} \mu_n \alpha_{m,n} \tag{3.8}$$

The CORDIC iterations driving the y_n variables to zero is given by:

$$\begin{aligned} x_{n+1} &= x_n + m\ \text{sign}(x_n)\ \text{sign}(y_n)\ y^n 2^{-s_{m,n}} \\ y_{n+1} &= y_n - \text{sign}(x_n)\ \text{sign}(y_n)\ x_n 2^{-s_{m,n}} \\ z_{n+1} &= z_n + \text{sign}(x_n)\ \text{sign}(y_n)\alpha_{m,n} \end{aligned} \tag{3.9}$$

Analysis of CORDIC Functions for PFs

Using the unified CORDIC iteration equations for $m \in (1, 0, -1)$ configurations a large class of mathematical functions can be computed in hardware using rotation and vectoring modes. In linear configuration ($m = 0$), the evaluation of a multiplication function can be performed. However, the computation of a multiplication with the linear configuration requires a number of clock cycles due to the sequential nature of the CORDIC algorithm. Alternatively, multiplication can be efficiently performed with a single clock using standard MAC (Multiply-Accumulate) unit. Therefore, the multiplication realized with the linear CORDIC

mode ($m = 0$) can not compete to the standard MAC unit. In contrast all functions calculated in the circular and hyperbolic modes compare favorably to the respective implementations on DSPs [4, 7]. Therefore, a CORDIC processing element extension for $m \in \{1, -1\}$ to standard DSPs is the most attractive possibility.

The configurations to the variables x, y and z, and the set of equations that the unified CORDIC iteration equations converges after k iterations in rotation and vectoring modes is given in Table 3.3. Where, in circular-rotation mode of configuration the functions f_1 and f_2 correspond to the $sine()$ and $cosine()$ functions respectively and for hyperbolic-rotation configuration f_1 and f_2 correspond $sinh()$ and $cosh()$ functions. In vectoring-hyperbolic configuration the function f_3 corresponds to $tanh^{-1}$ and in circular-hyperbolic configuration it corresponds to tan^{-1}.

The evaluation of the sine and cosine functions are obtained directly by applying the properties given in Table 3.3. However for the evaluation of an exponential and natural logarithm functions indirect properties are used. The evaluation of the exponential function is obtained indirectly by applying the property:

Table 3.3: CORDIC configurations for functions evaluation. $K \sim K_m(n \to \infty)$

Mode	**Configuration**	
	Circular	**Hyperbolic**
Rotation	Functions	
	Sine/Cosine	Exponential
$x = K(xf_1z - yf_2z)$	$x = 1/K$	
$y = K(yf_1z - xf_2z)$	$y = 0$	
$z = 0$	$z =$ input	
	$K \sim 1.646$	$K \sim 0.828159$
Vectoring	Functions	
	tan^{-1}	Logarithmic
$x = K\sqrt{x^2 - y^2}$	$x = 1$	
$y = 0$	$y =$ input	
$z = z + f_3\left(\frac{y}{x}\right)$	$z = 0$	

$$exp(z) = sinh(z) + cosh(z) \tag{3.10}$$

Similarly, in the case of natural logarithm function the property given by equation (14) is used for its indirect evaluation.

$$lnw = 2tanh^{-1}(\frac{y}{x}) \tag{3.11}$$

Where, $x = w + 1$ and $y = w - 1$.

As the CORDIC algorithm works for a limited range of the input arguments for the evaluation of the elementary functions, it is required to extend the range of the inputs for each mode of operation by applying proper *pre-scaling* identities. This is achieved by dividing the original input arguments to the CORDIC algorithm by a constant to obtain a quotient Q and remainder D [8]. In the case of the sine and cosine functions the constant value corresponds to $(\frac{\pi}{2})$, and $log_e 2$ for the exponential and logarithmic functions. The pre-scaling identities for all the required functions are given in Table 3.4.

Table 3.4: Pre-scaling identities for function evaluations [8–10]

Identity	Domain
$\sin\Big(Q\frac{\pi}{2} + D\Big) = \begin{cases} \sin(D) & \text{if Q mod 4=0} \\ \cos(D) & \text{if Q mod 4=1} \\ \text{-sin}(D) & \text{if Q mod 4=2} \\ \text{-cos}(D) & \text{if Q mod 4=3} \end{cases}$	$\lvert D\rvert < \frac{\pi}{2}$
$\cos\Big(Q\frac{\pi}{2} + D\Big) = \begin{cases} \cos(D) & \text{if Q mod 4=0} \\ \text{-sin}(D) & \text{if Q mod 4=1} \\ \text{-cos}(D) & \text{if Q mod 4=2} \\ \sin(D) & \text{if Q mod 4=3} \end{cases}$	$\lvert D\rvert < \frac{\pi}{2}$
$exp(Qlog_e 2 + D) = 2^Q\Big(cosh(D) + sinh(D)\Big)$	$\lvert D\rvert < log_e 2$
$log_e(M2^M) = log_e(M) + Elog_e 2$	$0.5 \leq M < 1$

The accuracy of the functions evaluated with the unified CORDIC iteration equations are affected with two primary sources of errors [10]. The first sources of error is an angle approximation errors. Theoretically, the rotation angle is approximated by decomposing into infinite number of elementary angles. However, for practical implementation a finite number of k micro-rotations are considered in order to approximate the input rotation angle which results in an approximation error of $\phi \leq \alpha_{m,k-1}$. The second source of error is a finite precision error that results due to the fixed point representations of the different variables in the fixed point realization of the CORDIC algorithm. Additionally the rounding error, which increases with the number of iterations, has to be taken into account.

3.4.2 CORDIC Harware Architecture

Based upon the hardware realization of the three iterative equations, CORDIC architectures can be broadly classified as folded and unfolded [10, 11].

Folded Architecture

The direct mapping of the iterative operations of the unified CORDIC algorithm, equation 3.1, into hardware results in the folded architecture [12, 13] shown in Fig. 3.6. In the folded architecture a single adder/subtractor unit, a shifter and a register for holding the computed values after each iteration are used. As single shift-add/sub stage is used, the folded architecture have to be multiplexed in time domain for carrying out all the iterations with a single functional unit. The sets of elementary rotation angles $\alpha_{m,n}$ to be used can be stored in a ROM or in register files, if the number of angles is reasonably small. A control unit steers the operations in the data path according to the chosen coordinate system m and the chosen shift sequence.

The folded architecture exactly emulates the sequences of the unified CORDIC algorithm steps. For the initial level of calculation, initial values of x_0, y_0, and z_0 are provided at each branch via a multiplexer to a register block. During FPGA implementation, these initial values are given hardwired in a word wide manner. In the x and y-branches, values are first passed to a shift unit and finally subtracted or

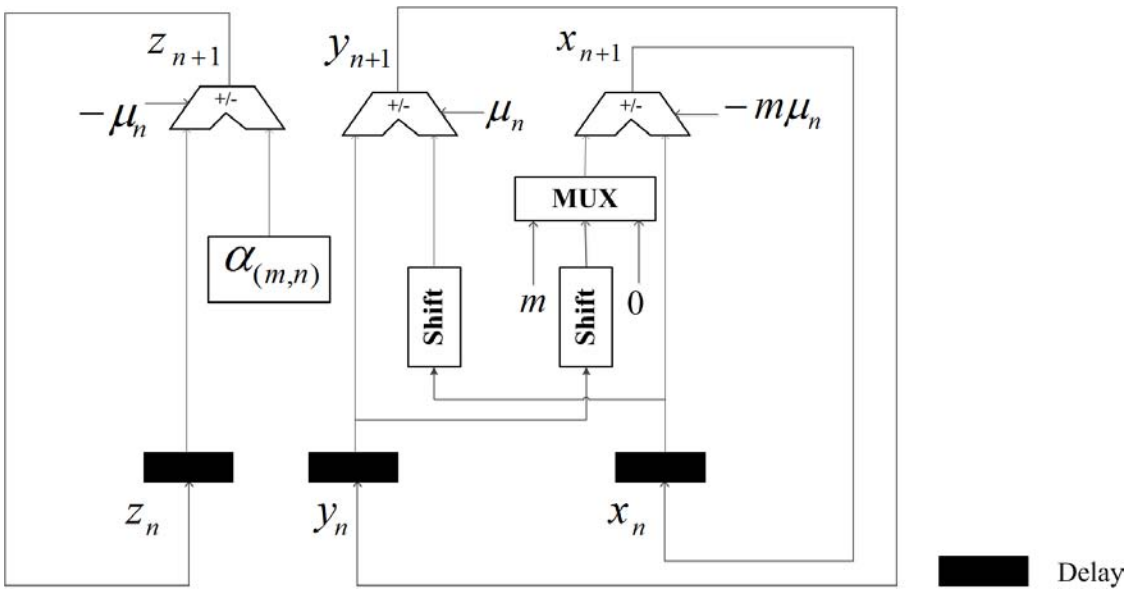


Figure 3.6: Folded (serial) CORDIC architecture.

added to un-shifted signal value from the reverse path. The operation of the parallel adder/subtractors is controlled by the rotation direction which is obtained based on the MSB of the z-branch. In the z-branch, based on the number of iterations k, a constant angle value of $\alpha_{m,n}$, $n = 0, 1, ..., k-1$, accessed from a lookup table is added or subtracted to a register value. After k operations output is passed again to register block before primary values are fed in again and this final value can be accessed as output. The addressing of the constant values $\alpha_{m,k}$ and the control of the multiplexers units can be achieved with a finite state machine design.

Due to time multiplexing to share the hardware resources in each path, the folded architecture is not suitable for high speed applications. Furthermore, because of the necessity to implement a number of different shifts according to the chosen shift sequence at each iteration, the folded architecture requires the use of variable shifters (barrel shifters). These variable shifter do not map well into FPGA architectures due to the high fan-in required and typically require several layers of the FPGA logic [14]. The result is a slow design that uses large number of logic cells. In addition the output rate is also limited by the fact that the operation is performed iteratively and therefore the maximum output rate equals $1/k$ times the clock rate. A higher clock rate and more compact design of the folded architecture with simplified interconnect

and logic can be achieved using bit serial arithmetic. Fig. 3.7 shows such a bit serial CORDIC architecture where it consists of three bit serial adder subtractor, three shift registers and a serial ROM. The number of clock cycles required for each of the n iterations in this design is equivalent to the width of the data word.

Unfolded Architecture

The unfolded architecture shown in Fig. 3.8 is acquired by unrolling (unfolding) the iteration process so that each n processing element always perform at the same iteration [15–18]. Unfolding is a transformation rule applied to a folded architecture to reorganize its operations onto a combinatoric structure [11]. In unfolded architectures no reusing of the same unit is performed during execution of the complete CORDIC algorithm as in the folded case. This results rapid increase in circuit complexity since no time multiplexing of components is exploited. On the other hand, the throughput is highly increased since it becomes always equal to the inverse of the number of clock cycles. The unfolded CORDIC architecture implement hard-wired shifters rather than area and time consuming barrel shifters. As the shifter needs not to be updated as in the iterative structure it make their FPGA implementation quite feasible.

The unfolded architectures can be realized as a parallel or pipelined architecture, where in both cases the shift-add/sub operations are implemented in parallel using

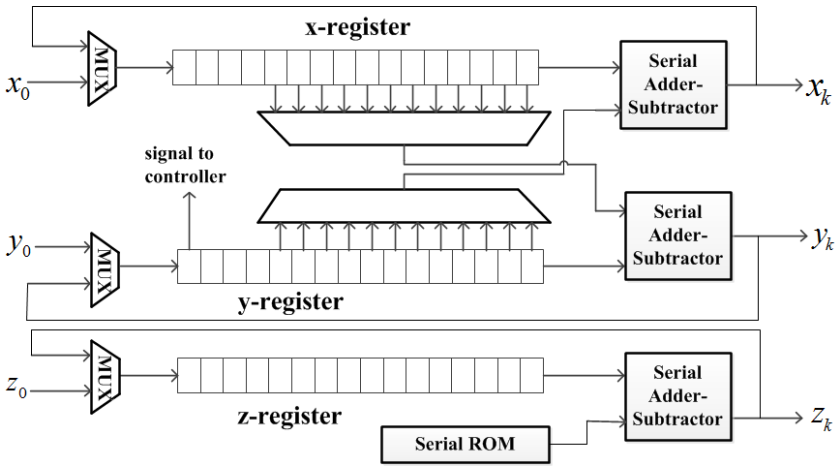


Figure 3.7: Bit-serial iterative CORDIC architecture (adopted from [14]).

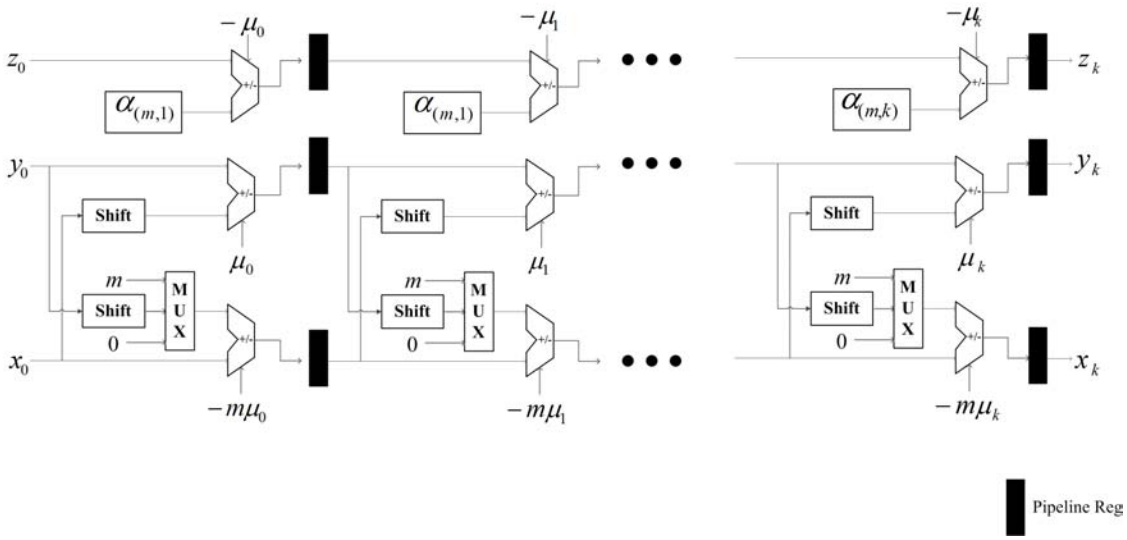


Figure 3.8: Unfolded (pipelined) CORDIC architecture.

an array of shift-add/sub stages. The difference between the parallel and pipelined realizations is the use of pipeline registers in between each iteration phase in the later case. In the parallel CORDIC architecture, instantiation of blocks must be performed k times to achieve a k bit precision output. Since all iterations are done parallelly, the outputs are available within a single clock cycle. Hence there is no need to wait for k clock cycles. Pipelined CORDIC architecture is advantageous if relatively long streams of data have to be processed in the same manner since it takes a number of clock cycles to fill the pipeline and also to flush the pipeline [4]. The first output of a k-stage pipelined CORDIC core is obtained after k clock cycles. Thereafter, outputs will be generated during every clock cycle. Compared to the folded and parallel architectures, a pipelined CORDIC architecture provides higher frequency of operation making it the best possible option for high speed applications [19–21]. A drawback of pipelined structure is the increase in area introduced by the pipeline registers.

3.4.3 CORDIC PE Architecture

The variety of functions calculated by the CORDIC algorithm leads to the idea of proposing a programmable CORDIC processing element (PE) for accelerating the intensive computations in PF applications. The architecture of the CORDIC

PE with the input/output interface is shown in Fig. 3.9. There are two 32-bit wide input ports for two possible arguments u_0 and u_1. There is a 3-bit input port, $Config$, which specifies which function is to be evaluated, as per the encoding used for the different functions. In addition to these input interfaces, a "Start" input signal is used to inform the CORDIC PE that the desired values are on the inputs and to begin calculation. There is also a clock input signal. In terms of output ports, there is a 32-bit result output port, $f()$ and a $Done$ output signal to indicate the end of a computation. The different functions that can be obtained at the output port $f()$ are: $sin()$, $cos()$, $exp()$, $atan2()$ and $ln()$ function.

The CORDIC PE is composed of three hardware computational sub-modules: CORDIC-core, Pre-scaling and Post-scaling. The implementation of these sub-modules is based on the folded or unfolded CORDIC-core architectures for the CORDIC-core sub-module, and the identity equations given in Table 3.4 for the Pre-scaling and Post-scaling sub-modules. The identity equations are used by the Pre-scaling and Post-scaling modules to extend the range of input arguments while evaluating the different functions. The CORDIC PE involves three stages of computations, first any input arguments outside the specified domain are scaled before being input to the CORDIC core at the centre of the CORDIC PE. The

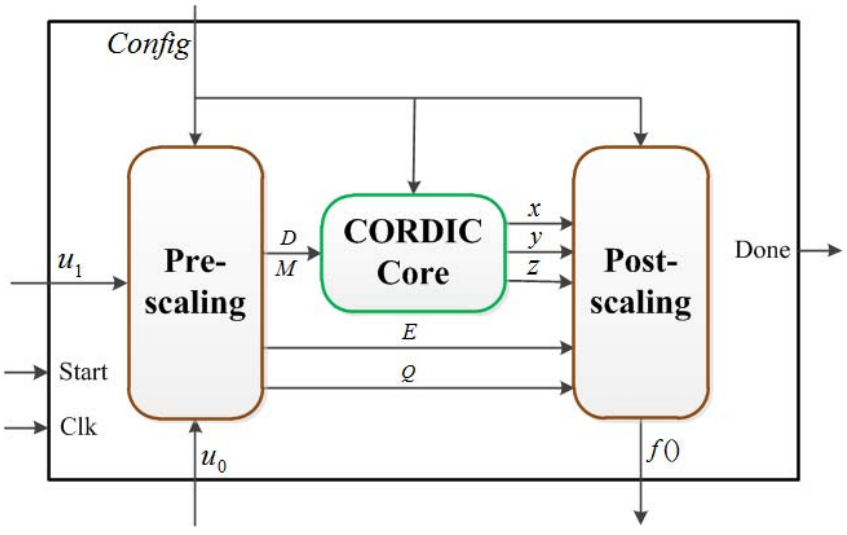


Figure 3.9: Architecture of the CORDIC PE.

results from the CORDIC core finally pass through Post-scaling module, to provide the true result.

3.5 Random Number Acceleration Technique

High quality Gaussian (normal) pseudo random number generators are key components in many scientific applications of stochastic problems and Monte Carlo simulations. Most PF application formulations, particularly for models involving Gaussian noise require the generation of Gaussian random numbers for their operation [22]. For example, as explained in the previous chapter the sampling step of the PF in the Grid-based Fast SLAM application requires the generation of Gaussian random numbers per particle filtering iteration. This section presents the technique for the generation of Gaussian random numbers in hardware with the objective of accelerating the overall PF computations.

3.5.1 Review on GRNGs

Gaussian Random Number Generators (GRNGs) aim to produce random numbers that, to the accuracy necessary for a given application, are statistically indistinguishable from samples of a random variable with an ideal Gaussian distribution. The generation of a normally distributed pseudo random number involves transforming a uniformly distributed random numbers into a normal distribution by applying appropriate transformation methods [23];- inversion, transformation, rejection sampling and recursion.

Inversion [24–27]: applies the inverse cumulative distribution function (ICDF) of the target distribution to uniformly distributed random numbers. The CDF inversion method simply inverts the CDF to produce a random number from a desired distribution. The ICDF converts a uniformly distributed random number $x \in (0, 1)$ to one output $y = ICDF(x)$ with the desired distribution. The disadvantage in such method lies in the need to calculate the CDF or its inverse.

Transformation (Box-Muller method [28–30]): In this method trigonometric functions are used to transforms a pair of uniformly distributed random numbers

into a pair of normally distributed random numbers. Its advantage is that it provides a pair of random numbers for each call deterministically. A drawback for hardware implementations is the high demand of resources needed to accurately evaluate the trigonometric functions.

Recursion (Wallace method [31]): utilizes linear combinations of previously generated Gaussian numbers to produce new outputs.

Rejection sampling (Ziggurat method [2, 32–34]): generates normal random numbers by setting conditionality i.e. it only accepts correct random values if they are within specific predefined ranges in the target distribution and discards others. The Ziggurat method described in this section is an example of such method. The Ziggurat method is one of the fastest methods to generate normally distributed random numbers while also providing excellent statistical properties [2, 35]. The key strength of the Ziggurat method is that most of the cases only two table lookups, a multiply, and a compare are needed per variate. As a result, it produces most of its outputs in a single clock cycle using fast integer operations. However, for small percentage of the outputs complex elementary functions such as exponential and natural logarithmic has to be evaluated.

3.5.2 The Ziggurat Algorithm

A Gaussian distribution with mean zero and standard deviation one, often known as a "standard normal" distribution, has the probability density function (PDF):

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{3.12}$$

The Ziggurat algorithm generates normally distributed random variates from an arbitrary decreasing PDF $\phi(x)$ by applying acceptance-rejection methods. In the acceptance-rejection methods, random variates are generated considering a set of points $C = \{(x, y)\}$ under the curve of the function $\phi(x)$ and a subset Z of these points, $Z \subset C$, by uniformly taking the random points (x, y) until $(x, y) \subset C$. The acceptance-rejection method involves conditionality in accepting correct random values that are part of the target distribution and rejecting incorrect ones.

The Ziggurat partitions the standard normal density function (given by equation 3.12) for $x > 0$, into n horizontal rectangular blocks R_i, where $i = 0, 1, 2, \ldots, (n-1)$, extending horizontally from $x = 0$ to x_i and vertically from $\phi(x_i)$ to $\phi(x_{i-1})$. The bottom block consists of a rectangular block joined with the remainder of the density starting from a point r (Fig. 3.10). All the rectangular blocks have equal area v given by equation 3.13.

$$v = x_i[\phi(x_{i-1}) - \phi(x_i)] = r\phi(r) + \int_r^{\infty} \phi(x)dx \tag{3.13}$$

The description of the Ziggurat algorithm for generating normally distributed random variates by partitioning the normal density distribution, and with the application of the acceptance-rejection method is given in Algorithm 11. In line 4 of the Ziggurat Algorithm 11, the method to generate the normal random numbers from the tail of the distribution is obtained using Algorithm12. The Ziggurat algorithm requires a table of x_i points and their corresponding function values ϕ_i. The number of rectangular blocks n is normally considered as power of two $(64, 128, 256, ..)$, and for $n = 128$ a value of $r = 3.442619855899$ is used to determine the x_i points that are required in the hardware realization of the algorithm [2].

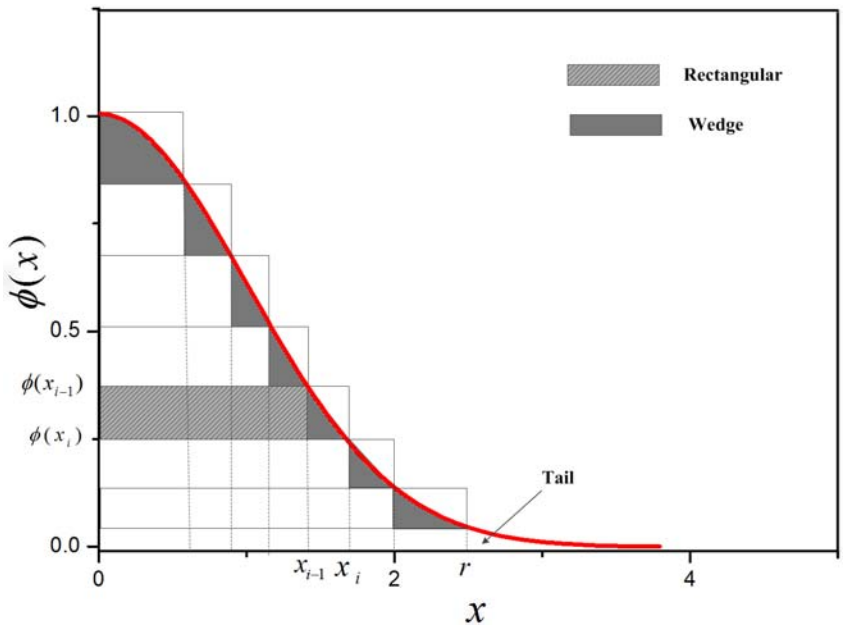


Figure 3.10: Partitioning of a Gaussian distribution into rectangular, wedge, and tail regions with Ziggurat method.

Algorithm 11 The Ziggurat Algorithm [32]

Input: $\{x_i^i\}_{i=1}^{n-1}$

1: Draw an index i of a rectangle ($0 \leq i \leq n-1$) at random with probability $\frac{1}{n}$
2: Draw a random number x from the rectangle i as $x = U_0 x_i$. where U_0 is a uniform random number drawn from a uniform distribution $U(0,1)$.
3: Rectangular
If $i \geq 1$ and $x < x_{i-1}$ accept x
4: Tail
If $i = 0$, accept x from the tail
5: Wedge
Else if $i > 0$ and $U_1|f(x_{i-1}) - f(x_i)| < f(x) - f(x_i)$ accept x, where U_1 is a uniform random number.
6: Return to step 1
7: **return** normal random number x

3.5.3 Ziggurat GRNG Hardware Architecture

In the description of a Gaussian random number generator algorithm, the existence of a uniform random number generator (URNG) that can produce random numbers with the uniform distribution over the continuous range $(0,1)$ denoted by $U(0,1)$ is assumed. In the Ziggurat algorithm, Algorithm 11, for example two uniform random numbers U_0 and U_1 are required for generating a Gaussian random number from the tail and wedge regions respectively. The generation of these two uniform random numbers is achieved with a Tausworthe URNG[3]. Although traditional linear feedback shift registers (LFSRs) are often sufficient as a URNG, the Tausworthe URNGs are fast, provide better quality and occupy less area [33]. The Tausworthe URNG follows the algorithm with its C code implementation illustrated in Fig. 3.11 as presented in [3]. It combines three LFSR-based URNGs to obtain improved statistical properties and generates a 32-bit uniform random number per clock with

Algorithm 12 Generate from the tale

1: Do
2: Generate i.i.d. uniform $(0,1)$ variates u_0 and u_1
3: $x \leftarrow \frac{-ln(u_0)}{r}, y \leftarrow -ln(u_1)$
4: While $(y+y) < x^2$
5: Return $x > 0$? $(r+x)$: $-(r+x)$

large period of 2^{88}. The hardware architecture of the Tausworthe URNG based on its description in Fig. 3.11 is shown in Fig. 3.12.

The generation of Gaussian random numbers from the tail (line 4) and wedge (line 5) regions of the Ziggurat algorithm (Algorithm 11), requires the evaluation of natural logarithmic and an exponential function respectively. The hardware evaluation of these functions is mostly performed with the application of polynomial approximation method. For example in the two most relevant discussions on hardware implementation for Ziggurat algorithm on FPGAs [33, 34] such an approach is used. These approaches lead to a tradeoff between the accuracy and resource utilizations. However, more accurate methods like CORDIC seem to be better candidate for such an application. But one of the limitations suggested in the literature for the use of CORDIC algorithm in the Ziggurat method is the high resource requirement. This is related to the need to use multiple CORDIC cores for the evaluation of each individual elementary function involved in the Ziggurat algorithm. To take advantage of the accuracy of the CORDIC algorithm, in this work a single CORDIC core that is configurable to the specific function evaluation at run-time is proposed and used. Such approach avoids the large resource requirements in the CORDIC unit for the Ziggurat algorithm. In addition, the accuracy level in the CORDIC approach can be adjusted by simply increasing and decreasing the iteration levels depending on the required level of accuracy for the specific application at hand.

```
unsigned long s1, s2, s3, b;

double taus88 ()
  {                    /* Generates numbers between 0 and 1. */
  b  = (((s1 << 13) ^ s1) >> 19);
  s1 = (((s1 & 4294967294) << 12) ^ b);
  b  = (((s2 <<  2) ^ s2) >> 25);
  s2 = (((s2 & 4294967288) <<  4) ^ b);
  b  = (((s3 <<  3) ^ s3) >> 11);
  s3 = (((s3 & 4294967280) << 17) ^ b);
  return ((s1 ^ s2 ^ s3) * 2.3283064365e-10);
  }
```

Figure 3.11: C code description of the Tausworthe URNG

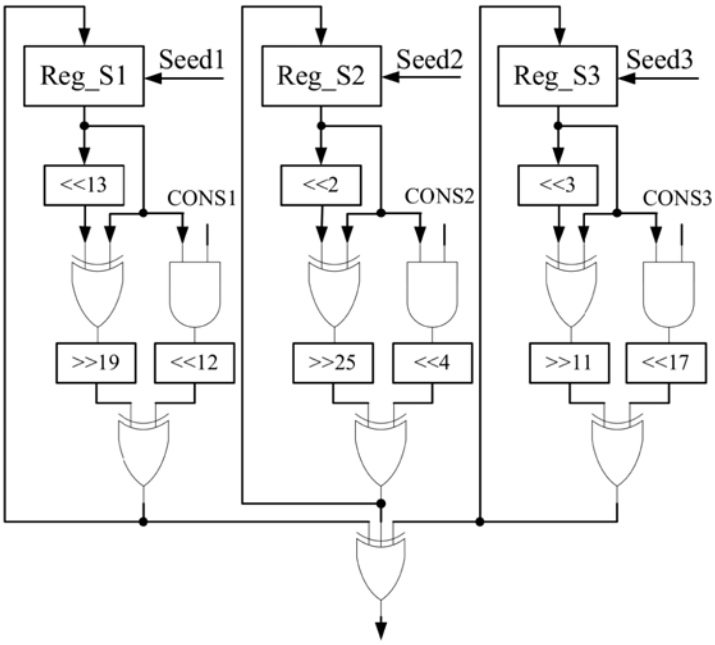


Figure 3.12: Architecture of Tausworthe URNG.

The proposed hardware architecture based on the Ziggurat algorithm for a GRNG is shown in Fig. 3.13. The three computational modules comprising the architecture are: Ziggurat GRNG, CORDIC and Tausworthe URNG modules. The Ziggurat GRNG module is the main module responsible for the generation of the normal random numbers. The Tausworthe URNG module provides two uniform random numbers U_0 and U_1 at each clock cycle required by the Ziggurat module. The CORDIC module is responsible to evaluate a specific function while requested by the Ziggurat module to generate the random variates from the tail or wedge region. The hardware design of the Ziggurat module follows the description of the Ziggurat algorithm given in Algorithm 11, where it composed of individual hardware blocks to generate the normal random number from the rectangular, wedge and tail region of the distribution (Fig. 3.10 and Algorithm 11). All the three modules (wedge, rectangular and tail) work independently of one another. The Region selector module determines the conditions for generating the normal random number from the rectangular, wedge and tail regions of the distribution. In addition, it also provides the necessary configuration bits required to configure the CORDIC module while the normal random number is generated from the wedge and tail sections.

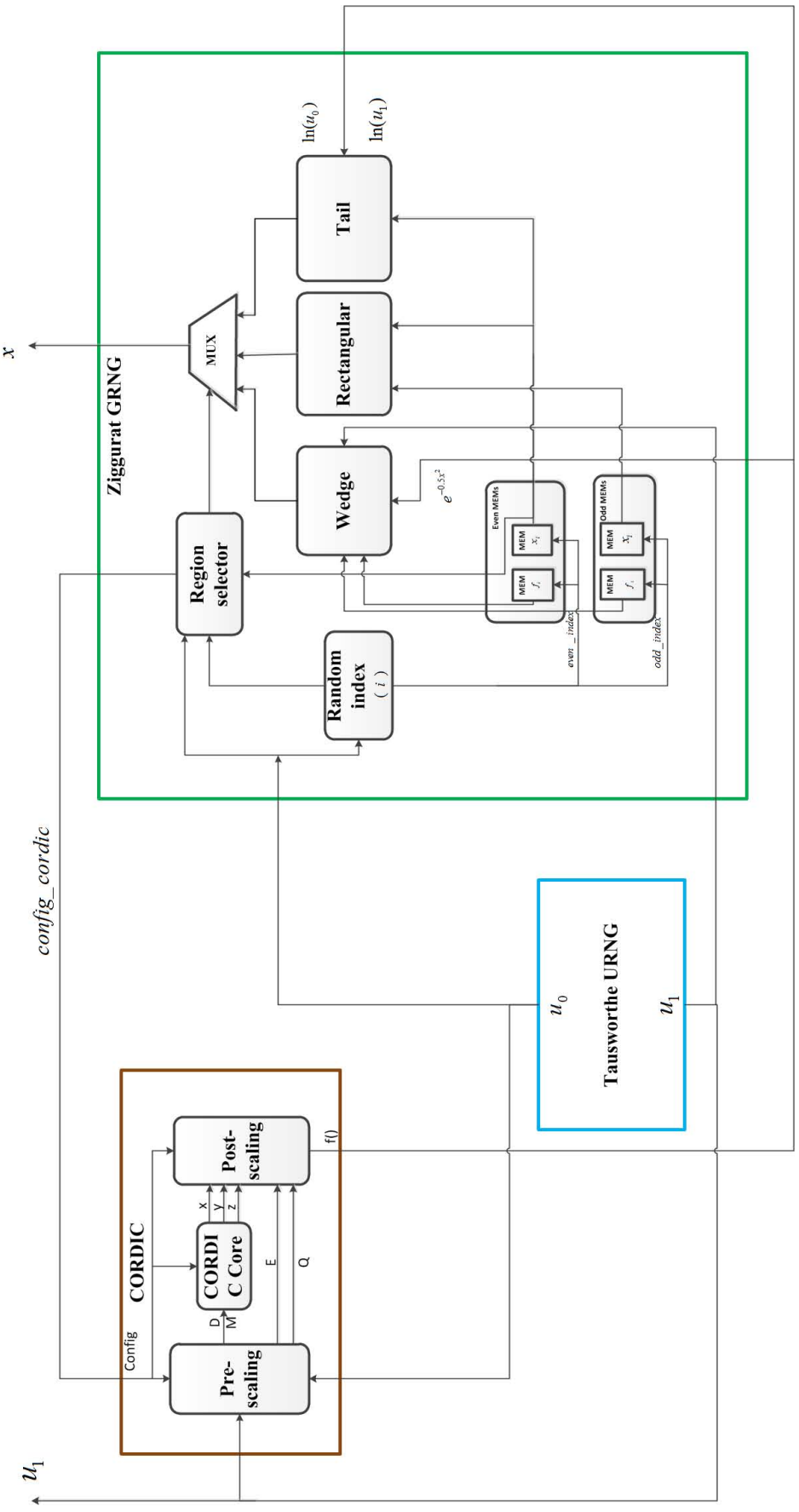


Figure 3.13: Architecture of the Ziggurat module.

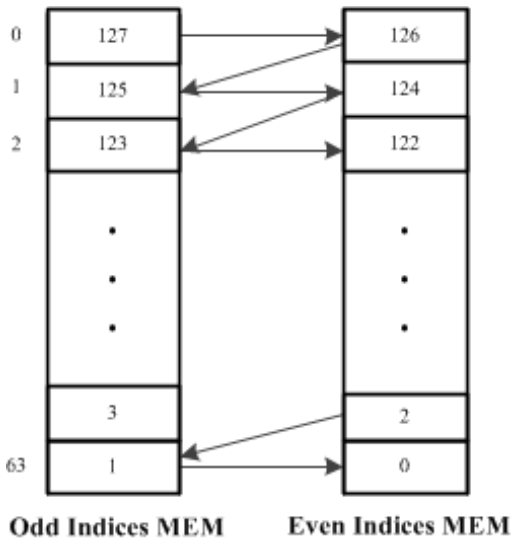


Figure 3.14: Illustration for the storage of the x_i and f_i coefficients in memory and the parallel access of the x_i ,x_{i-1} , f_i and f_{i-1} for $n = 128$.

To further speedup the normal random number generation process in the Ziggurat GRNG sub-module, an effective mechanism for the simultaneous access to the coefficients x_i and ϕ_i is required. This is achieved by dividing the random index i into even and odd values, and storing the respective x_i and ϕ_i in separate memories (Fig. 3.14). While the generated index i is an odd value the x_i and x_{i-1} are read from the odd and even memories at the same memory index positions simultaneously. However, if the generated index is with an even value, then x_i is read from the even memory and the x_{i-1} is read from the odd memory at the next memory position in parallel. As a result the parallel access of the coefficients is achieved with such an effective method.

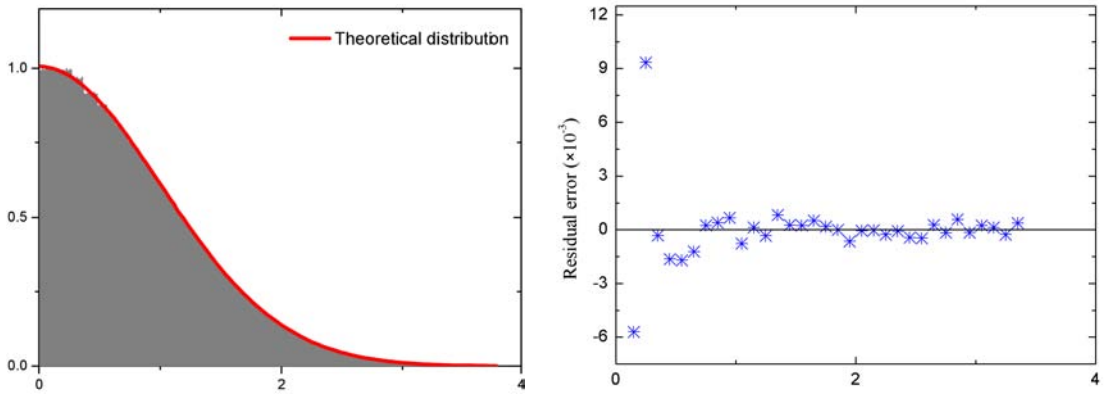
3.5.4 Ziggurat GRNG FPGA Implementation

The implementation of the architecture given in Fig. 3.13 is performed on a Xilinx Kintex-7 KC705 FPGA device. The design is written in VHDL and synthesized using the Xilinx ISE 14.6. The implementation is done for a value of $n = 128$. The x_i and ϕ_i coefficients that are stored in their respective even and odd memories are all 32 bits wide represented as fixed point number with $Q_{(8:24)}$ format. From the synthesis results, the implementation shows a maximum clock frequency of

Table 3.5: Comparison on performance and resource utilization among different FPGA implementation of the GRNGs

FPGA target devices	Kintex 7	Vertex 4	Vertex 2			
GRNGs	Ziggurat				Box-Muller	Wallace
References	Our Implementation		[34]	[33]	[28]	[31]
M Samples/sec	689.2	515.4	400	169	466	155
Maximum Frequency(MHz)	697.69	521.730	200	170	233	155
Slices	236	299	500	891	1528	770
Block RAMS	-	-	4	4	12	4
DSP Slices	4	4	24	2	3	6

$689.2MHz$ on the KC705 FPGA device. For a value of $n = 128$ the efficiency of the normal random generator is 98.78% [2]. As a result, the corresponding throughput of our implementation is $0.9878 \times 689.2 = 697.69$ samples per second which is 58.04% higher throughput from previous implementation. The results obtained are given in Table 3.5 with comparison to other hardware implementations of the GRNGs from the literature. The proposed architecture is also synthesized for the Vertex 4 FPGA device to provide fair comparison with the most recent implementation of the Ziggurat from the literature [34]. Table 3.5 shows that the proposed architecture provides a high throughput with comparable resource



(a) Hardware and theoretical distributions (b) Residual error

Figure 3.15: Histogram of hardware generated samples and theoretical Gaussian distribution (half of the distribution) (a) and corresponding residual error (b).

utilization among all the other previous hardware implementations. In addition, the proposed implementation does not utilize the Block RAM resources of the FPGA as all the required constant variables for both CORDIC and Ziggurat algorithms are implemented as ROM (lookup tables).

The histogram plot for 10^7 samples generated by the proposed hardware architecture is shown in Fig. 3.15 with the theoretical Gaussian distribution curve and the corresponding residual error (fitting deviation) values. As Fig. 3.15 shows, the hardware generated Gaussian random numbers fitted well with the theoretical curve with an average residual error of 1.311×10^{-4}. Furthermore, the quality of the Ziggurat GRNG is confirmed using the goodness of fit tests such as the Shapiro-Wilk test [36] and the coefficient of determination (R^2) test [37] ($R^2 = 0.99996$).

References

[1] J. S. Walther. "A unified algorithm for elementary functions". In: *Proceedings of the May 18-20, 1971, spring joint computer conference*. ACM. 1971, pp. 379–385.

[2] G. Marsaglia, W. W. Tsang, et al. "The ziggurat method for generating random variables". In: *Journal of statistical software* 5.8 (2000), pp. 1–7.

[3] P. L'ecuyer. "Maximally equidistributed combined Tausworthe generators". In: *Mathematics of Computation of the American Mathematical Society* 65.213 (1996), pp. 203–213.

[4] H. Dawid and H. Meyr. "CORDIC algorithms and architectures". In: *Digital Signal Process Multimedia Syst* 2 (1999), pp. 623–655.

[5] R. Parris. "Elementary Functions and Calculators". In: *Phillips Exeter Academy. Disponível em: http://math. exeter. edu/rparris/peanut/cordic. pdf* (2012).

[6] X. Hu, R. G. Harber, and S. C. Bass. "Expanding the range of convergence of the CORDIC algorithm". In: *Computers, IEEE Transactions on* 40.1 (1991), pp. 13–21.

[7] R Mehling and R Meyer. "CORDIC-AU, a suitable supplementary unit to a general-purpose signal processor". In: *AEU. Archiv für Elektronik und Übertragungstechnik* 43.6 (1989), pp. 394–397.

[8] A. Boudabous, F. Ghozzi, M. W. Kharrat, and N. Masmoudi. "Implementation of hyperbolic functions using CORDIC algorithm". In: *Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on*. IEEE. 2004, pp. 738–741.

[9] T. Vladimirova, D. Eamey, S. Keller, and M. Sweeting. "Floating-Point Mathematical Co-Processor for a Single-Chip On-Board Computer". In: *Proceedings of the 6th Military and Aerospace Applications of Programmable Logic Devices and Technologies (MAPLD'2003)* (2003).

[10] B. Lakshmi and A. Dhar. "CORDIC architectures: a survey". In: *VLSI design* 2010 (2010), p. 2.

[11] S. Wang and V. Piuri. "A unified view of CORDIC processor design". In: *Application Specific Processors*. Springer, 1997, pp. 121–160.

[12] U. Meyer-Baese and U Meyer-Baese. *Digital signal processing with field programmable gate arrays*. Vol. 65. Springer, 2007.

[13] M. D. Ercegovac and T. Lang. *Digital arithmetic*. Elsevier, 2004.

[14] R. Andraka. “A survey of CORDIC algorithms for FPGA based computers”. In: *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*. ACM. 1998, pp. 191–200.

[15] J. Bu, E. Deprettere, and F du Lange. “On the optimization of pipelined silicon CORDIC algorithm”. In: *Proc. EUSIPCO Signal Process. III: Theories Applicat* (1986), pp. 1227–1230.

[16] Y.-j. Dai and Z. Bi. “CORDIC algorithm based on FPGA”. In: *Journal of Shanghai University (English Edition)* 15 (2011), pp. 304–309.

[17] S. Vadlamani and W. Mahmoud. “Comparison of CORDIC algorithm implementations on FPGA families”. In: *System Theory, 2002. Proceedings of the Thirty-Fourth Southeastern Symposium on*. IEEE. 2002, pp. 192–196.

[18] R. Bhakthavatchalu, M. Sinith, P. Nair, and K Jismi. “A comparison of pipelined parallel and iterative CORDIC design on FPGA”. In: *Industrial and Information Systems (ICIIS), 2010 International Conference on*. IEEE. 2010, pp. 239–243.

[19] A. Mandal and R. Mishra. “FPGA Implementation of Pipelined CORDIC for Digital Demodulation in FMCW Radar”. In: *Infocommunications Journal* 5.2 (2013), pp. 17–23.

[20] A. De Lange, E. Deprettere, A van der Veen, and J Bu. “Real time application of the floating point pipeline CORDIC processor in massive-parallel pipelined DSP algorithms”. In: *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*. IEEE. 1990, pp. 1013–1016.

[21] C. S. Lee and P. R. Chang. “A maximum pipelined CORDIC architecture for inverse kinematic position computation”. In: *Robotics and Automation, IEEE Journal of* 3.5 (1987), pp. 445–458.

[22] A. Athalye. *Design and implementation of reconfigurable hardware for real-time particle filtering*. ProQuest, 2007.

[23] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor. “Gaussian random number generators”. In: *ACM Computing Surveys (CSUR)* 39.4 (2007), p. 11.

[24] N. A. Woods and T. VanCourt. “FPGA acceleration of quasi-Monte Carlo in finance”. In: *Field programmable logic and applications, 2008. FPL 2008. International Conference on*. IEEE. 2008, pp. 335–340.

[25] R. C. Cheung, D.-U. Lee, W. Luk, and J. D. Villasenor. “Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 15.8 (2007), pp. 952–962.

[26] C. De Schryver, D. Schmidt, N. Wehn, E. Korn, H. Marxen, and R. Korn. “A new hardware efficient inversion based random number generator for non-uniform distributions”. In: *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*. IEEE. 2010, pp. 190–195.

[27] C. De Schryver, D. Schmidt, N. Wehn, E. Korn, H. Marxen, A. Kostiuk, and R. Korn. "A hardware efficient random number generator for nonuniform distributions with arbitrary precision". In: *International Journal of Reconfigurable Computing* 2012 (2012), p. 12.

[28] D.-U. Lee, J. D. Villasenor, W. Luk, and P. H. Leong. "A hardware Gaussian noise generator using the Box-Muller method and its error analysis". In: *Computers, IEEE Transactions on* 55.6 (2006), pp. 659–671.

[29] J. S. Malik, A. Hemani, and N. D. Gohar. "Unifying CORDIC and Box-Muller algorithms: An accurate and efficient Gaussian Random Number generator". In: *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*. IEEE. 2013, pp. 277–280.

[30] D.-U. Lee, W. Luk, J. D. Villasenor, and P. Y. Cheung. "A Gaussian noise generator for hardware-based simulations". In: *Computers, IEEE Transactions on* 53.12 (2004), pp. 1523–1534.

[31] D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. Leong. "A hardware Gaussian noise generator using the Wallace method". In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 13.8 (2005), pp. 911–920.

[32] J. A. Doornik. "An improved ziggurat method to generate normal random samples". In: *University of Oxford* (2005).

[33] G. Zhang, P. H. Leong, D. U. Lee, J. D. Villasenor, R. C. Cheung, and W. Luk. "Ziggurat-based hardware Gaussian random number generator". In: *Field Programmable Logic and Applications, 2005. International Conference on*. IEEE. 2005, pp. 275–280.

[34] H. Edrees, B. Cheung, M. Sandora, D. B. Nummey, and D. Stefan. "Hardware-Optimized Ziggurat Algorithm for High-Speed Gaussian Random Number Generators." In: *ERSA*. 2009, pp. 254–260.

[35] P. H. Leong, G. Zhang, D.-U. Lee, W. Luk, J. Villasenor, et al. "A Comment on the Implementation of the Ziggurat method". In: *Journal of Statistical Software* 12.7 (2005), pp. 1–4.

[36] S. S. Shapiro and M. B. Wilk. "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3/4 (1965), pp. 591–611.

[37] `https://en.wikipedia.org/wiki/Coefficient_of_determination`.

4

HW/SW Approach for PF-SLAM FPGA Architecture and Implementation

4.1 Introduction

This chapter provides an FPGA system-on-chip realization of the PF-SLAM methodology explained in Chapter 2 with the objective to overcome PF low speed problems. A HW/SW co-design approach is used for the design of a HW/SW PF-SLAM system by applying the PF acceleration techniques from Chapter 3.

The organization of this chapter is as follows. Initially a brief discussion is provided on embedded system design and on the tools used to assist in the design and implementation of the PF-SLAM on FPGA platform as it is the platform used for the development of the PF system. Second, the explanation on the HW/SW PF-SLAM architecture is given. Then follows the performance evaluation of the HW/SW PF-SLAM system considering that the inputs to the system comes mainly from laser scanners. The analysis on the performance of the system is presented based on a low cost alternative laser scanner and with standard laser scanners to corroborate the good behaviour of the PF-SLAM system also with low cost laser scanners in low cost robotics applications. Finally, with an effort to further speedup the PF computations a new approach for parallel PF implementations is presented. The proposed approach helps to avoid the high communication data

flux involved with traditional parallel PF architectures through few data exchanges among parallel particle processors via a central processing unit.

4.2 Embedded Systems Implementation

4.2.1 On the Use of FPGA Based Embedded Systems

Embedded systems are computer systems that are dedicated to a specific purpose. Embedded systems composed of embedded blocks as parts of a complete device and often defined as Systems on Chip (SoC) which consist of processing cores (CPUs), Micro-controllers, DSPs and hardware specific cores tailored for dedicated tasks. The technologies for the design of embedded systems is based on software / hardware solutions. Software solutions based on a micro-programmed system, such as a CPU, general purpose graphic processing units (GPGPUs), a Micro-controller or a DSP, present the very best flexibility. However, their main drawback is that their computation is weakly intensive, especially for some embedded applications where the process requires a long computation time. Therefore, for most applications the computation time will be prohibitive if a software solution is adopted.

Hardware based solutions include FPGAs, application specific integrated circuit (ASIC) and application specific standard product (ASSP). FPGAs enable the optimization of hardware resources and the use of soft processor cores that are implemented within the FPGA logic for real-time computing. MicroBlaze [1] and Nios [2] are the most popular soft core examples provided by the two major FPGA vendors Xilinx and Altera respectively. A modern FPGA consists of a 2-D array of programmable logic blocks, memories, fixed-function blocks, and routing resources implemented in the CMOS technology. ASIC is an integrated circuit specialized to implement application specific tasks. In the ASIC the design is physically etched in the silicon. The main limitation of an ASIC is its price and the time-to-market, which limits its use to applications that requires very high performance or high volume production. The design using ASIC offers better performance, density and power consumption when compared to an FPGA. However, compared to ASIC the FPGAs have the advantages in flexibility, lower price and shorter time to

market design. ASSPs are designed and implemented in exactly the same way as ASICs. The only difference is that ASSPs are more general-purpose device that they intended for use by multiple system design houses.

FPGAs are more appealing choice for embedded system designs compared to microprocessor-based solutions, ASIC solutions, or other System-on-a-Chip (SoC) solutions. Compared to microprocessor-GPU based solutions, FPGA based solutions provide performance improvements with less power consumption. In addition, FPGAs flexibility provide the possibility of frequent updates of the digital hardware functions. The use the dynamic resources allocation feature in FPGAs also helps in order to instantiate each function for the strict required time. This permits the enhancement of silicon efficiency by reducing the reconfigurable array's area. Overall, FPGA-based embedded system is an excellent choice to perform fast processing while maintaining a good level of flexibility in allowing any modifications in the run time required for current embedded systems. The discussion on PF implementations based on FPGAs and the other platforms is provided in Section 1.2 of Chapter 1.

4.2.2 Embedded Processors in FPGAs

In the design of embedded systems based on FPGAs, there exists a hard-core and/or soft-core processor option. Hard-core processors are fixed and have dedicated resources of the FPGAs. Soft-core processors are implemented entirely in the FPGA logic and unlike the hard-core processors they must be synthesized and fit into the FPGAs fabric. Soft-core processors provide a substantial amount of flexibility through the configurable nature of the FPGA. MicroBlaze is one typical example of a soft-core processor which is used for the design of the PF-SLAM HW/SW system presented in this Chapter.

MicroBlaze is a 32-bit reduced instruction set computer (RISC) Harvard architecture soft-core processor core optimized for embedded applications in Xilinx FPGAs. The MicroBlaze soft-core processor solution provides the flexibility to incorporate a combination of peripherals, memory and interface features for the specific embedded application. The functional block diagram of the MicroBlaze

core is shown in Fig. 4.1. The architecture of Microblaze is a single -issue, 3 -stage pipeline with 32 general-purpose registers, an Arithmetic Logic Unit (ALU), a shift unit, and two levels of interrupt. In addition to this basic design, the processor is highly configurable allowing the selection of a specific set of features tailor to the exact needs of the target embedded application. These additional features include: barrel shifter, divider, multiplier, single precision floating point unit (FPU), instruction and data caches, exception handling, debug logic, Fast Simplex Link (FSL) interfaces and others. The configurability of the MicroBlaze architecture provides the flexibility in the compromization of performance requirements against the logic area cost of the soft processor for target application. Any optional features which are not required by the specific application are not implemented therefore do not utilize the FPGA resources.

The processor supports four different type memory interfaces: Local Memory Bus (LMB), IBM's Processor Local Bus (PLB) or On-chip Peripheral Local Bus (OPB), the AMBA AXI4 (Advanced eXtensible Interface 4) interface and ACE interface, and Xilinx Cache Link (XCL). The LMB provides single-cycle access to on-chip dual port block RAM. The AXI4 and PLB/OPB interfaces provide

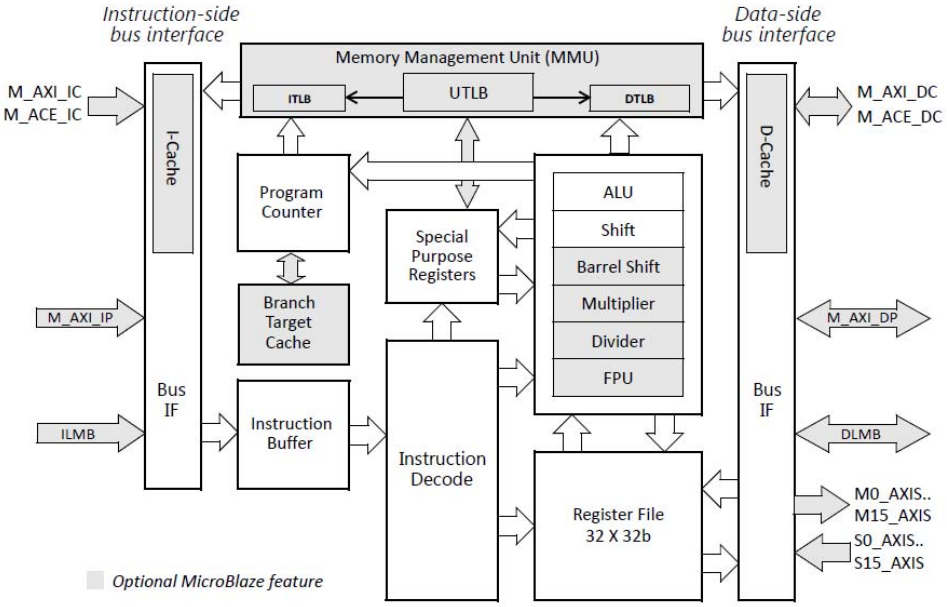


Figure 4.1: MicroBlaze architecture [3]

a connection to both on-chip and off-chip peripherals and memory. The ACE interfaces provide cache coherent connections to memory. The XCL interface is intended for use with specialized external memory controllers. MicroBlaze also supports up to 16 Fast Simplex Link (FSL) or AXI4-Stream interface ports, each with one master and one slave interface. FSL is a uni-directional point-to-point communication channel bus used to perform fast communication between any two design elements on the FPGA. The FSL interfaces are used to transfer data to and from the register file on the processor to hardware running on the FPGA. The performance of the FSL interface can reach up to 300 MB/sec. The FSL bus system is ideal for MicroBlaze-to-MicroBlaze or streaming I/O communications.

4.2.3 FPGA Development Tools

For embedded system development, a broad range of development system tools collectively called the ISE Design Suite is offered. The ISE Design Suite is a Xilinx development system product that is required to implement designs into Xilinx programmable logic devices. For embedded system design an Embedded Edition of the Xilinx ISE Design Suite is offered comprising main tools like the Integrated Software Environment (ISE), the Embedded Development Kit (EDK), and hardware Intellectual Properties (IP), drivers and libraries for embedded software development.

ISE is a Xilinx development system product that is required to implement designs into Xilinx programmable logic devices. EDK is a suite of tools and IPs that enables designing a complete embedded processor system for implementation in a Xilinx FPGA device. The EDK enables the design and integration of both the hardware and software using two main tools, Xilinx Platform Studio (XPS) and Software Development Kit (SDK). The XPS is the development environment used for designing the hardware portion of the embedded processor system. The SDK, based on the Eclipse open-source framework, is used for development and verification embedded software applications. The SDK is also available as a standalone program. Recently the ISE Design Suite is offered as the Vivado Design Suite with some additional features for system on a chip development and high-level synthesis.

4.3 PF Embedded Design Based on MicroBlaze Processor

The global architecture of the proposed system for the PF in a Grid-based Fast SLAM algorithm is given in Fig. 4.2. It comprises of an embedded Microblaze processor responsible for the execution of software functions, the PF hardware accelerator (PF HW accelerator) and, other peripheral such as timer and universal asynchronous receiver transmitter (UART) cores with the purpose to help in the analysis and verification of the system.

In the architecture shown in Fig. 4.2, the sampling module is responsible for generating particles by applying a probabilistic motion model of the robot and odometry sensor data. The importance weight module is responsible for the evaluation of the weights of the particles using the laser scanner measurement data and the occupancy grid map data. The resampling module conducts the evaluation of the replication factors of the particles based on their weight obtained from the importance weight module. For its operation this module requires a uniform random number generation. The OGM module performs map related operations. The PF

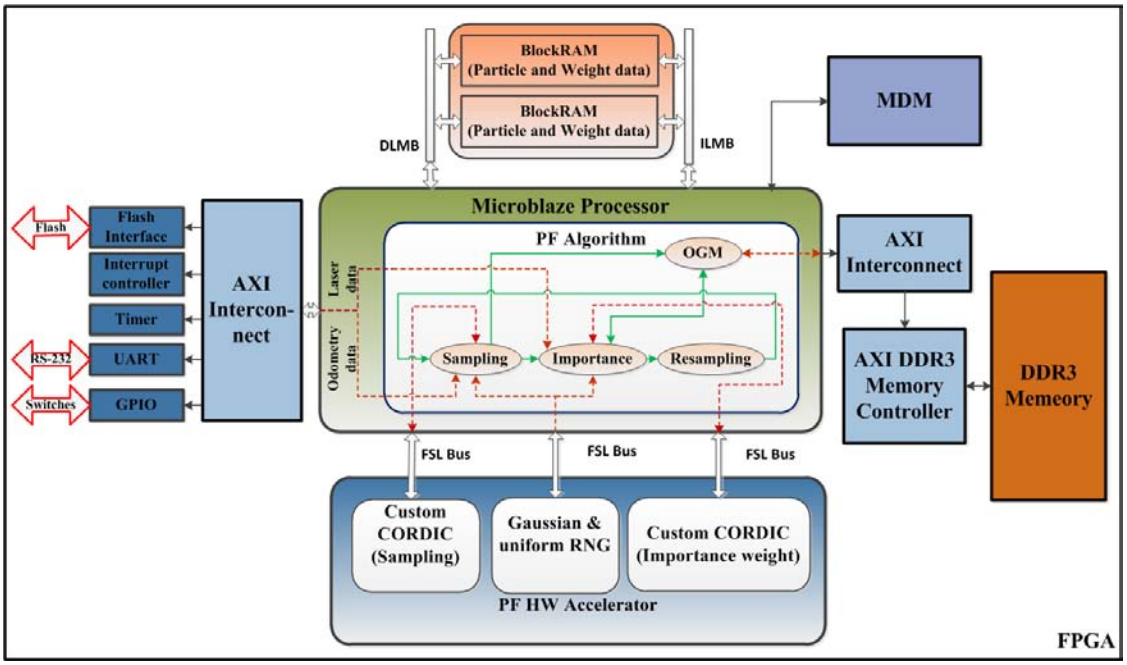


Figure 4.2: MicroBlaze processor based PF-SLAM FPGA architecture

hardware acceleration module accelerates the computational demanding steps of the PF to SLAM application. The PF HW accelerator contains the CORDIC and RNG cores where they are connected to the Microblaze soft-core processor through a dedicated one-to-one communication bus, FSL (Fast simplex Link) for fast streaming of data. The details of PF HW accelerator is shown in Fig. 4.3.

In the PF HW accelerator, individual CORDIC hardware modules are assigned for the evaluation of different functions in the sampling, $f_S()$, and importance weight, $f_I()$, steps and are configured through their *config* port. The CORDIC modules implements the architecture of the CORDIC module explained in Chapter 3, Section 3.4.3. In Fig. 4.3, the input/output interfaces to the CORDIC module is given with two possible inputs I_0 and I_1, and a three bit configuration input port *config* to choose a specific function for evaluation. Depending on the configuration bits on the *config* input interface to the CORDIC module, a specific function is evaluated and the result is provided at the output interface. Uniform (URN) and Gaussian random numbers (GRN) are provided to the resampling and sampling steps respectively by the RNG module. The hardware architecture of the RNG module comprises the Tausworthe URNG, CORDIC and Ziggurat GRNG sub-modules as explained in Chapter 3, Section 3.5.3. The Tausworthe URNG module is responsible for generation of two uniform random numbers (U_0 and U_1) that are used by the Ziggurat module and in the resampling step of the PF.

4.4 PF HW/SW FPGA Implementation

For verification of the proposed architecture shown in Fig. 4.2, both a realistic simulation generated log data and real experimental log data from odometry and laser scanner is interfaced with a DDR3 SDRAM memory in order to synchronize data for the processing. The odometry data is used in the sampling step to generate new particle instances and the laser data is used in the importance weight step for the evaluation of particle weights. The particles and their associated weights are buffered to a Block RAM for fast accessing.

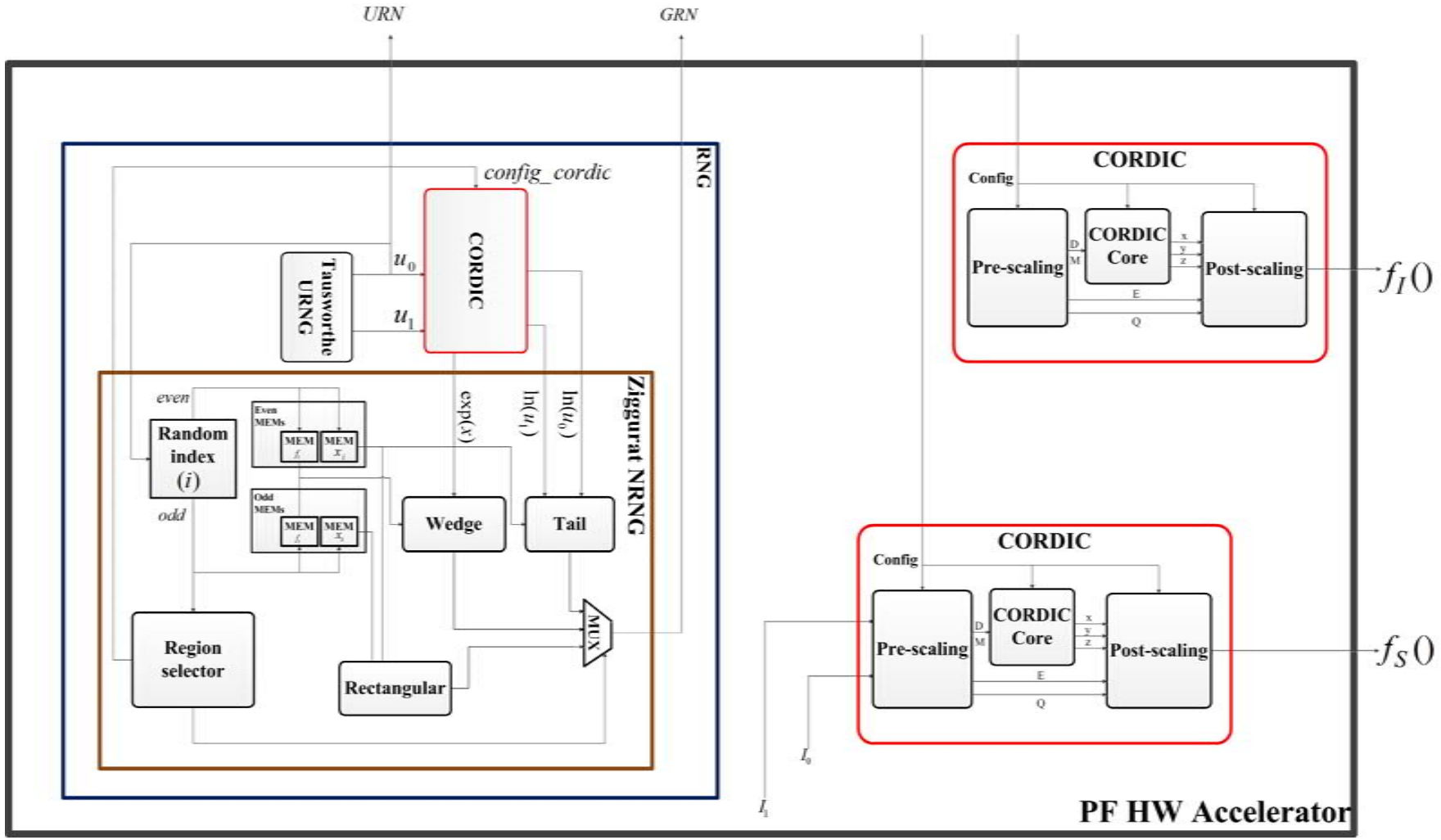
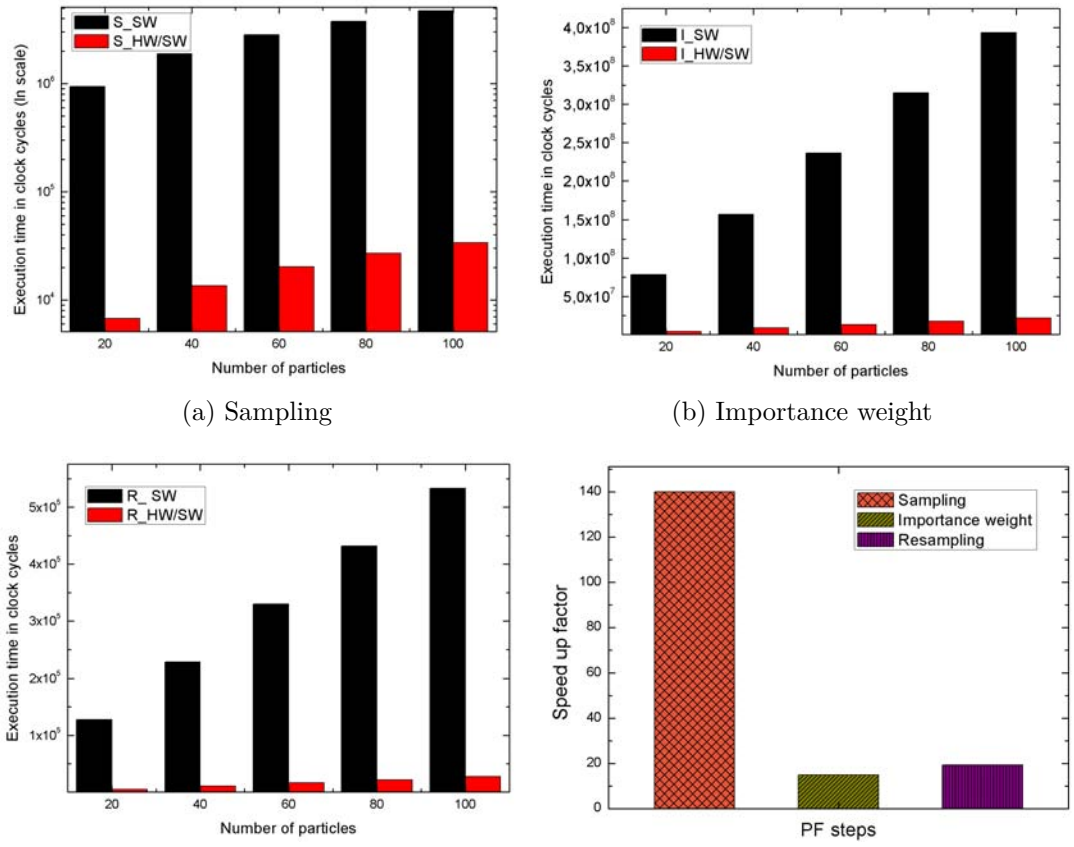


Figure 4.3: Internal structure of the PF HW Acceleration module

The Xilinx version 14.6 ISE, EDK and SDK are used for the design and implementation of the system architecture on Xilinx Kintex-7 KC705 FPGA device running at 100 MHz. The design of the hardware modules is written in VHDL language, where the different variables are represented as fixed point numbers with $Q_{(8:24)}$ format (i.e. 8 bits for the integer part and 24 bits for the fractional part). For the different variables in the software part of the algorithm a 32 bit floating point representation is used by enabling the FPU of the MicroBlaze processor.

Fig. 4.4 summarizes the execution time given by the number of clock cycles for each step of the PF in the Grid-based Fast SLAM algorithm. Fig. 5.16(a), 5.16(b) and 5.16(c) show the comparisons on the number of clock cycles required for sampling, importance weight and resampling steps respectively in the case of an



(a) Sampling (b) Importance weight

(c) Resampling (d) Speed up

Figure 4.4: Comparison on execution times for the sampling (S), importance weight (I) and resampling (R) steps for embedded software implementation (SW) vs HW/SW embedded implementation and the corresponding speed up factor in the execution time for each step (d).

embedded software and HW/SW implementations. Fig. 5.16(d) shows the overall speed up obtained with the HW/SW implementation over the embedded software implementation. The results in Fig. 4.4 show that, the hardware acceleration leads to better speed up in the execution time of the HW/SW PF in all the three steps of the algorithm. In particular, a significant speedup is achieved in the sampling step which can be attributed to the fast generation of Gaussian random numbers by the random number generator hardware module. Compared to the other steps, the importance weight steps shows a relatively higher clock cycles as shown in Fig. 5.16(b). This is due to the fact that, for the evaluation of the weight of each particle in the importance weight step, it is required to transform every laser scan measurements point data (range and bearing angle) from a robot frame of reference to a global frame of reference where normally more than hundreds of laser scans points have to be evaluated from the sensor. This requires the evaluation of the sine and cosine functions for every scan point. Furthermore, for every laser scan end point the computation of an exponential function and scan matching between occupied points in a map is required. These are the main reasons attributed to the relatively large clock cycles obtained in the importance weight step. In general, the presented approach leads to an improvement in the speedup of 140×, 14.87× and 19.36× in the sampling, importance weight and resampling steps respectively (Fig. 5.16(d)).

In respect to the resource usage of the implemented PF-SLAM on FPGA, Table 6.1 shows the hardware resource utilization of the whole HW/SW system architecture (Fig. 4.2) and the PF acceleration module (Fig. 4.3). It can be seen that few of the available FPGA resources are used for the implementation.

Table 4.1: Resource utilization for PF hardware acceleration module and the HW/SW co-design system on Xilinx Kintex-7 KC705 FPGA

Resources	PF HW Acc.	HW/SW System	Available
Slice registers	101 (0%)	9406 (2%)	407600
Slice LUTs	4216 (2%)	12246 (6%)	203800
DSP48E1s	12 (1%)	23 (2%)	840
BRAMB36E1	2 (0%)	42 (9%)	445

4.5 PF-SLAM Performance Evaluation

4.5.1 On The Use of Laser Scanners

In mobile robotic applications for performing SLAM efficiently, the robot needs to sense, calculate the distances to the obstacles and build the map for robot navigation. To achieve that, laser scanners are widely used in mobile robotics localization systems [4]. However, the high price tag in the most commonly used laser scanners, SICK LMS 200 and Hokuyo URG-04LX [5], has been a major drawback for many hobbyist and educational robotics practitioners which results in the need for alternative low cost laser scanners. This section presents the evaluation of the PF-SLAM implementation explained in sections 4.3 and 4.4 using also a low cost Neato XV-11 laser scanner and other typically used laser scanners. For the typically used laser scanners the proposed system is validated under complex robot environments based on real data sets.

4.5.2 On Neato XV-11 Laser Scanner

The domestic vacuum cleaner robot Neato XV-11 [6], shown in Fig. 4.5 (left) includes an alternative low cost 360 degree laser scanner [7]. The laser scanner can be removed from the robot Fig. 4.5 (right) in order to allow robotics practitioners to use it in their projects. Instead of using time of flight measurement, like the more expensive laser scanners, it uses triangulation to determine the distance to the target, using a fixed-angle laser, a CMOS imager and a DSP for subpixel

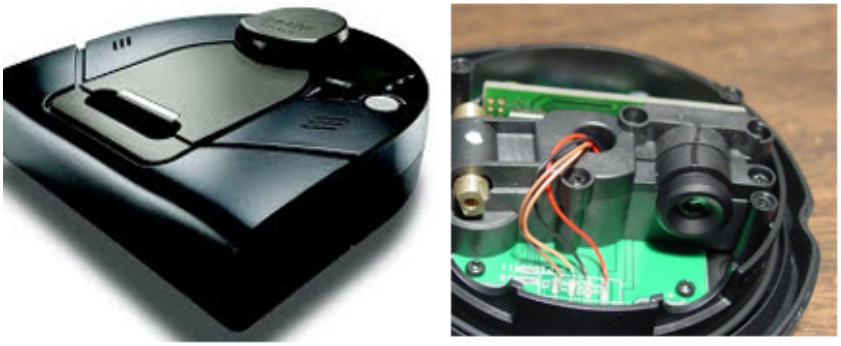


Figure 4.5: Neato XV-11.

interpolation [8]. The sensor establishes a serial communication with a 115200 bps baudrate, sending data up to about 5 Hz. Its power consumption without motor is relatively low: 145 mA at 3.3 V, which is very important factor in order to increase the autonomy of a mobile robot powered by only on-board batteries. It provides a 360 ° range of measurements, with an angular resolution of 1 degree, with its range from 0.2m up to 6m with an error inferior to 0.03m up to 6m measures.

In order to obtain all the data information required for evaluating the PF-SLAM system using this laser scanner, the following procedure can be established. First, using a SimTwo [9] realistic simulation software (screen shot shown in Fig. 4.6) that supports several types of robots the behavior of the laser scanner can be mimicked based on its model in a virtual simulation environment. Then a simulation is performed by navigating a robot equipped with a hacked Neato XV-11 laser scanner in a Robot@Factory competition maze [10]. Based on the actuator [11] and laser scanner models [12] used in the simulator, the simulation finally provides the robot odometry and laser scanner data information for performing SLAM on the FPGA.

4.5.3 Results Discussion

This section presents qualitative and quantitative results obtained based on the application of the proposed system on different laser scanners. First the proposed system is validated based on the Neato XV-11 low cost laser scanner using the procedure just introduced. Then, further evaluation of the system is conducted by applying datasets gathered with real robots from the two most commonly used laser

Figure 4.6: SimTwo simulation environment

scanners, Hokuyo and SICK. In all cases a total of 100 particles and an occupancy grid map with a resolution of 0.01 meters/map cells are used. The performance of the system is evaluated qualitatively on the generated occupancy grid map and by evaluating the error between the ground truth and the PF estimated pose. In the evaluation of the estimation errors in the pose of the robot, while no ground truth is available a measure of the error against an approximated robot path generated by the GMapping approach is used [13].

Neato XV-11

For the evaluation of the system on the Neato XV-11 laser scanner, a simulated Robot@Factory competition maze environment shown in Fig. 4.7 (top) is used. A robot is guided to navigate in the maze while collecting odometry and laser scanner data for post processing in the SLAM algorithm on the FPGA. Raw collected data is first processed to calibrate the odometry. Then synchronization between the calibrated odometry data and laser data is performed. The final data is used for performing the SLAM algorithm on the FPGA, with the objective of constructing a probabilistic occupancy grid map of the robot's maze environment while estimating the trajectory of the robot simultaneously. Fig.4.7 (bottom) shows the FPGA implementation result for the occupancy grid map generated based on the odometry and laser log data. The size of the occupancy grid map is $4meter \times 4meter$.

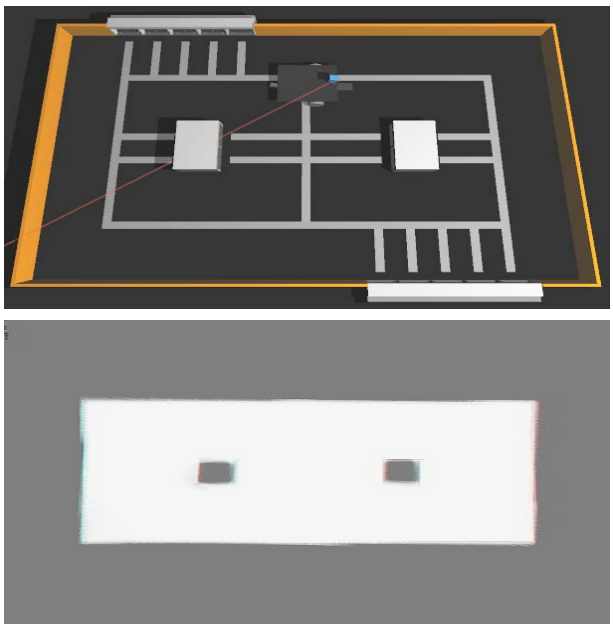


Figure 4.7: A simulation maze environment (top) and the occupancy grid map (bottom) constructed based on the laser model on the FPGA.

For qualitative analysis of the performance of the implementation, the plot of the trajectory of the robot obtained from odometry, ground truth and particle filter estimate are provided in Fig. 4.8. The results in Fig. 4.8 shows that the particle filter estimated path of the robot is close to the actual ground truth path of the robot, which confirms the good performance of the implementation. Besides the qualitative analysis, a quantitative evaluation of the performance of the particle filter to the pose estimation of the robot is performed by evaluating the error between the ground truth and the particle filter estimated pose (Fig. 4.9(a) and

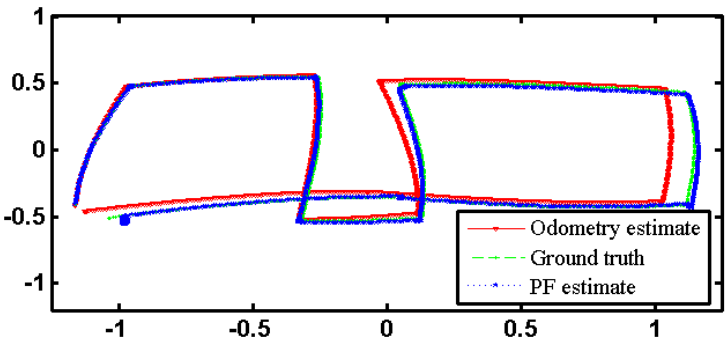
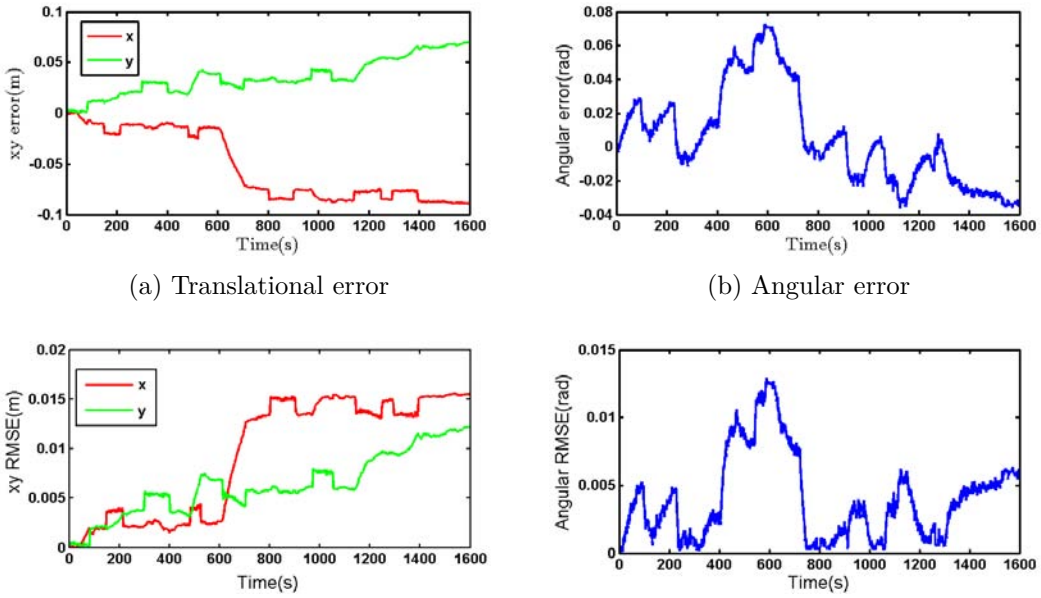


Figure 4.8: Robot trajectories for the odometry estimate, ground truth and the particle filter estimated path



(a) Translational error

(b) Angular error

(c) Translational RMSE

(d) Angular RMSE

Figure 4.9: Angular and translational errors for pose estimation

4.9(b)), and the root mean squared error (RMSE) metrics (Fig. 4.9(c) and 4.9(d)). The pose error shown in Fig. 4.9(c) and 4.9(d) is obtained by taking the difference of the sample mean of the particle filter with 100 particles and the ground truth data. The RMSE for the pose is calculated by averaging 50 independent runs of the algorithm at each time step.

Hokuyo

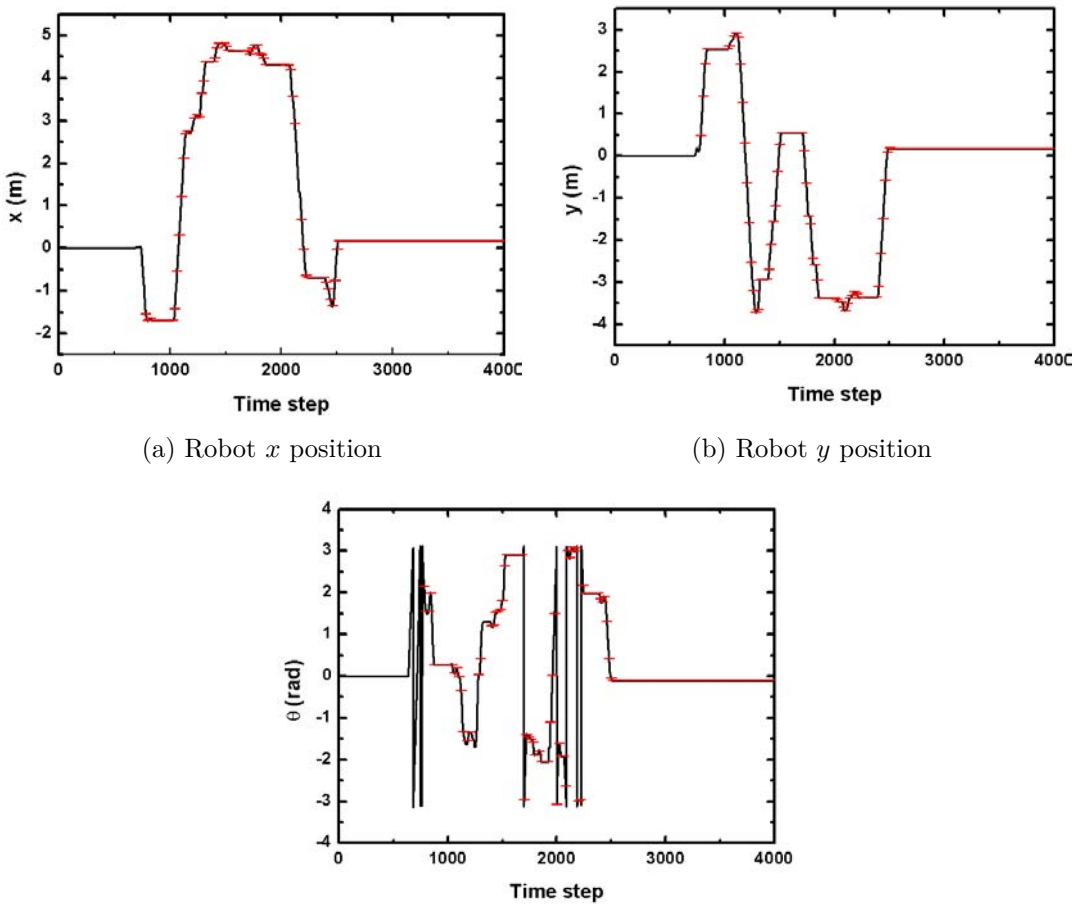
For the evaluation of the system based on a Hokuyo laser scanner, an experiment is conducted with a TurtleBot [14] mobile robot equipped with a Hokuyo laser scanner in an office environment. While the robot is navigating autonomously, raw odometry and laser scanner data are acquired for processing on the FPGA. The robot navigates starting from the inside of the lab and moves around a corridor and returns back to its starting position. The resulting map and particle filter estimated trajectory generated by the proposed system for this experiment is shown in Fig. 4.10. Imperfect edges in some parts of the generated occupancy grid map part of the office resulted due to the fact that there are a cloud of boxes and other materials in the office.

Figure 4.10: Generated map of a lab at UAB university

As no ground truth is available, the navigation error is estimated using the GMapping tool from ROS (Robot Operating System). Fig. 4.11 shows the evolution in the estimated robot pose with time and the corresponding errors shown as error bars. Our system shows no significant error for all the x, y, and θ of the robot pose.

SICK

Finally the proposed system is validated on SICK laser scanners involving large and complex indoor environments. The raw odometry and laser scanner data for conducting this validation is obtained from the Robotics Data Set Repository (Radish) [15]. In this case, a ground truth is neither available, therefore the estimation error is evaluated in respect to the pose obtained with the GMapping tool. The resulting map and estimated trajectory generated by applying our system to the Ubremen-Cartesium and MIT CSAIL data sets which are based on a SICK LMS laser scanner from Radish can be seen in Fig.4.12, top and bottom respectively. The raw odometry trajectory and the estimated trajectory of the robot are also shown in Fig 4.12, where in both cases our system is able to correct the high odometry error and generates the correct maps. The pose estimation errors for the Ubremen-Cartesium and MIT CSAIL data sets is shown in Fig. 4.13 and Fig.4.14 respectively, where low errors are achieved in both cases.



(a) Robot x position

(b) Robot y position

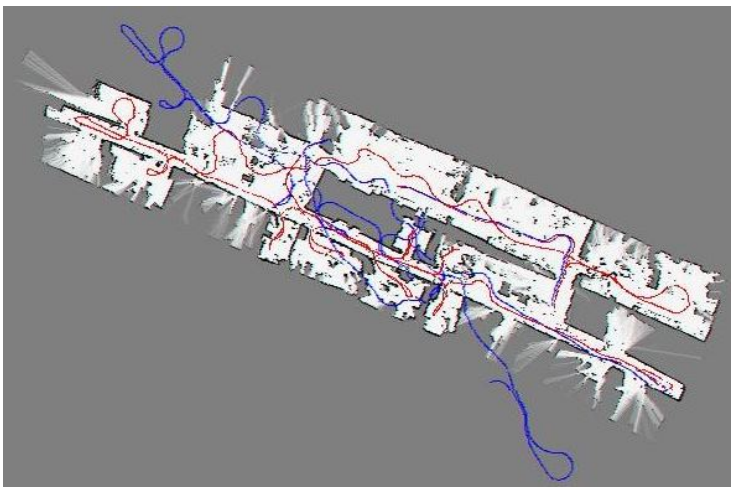
(c) Robot orientation (θ)

Figure 4.11: x, y, and θ of robot pose evolution with time and corresponding errors in pose estimation shown as error bars (UAB lab)

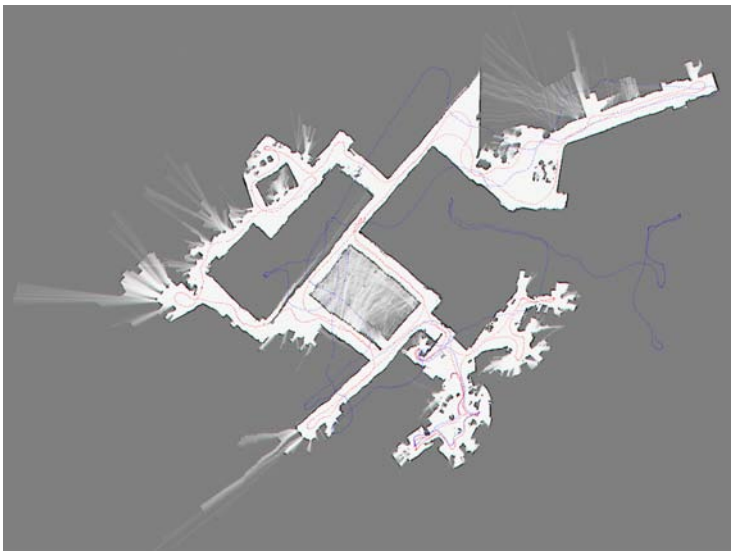
Summarizing Results

The mean of the pose estimation errors for all the data sets is summarized in Fig. 4.15 for the (x, y) position and orientation θ of the robot. The mean error is calculated by averaging 50 independent runs of the algorithm over a given number of time steps. The results show that in general the proposed system is capable to maintain low errors in the (x, y) positions and orientation θ of a robot. Furthermore, the results confirm that for the low cost laser scanner low estimation errors are also achieved.

In addition, a study on the effect of the number of particles on the estimation performance is conducted based on the RMSE metrics. Fig. 4.16 shows the results obtained for the effect of the number of particles on the RMSE for the pose of a



(a)

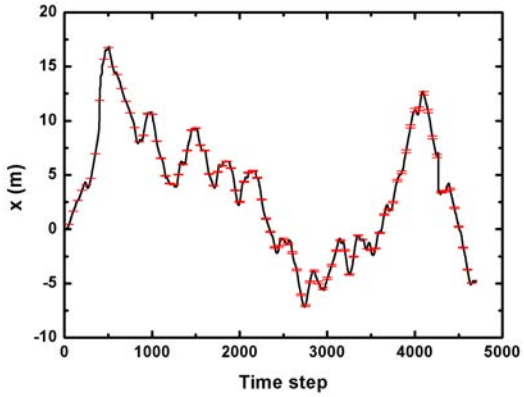


(b)

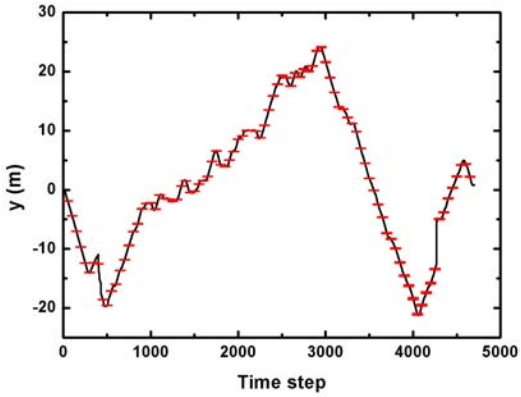
Figure 4.12: Map generated based on the Ubremen-Cartesium (a) and MIT CSAIL (b) data sets. The particle filter estimated and the raw odometry trajectories are shown with red and blue colors respectively

robot based on the Ubremen-Cartesium and MIT CSAIL data sets. The results of Fig. 4.16 shows a decrease in the RMSE with the number of particles.

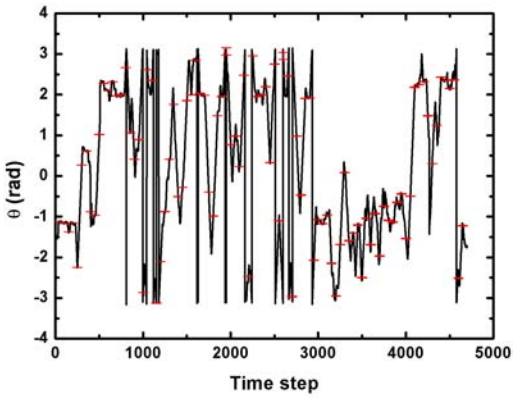
The performance evaluation of the system on low cost and standard laser scanners shows the goodness of the method in front of the very low cost alternative laser scanner. The results achieved with the application of the presented system using the low cost laser scanner shows good performance in the estimation and confirms that such low cost laser range finder can be used in low cost applications.



(a) Robot x position



(b) Robot y position

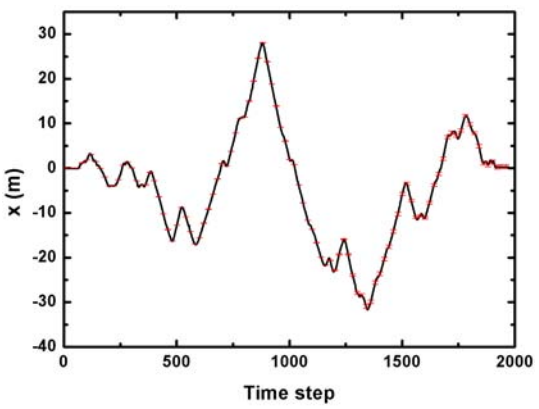


(c) Robot orientation (θ)

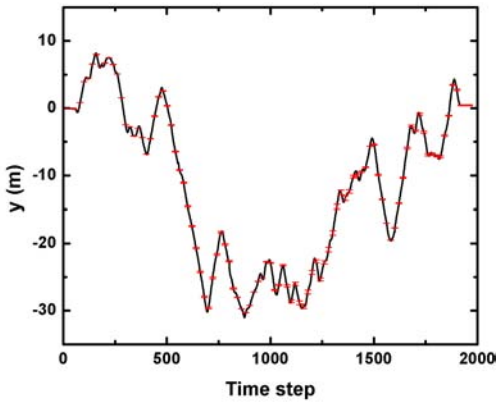
Figure 4.13: x, y, and θ of robot pose evolution with time and corresponding errors in pose estimation shown as error bars (Ubremen-Cartesium)

4.6 Parallel PF with Metropolis Coupled MCMC

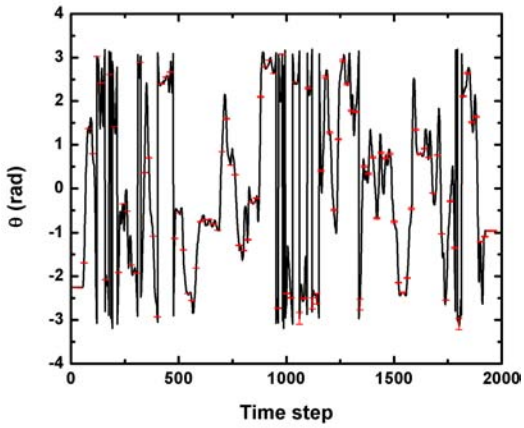
To make PFs amenable to real time applications, a speedup of their intensive computations through parallel processing is an appealing option. This can be achieved as most of the operations in PFs are performed independently. In particular, the sampling and importance weight computations can be easily parallelized and pipelined as there is no data dependencies between them. However the resampling step, in particular with traditional resampling methods, creates a bottleneck in the full parallelization of the whole PF steps computations. Furthermore, the acceleration of PF computations with parallel hardware implementation normally requires a significant amount of communication overhead between parallel particle processing units and a central unit. The huge communication overhead results due



(a) Robot x position



(b) Robot y position



(c) Robot orientation (θ)

Figure 4.14: x, y, and θ of robot pose evolution with time and corresponding errors in pose estimation shown as error bars (MIT CSAIL)

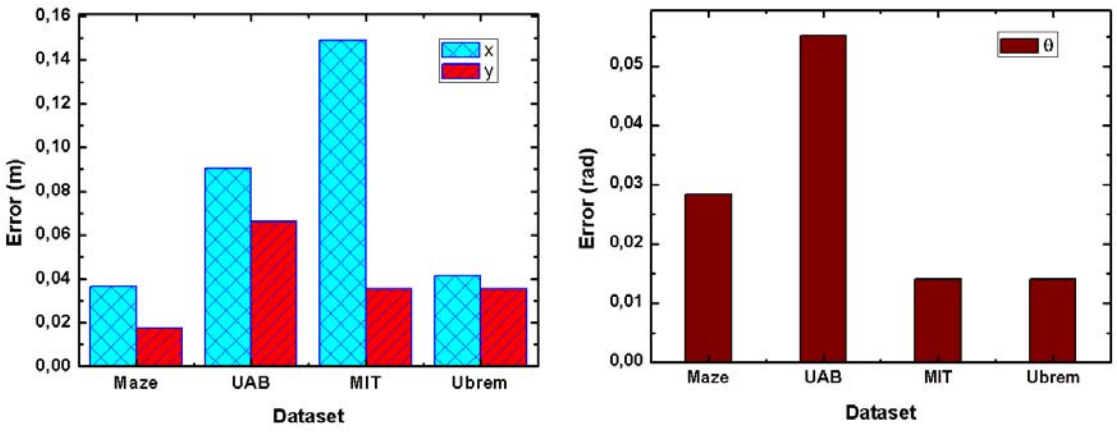
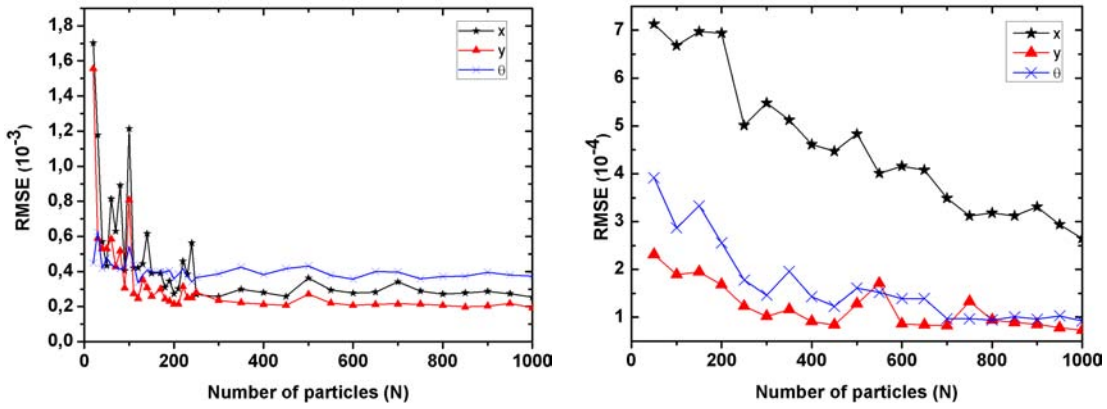


Figure 4.15: Comparison of mean errors for the pose (x, y, θ) of all datasets.



(a) (b)

Figure 4.16: Effect of number of particles on RMSE for Ubremen-Cartesium (a) and MIT CSAIL (b) data set.

to the large volume of particle information exchange in the resampling process[16].

To reduce the amount of data communication overhead and accelerate the whole PF computations, a parallel Metropolis Coupled MCMC ($P(MC)^3$) approach is introduced in this work. The $P(MC)^3$ approach has the advantages of bottleneck-free pipelined implementation with the sampling and importance weight steps, which leads to the reduction of the overall latency of the PF computation. Furthermore, the $P(MC)^3$ method is suitable for parallel PF realization as it involves few data exchange and provides simplified parallel architecture design.

The $P(MC)^3$ applies the IMHA algorithm (see Chapter 2, Section 2.8.5) by running n Markov chains in parallel. Some of the chains are *heated* by raising their posterior probability to a power β, where $0 < \beta < 1$. Incremental heat value of the ith chain is given by $\beta_i = 1/[1 + \Delta T(i-1)]$, where $\Delta T > 1$ is a temperature increment parameter [17]. The *heating parameter* of the first chain is 1 (i.e. $\beta_1 = 1$) and it is unaltered making it the only *cold chain*. The details of the $P(MC)^3$ is given in [17, 18]. The algorithm works by running Markov chains in parallel where heat values are swapped between two randomly selected chains at certain interval of the iteration to provide better mixing among the parallel chains through a central processing unit (CU). Based on the $P(MC)^3$ approach, our proposed PF algorithm follows the steps shown in Algorithm 13.

Algorithm 13 Parallel PF with $P(MC)^3$

1. Perform the *sampling* and *importance weight* computation in parallel as described in section II.
2. Perform $P(MC)^3$ in parallel PEs with IMHA by initializing each Markov chain with the respective first particle index.
   (a) Exchange heat values β_i between two randomly chosen PEs based on the heat acceptance probability [17] at certain interval of the iteration.
   (b) Send the acceptance probability P_a calculated in the PEs, where particles are accepted to the CU.
   (c) The highest P_a PE send its particle data to the CU and CU sends the particle data to the other PEs.
   (d) Based on the particle data of the PE with largest P_a, the rest of the PEs evaluate their proposed particles for acceptance as shown in Fig. 4.17.
   (e) Each PE proposes its next particle and evaluates the acceptance probability based on the current Markov chain particle.
   (f) Return to step 2 until all the particles of each PE are considered for resampling
3. Go to the *sampling* and *importance weight* steps

The exchange of β_i in step 2(a) improves the mixing of particles among the multiple Markov chain running in each processing elements (PEs). In step 2(b) the CU compares the acceptance probabilities P_a's for the proposed particles by each PE and requests the PE with the highest P_a value to send its particle data to the CU. It will be broadcasted to the rest of the PEs. Based on the particle data from the PE with the highest P_a value, and by proposing a particle from their Sampled Particle Memory, each PE evaluates the acceptance probability P_a in order to make the decision of whether to accept or reject the proposed particle. Fig. 4.17 shows the memory architecture for the processing of particle data from the Sampled Particle MEM in each PE during the resampling step, where the resampled particle set is stored in the resampled particle memory (Resampled Particle MEM). If a proposed particle P_i from the Sampled Particle MEM in a given PE is accepted then it is written to the same memory address in the Resampled Particle MEM. If the

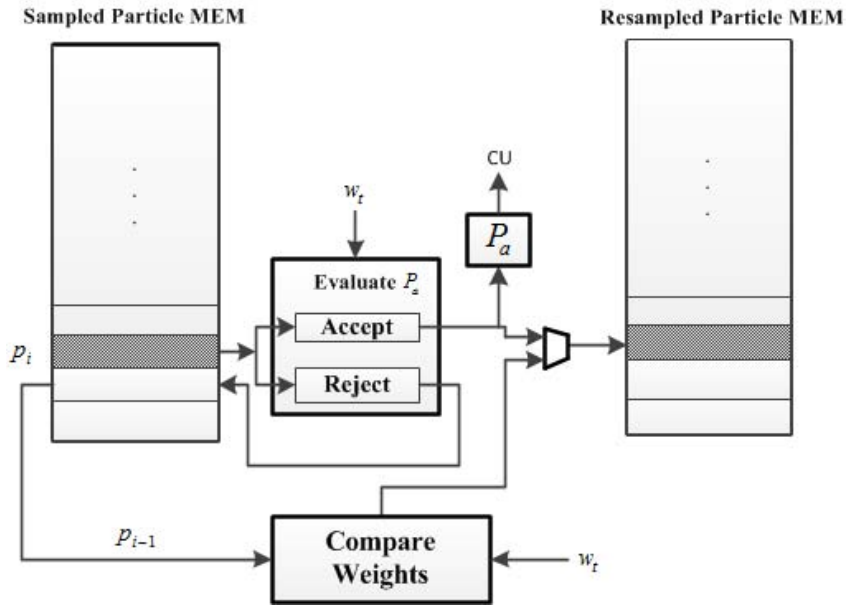


Figure 4.17: Sampled and resampled particles memory (MEM) schemes for Metropolis-coupled MCMC

proposed particle is rejected, then two possible particle candidates are considered for acceptance; one is the previous Markov chain particle p_{i-1} from the Sampled Particle MEM and the other is a particle received from the CU. By comparing the weights of these two candidates, the particle with the highest weight is accepted and written to the same memory index in the Resampled Particle MEM. This new approach provides good mixing of particle among the PEs as well as selecting particles with highest weight value.

4.7 Parallel PF Architecture

The parallel architecture for the proposed PF is shown in Fig. 4.18. It consists of several PEs and a Central Processing Unit (CU). All the particle processing steps *sampling* (S), *importance weight* (I) and *resampling* $(MC)^3$ are performed locally in each PE. A HW/SW co-design approach is used in the design of each PE, where the software parts of the implementation is realized based on an embedded processor, and the hardware part is based on a PF hardware acceleration (PF HW accelerator) block. The details of the design of PF HW accelerator block are given in Section 4.3.

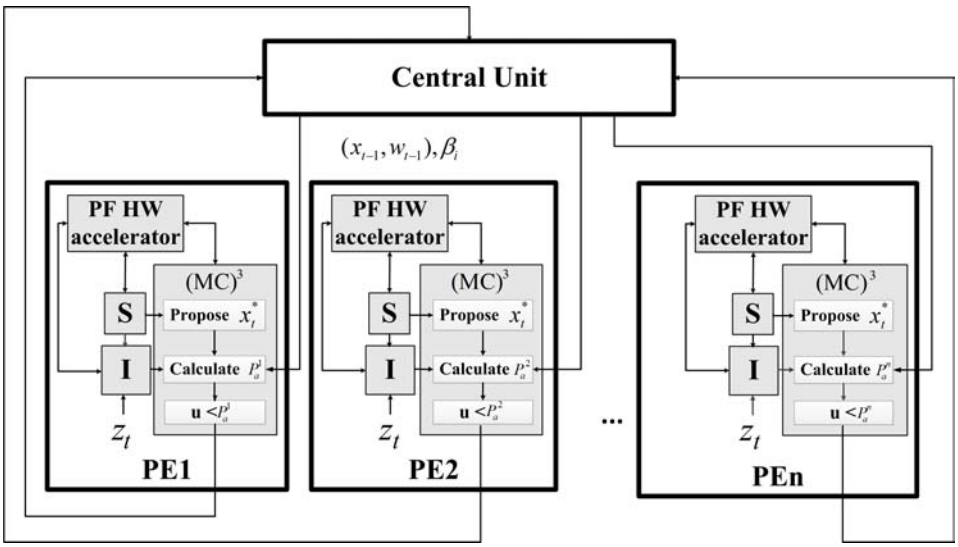


Figure 4.18: Parallel particle filter architecture

The CU performs simple tasks like global estimation and coordination among the PEs. Each PE communicates with the CU through a dedicated connection from its resampling step. The communication among the PEs is taking place through the CU that involves few data exchange such as the current chain of the Markov which consists of a single particle data, the heat values β_i, acceptance probabilities α and the inference value taken from the cold chain. One of the advantage of this proposal is that it leads to a distributed computation with very few information exchange between the parallel PEs through a CU. For example, considering the traditional parallel architecture [16], N weights and $N/2$ index factors have to be shared between the PEs and CU, and, in the worst case scenario, there could be $N/2$ inter-PE communication. In our proposed algorithm there is no inter-PE communication, which results in a reduced communication overhead and simplifies the parallel PF architecture design. Furthermore, as IMHA does not require all the normalized particle weights, resampling starts as soon as the first particle weight is available, and avoids the latency of the whole implementation.

4.8 Proposed Parallel PF Implementation and Results

The implementation of the parallel PF architecture presented in Fig. 4.18 is synthesized on a Xilinx Kintex 7-XC7K325T FPGA device. The CU is implemented using the Microblaze processor and the PEs are implemented using the PF HW/SW architecture shown in Fig. 4.2. Table 5.1 and Table 4.3 respectively provide the FPGA hardware resource utilization and the number of different Xilinx IP modules required in the synthesis of the parallel HW/SW PF architecture shown in Fig. 4.18, considering $n = 3$ PEs. Fig. 4.19 shows the FPGA hardware resources overheads of the parallel implementation (with $n = 3$ PEs), in respect to the serial implementation as shown in Fig. 4.2. The overheads are of 4.75×, 5.44×, 1.61× and 2.56× in slice registers, LUTs, DSP48E1 and RAM36E1 respectively.

The improvements in hardware acceleration obtained in the parallelization are made evident in Fig. 4.20(a) . It shows the processing time (number of clock cycles) of the PF by varying the number of particles and the number of parallel PEs. This result shows that with parallelization of the PF, significant improvement in the computation is achieved with parallel PEs compared to the serial implementation with single PE. The parallel implementation of the PF consisting of $n = 3$ PEs and for $N = 100$ particles, a speedup of 72.28× and 3.97× are achieved over an embedded software implementation and a serial single PE HW/SW implementation respectively. Fig. 4.20(b) shows the effect of increasing the number of parallel PEs

Table 4.2: FPGA resource utilization for serial PF and parallel PF with 3 PEs

Resources	Serial PF	Parallel PF with 3 PEs	Available resources
Slice registers	2950 (0.7%)	14025 (3%)	407600
Slice LUTs	3661 (1.8%)	19905 (9%)	203800
DSP48E1s	23 (2.7%)	37 (4%)	840
RAMB36E1	136 (31%)	348 (78%)	445

Table 4.3: Xilinx IP Resource utilization overview

IP Module	Number of Used
MicroBlaze	3
Fast Simplex Link (FSL) Bus	24
Local Memory Bus (LMB) 1.0	6
AXI Interconnect	2
Block RAM (BRAM) Block	9
LMB BRAM Controller	18
AXI 7 Series Memory Controller (DDR2/DDR3)	1
Processor System Reset Module	1
MicroBlaze Debug Module (MDM)	1
Clock Generator	1
AXI UART (Lite)	1
AXI Timer/Counter	1

on the overall execution time of the particle filter. The result of Fig. 4.20(b) further approves the processing time reduction with parallelism.

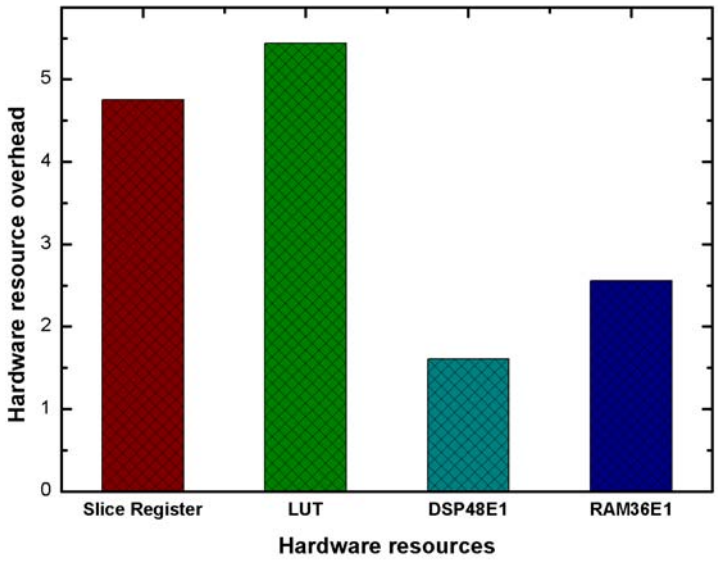
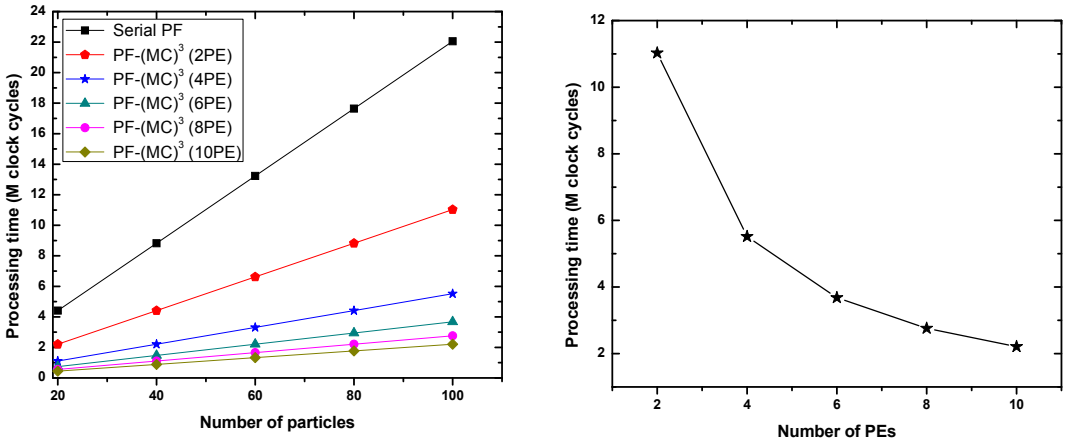


Figure 4.19: FPGA hardware resources overhead ratio of the parallel PF over the serial PF.



(a) (b)

Figure 4.20: Comparison between serial and parallel PF based on $(MC)^3$ (a) and timing for $N = 100$ particles with varying number of PEs (b)

4.9 Discussion on Parallel PF Implementations

In this section a discussion on the comparison of the performance of the proposed parallel PF with other parallel PF implementations is provided. However, the direct comparison of the proposed parallel PF with other parallel PF HW/SW implementations is difficult due to the variations in the computations involved in the sampling and importance weight steps depending on the specific application of the PF. Nevertheless, taking this into account, some comparisons of the implemented work can be made with previous implementations from the literature. Table 4.4 presents a summary of works implementing the PF for different applications. It provides information on the application area, the implementation device and platform, the number of particles and parallel PEs, the respective hardware resource utilization and the execution time of the implementations considered for comparison.

Table 4.4: Comparisons with other Implementations

Ref.	[19]	[20]	[21]	[22]	**Proposed**
Application	Waveform-agile radar tracking	3D facial pose tracking	Localization and people tracking	Tracking	Grid-based Fast SLAM
Platform	HW	HW/SW	FPGA/CPU	HW	HW/SW
FPGA Device	Virtex-5 XC5VSX240T	Virtex-4SX	Virtex-6 XC6VSX475T	Virtex-5 XC5VLX50T-3-FF1136	Kintex-7 XC7K325T
N	1000	100	70	64	100
No. of PEs	4	2	2	8	3
Slice registers	5	87.57	9.36	31.14	3
Slice LUTs	7	41.03	39	52.12	9
DSP48E1s	9	66.67	48	35.42	4
RAMB36E1	9	96.35	48	6.67	78
Execution Time (ms)	0.00684	73.72	18	0.00258	110.31/**0.994**

The work presented in [19] is a total hardware based implementation based on IMHA resampling. Besides the local computation of the PF steps in each PE, their approach requires an extra one-dimensional grouping method to reduce communication overheads. This leads to extra computations like local maxima and minima in each PE and their transfer to a CU for the global maxima and minima computation. This approach requires the exchange of a total $(2G \times d \times n) + (4 \times n)$ data between the PEs and the CU, where G is the number of groups in each PE, d is the dimension of the state vector, and n is the number of PEs. In respect to this work, our approach requires only a total of 4 data exchange (i.e. a single particle data, the heat values β_i, acceptance probabilities p_a and the inference value taken from the cold chain). As a result, in our approach data exchange is fixed and not dependent on other parameters like the dimension of the state vector and the number of PEs compared to this work.

In [22] a two-step parallel architecture is proposed. In the first step the sampling, importance weight, and output calculations are carried out in parallel. Then in the second step a sequential resampling based on a modified residual systematic resampling [23] algorithm is conducted. For conducting the residual systematic resampling, normalization of the particle weights is required. This results in the exchange of N weight data among the PEs. To minimize such large data exchange, the authors suggested first each PE to calculate the sum of the weights of its particles and send that data to the CU. The CU uses the sum of the weights of the particles from each PE to determine the number replicated particle each PE is allocated. This minimizes the particle weight data exchange by a factor of k. However their presented scheme requires interconnection among the PEs for exchange of surplus particles, which results in the communication overhead and complicate the hardware design. Further more, extra resampling step is required to be performed by the CU.

In [20] a parallel PF architecture based on an array of PEs, and a resampling unit along with a set of parameterized interfaces is proposed. The parameterized interfaces are suggested as a means to a parameterized design framework for different application of the particle filter. In the implementation a centralized resampling

based on the systematic resampling is considered, which results in immense data exchange between the PEs and the central resampling unit [24]. In [21] a HW/SW design based on an FPGA and a multi-threaded host CPU is realized for robot localization application. The sampling and importance are implemented on the FPGA with a long pipeline for processing many particles are at once and the resampling is implemented on the host CPU. Their approach requires the host CPU to gather all the particle data from the FPGA via PCI Express link to perform the resampling and particle size adaptation.

In contrast to the implementations given in Table 4.4, the proposed implementation shows a significant improvements in hardware resources utilization, except the block RAM requirements (due to the BRAM usage resulted from the occupancy grid map representation). The execution time of $110.31ms$ for our implementation in Table 4.4 account to the whole PF computations applied to the SLAM problem. SLAM is a complex problem in robotics, where it involves complex system and measurement models and requires intensive data processing. The intensive processing results due to the occupancy grid map update procedure and the requirement of processing large number of laser range finder sensor data. As shown in Fig. 3.3 of Chapter 3, most of the computations in the PF for the SLAM application lies in the importance weight step (69.03%). Where most of the computations in this steps account to the processing of laser data and the occupancy map update procedure. For fair comparison, if the number of clock cycles accounted for these procedures is neglected, the parallel implementation with 3 PEs could result in an execution time of about 0.994 ms as shown in Table 4.4. Therefore, the parallel processing of the large sensor data and map update procedure would be specially helpful for further speedup of the PF computation in robotics application where 100 ms level of sensor update rate is quite common.

The performance of our system is compared to parallel PF implementations in Table 4.4 which are based on a totally hardware or a HW/SW solution. Without considering the intensive sensor data processing and occupancy map update procedures, our implementations resulted in an improved performance while compared

to the execution time of the HW / SW based implementation in [20] and [21]. The low execution times reported in [19] and [22] are based on a total hardware parallel implementation for a tracking application, where compared to SLAM application the system and measurement models are simpler and does not involve large sensor data processing and memory access operations (as required for the occupancy grid map update procedure). While considering parameters like the amount and complexity of data exchange schemes and CU design complexity our algorithm shows optimization in respect to the works presented in [19–22]. For example, in these studies an additional particle regrouping step is required to be performed which introduces extra computation and requires certain information exchange between PEs and CU. In general, similar characteristics of the obtained results are reported by these authors considering different types of resampling algorithms. In general, besides the very complex arithmetic operations in the models involved in SLAM and the requirement of large sensor / memory data processing, compared to other FPGA based implementations of the PF, the proposed $P(MC)^3$ based parallel PF demonstrated improved processing speed as a result of the PF parallelization.

References

[1] http://www.xilinx.com/products/design-tools/microblaze.html.

[2] https://www.altera.com/products/processors/overview.html.

[3] *MicroBlaze Processor Reference Guide- Embedded Development Kit EDK 14.7*. English. Version UG081 (v14.7). Xilinx.

[4] K. O. Arras and N. Tomatis. "Improving robustness and precision in mobile robot localization by using laser range finding and monocular vision". In: *Advanced Mobile Robots, 1999.(Eurobot'99) 1999 Third European Workshop on*. IEEE. 1999, pp. 177–185.

[5] M. Alwan, M. Wagner, G. Wasson, and P. Sheth. "Characterization of Infrared Range-Finder PBS-03JN for 2-D Mapping". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. 2005, pp. 3936–3941.

[6] *Hacking the Neato XV-11*. https://xv11hacking.wikispaces.com/. Accessed:2015.

[7] *Neato Robotics XV-11 Tear-down*. https://www.sparkfun.com/news/490. Accessed:2015.

[8] P. Shah, K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, Y. Liu, H. M. Khan, J. Pinzarrone, L. Salinas, H. Tang, et al. *Distance sensor system and method*. US Patent 8,996,172. 2015.

[9] P. Costa, J. Gonçalves, J. Lima, and P. Malheiros. "Simtwo realistic simulator: A tool for the development and validation of robot software". In: *Theory and Applications of Mathematics & Computer Science* 1.1 (2011), p. 17.

[10] J. Gonçalves, J. Lima, P. Costa, and A. Moreira. "Manufacturing education and training resorting to a new mobile robot competition". In: *Conference on Flexible Automation and Intelligent Manufacturing FAIM 2012*. Tampere University of Technology. 2012.

[11] J. Gonçalves, J. Lima, P. J. Costa, and A. P. Moreira. "Modeling and simulation of the EMG30 geared motor with encoder resorting to simtwo: the official robot@ factory simulator". In: *Advances in Sustainable and Competitive Manufacturing Systems*. Springer, 2013, pp. 307–314.

[12] D. Campos, J. Santos, J. Gonçalves, and P. Costa. "Modeling and simulation of a hacked neato XV-11 laser scanner". In: *Robot 2015: Second Iberian Robotics Conference*. Springer. 2016, pp. 425–436.

[13] G. Grisetti, C. Stachniss, and W. Burgard. “Improved techniques for grid mapping with rao-blackwellized particle filters”. In: *Robotics, IEEE Transactions on* 23.1 (2007), pp. 34–46.

[14] `http://www.turtlebot.com`.

[15] `http://radish.sourceforge.net/index.php`.

[16] M. Bolic. “Architectures for efficient implementation of particle filters”. PhD thesis. Stony Brook University, 2004.

[17] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist. “Parallel metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference”. In: *Bioinformatics* 20.3 (2004), pp. 407–415.

[18] C. J. Geyer. “Markov chain Monte Carlo maximum likelihood”. In: (1991).

[19] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola. “Algorithm and parallel implementation of particle filtering and its use in waveform-agile sensing”. In: *Journal of Signal Processing Systems* 65.2 (2011), pp. 211–227.

[20] S. Saha, N. K. Bambha, and S. S. Bhattacharyya. “Design and implementation of embedded computer vision systems based on particle filters”. In: *Computer Vision and Image Understanding* 114.11 (2010), pp. 1203–1214.

[21] T. C. Chau, X. Niu, A. Eele, W. Luk, P. Y. Cheung, and J. M. Maciejowski. “Heterogeneous Reconfigurable System for Adaptive Particle Filters in Real-Time Applications”. In: *ARC*. Springer. 2013, pp. 1–12.

[22] H. A. A. El-Halym, I. I. Mahmoud, and S. Habib. “Proposed hardware architectures of particle filter for object tracking”. In: *EURASIP Journal on Advances in Signal Processing* 2012.1 (2012), pp. 1–19.

[23] M. Bolic, A. Athalye, P. M. Djuric, and S. Hong. “Algorithmic modification of particle filters for hardware implementation”. In: *Signal Processing Conference, 2004 12th European*. IEEE. 2004, pp. 1641–1644.

[24] A. Athalye, S. Hong, and P. M. Djuric. “Distributed architecture and interconnection scheme for multiple model particle filters”. In: *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*. Vol. 3. IEEE. 2006, pp. III–III.

5

HW Approach for PF-SLAM Processing Element Design

5.1 Introduction

For a high throughput PF realization, this chapter presents the design of the PF processing element (PE) with hardware implementation that includes the three step of the PF. For the development of a generic PF PE, the design approach considered in this chapter primarily focuses on the high-level data and control flow with the aim to provide the flexibility for easy integration of the design to different applications of the PF. As part of the PF PE design, important challenges in SLAM are also solved. Such as the design and implementation of a parallel laser scanner co-processor has been realized. The hardware design of the parallel laser co-processor has been realized with the objective of accelerating the processing of large measurement data from a laser scanner sensor in the update step of the PF applied to the SLAM problem. The hardware realization of Bresenham line drawing algorithm [1] that is applied in our application is also presented. Finally, the analysis and evaluation on the performance of the proposed PF PE based on real dataset is presented.

5.2 Proposed System Architecture

The high level block diagram of the proposed hardware architecture shown in Fig. 5.1 summarizes the global architecture for the PF implementation to the SLAM problem. The architecture is composed of two major hardware computational units, the PF processing element (PE) and laser range finder parallel processor (LRF parallel processor). The PE performs all the three steps of the PF, the sampling (S), importance weight (IW) and resampling (R). The LRF parallel processor performs the parallel processing of laser scanner data from a laser range finder (LRF) sensor and communicate with the PE module. Information about the position of the robot is obtained from an odometry sensor and used by the PE module as per the system model equations in the sampling step of the PF to the SLAM application. The details of the description on the hardware design of the LRF parallel processor and the PE modules are given in section 5.3 and 5.4 respectively.

5.3 Laser Scanner Parallel Co-processor Design

In PFs, an update to all the particles before the arrival of new measurement data is required. However, for systems with complex sensor models it is possible

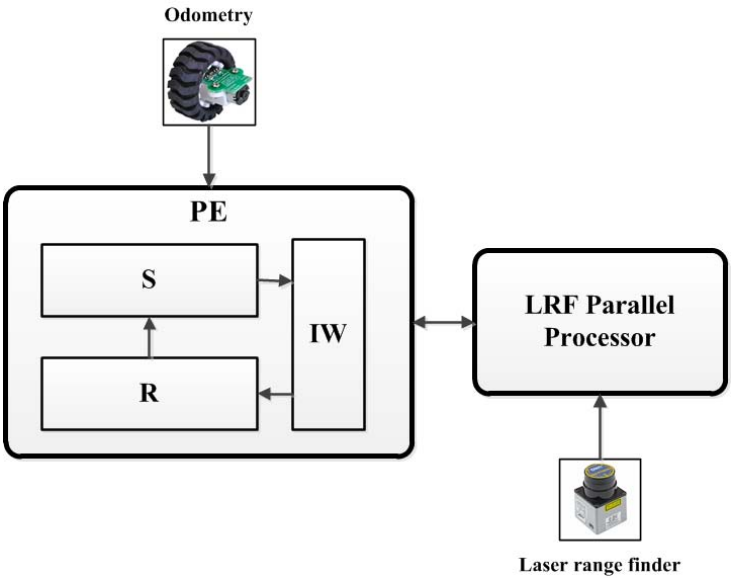


Figure 5.1: System architecture

that the update not being performed to all the particles before the arrival of new measurement data. This results in the discarding of valuable measurement information that cannot be processed in time.

In SLAM, a robot uses sensor measurements (laser or ultrasonic scanners, cameras, etc.) and odometry measurements to acquire information over its environment and perform localization and mapping. SLAM involves the processing of large measurement data at higher rate before every update step of the PF. Observations obtained based on laser scanner, normally involves several hundreds of scan points at a time. For each observation received, all the three steps of the PF are performed to a large number of particles (typically several hundreds or thousands). Once all the particles are processed through these three steps, the estimate of the state at the sampling instant is calculated and the next observation can be processed. These computations present significant computational load to the application of the PF to SLAM in real time. Furthermore, as the processing of such a large amount of sensor data with a central processor unit is a time consuming task, and for a single central processor unit computing the state estimate using PF even with a few sensor data is computationally too intensive, it requires the design of dedicated hardware for performing this task.

SLAM implementations based on laser scanner requires the projection of individual laser scan end points into a global coordinate space of the map, and the tracing of all the grid cells coordinates along the path of the individual laser beams. For projection of the individual scan points into the global coordinate frame of the map m it is required the pose (x, y, θ) of the robot in the global coordinate, the relative position (x_{sens}, y_{sens}) of the laser scanner sensor in the robot coordinate and the angular orientation θ_{sens} of the sensor beams relative to the robots heading. The end point of the individual laser scan point k at time t, (given by Z_t^k) for detected obstacles in the robot's environment is mapped to the global coordinate system via trigonometric transformation (as shown in Fig. 5.2). Performing such computations and tracing of the grid cells along individual laser beams for all the particles at a given time t leads to a major bottleneck in the PF computations. As a result the

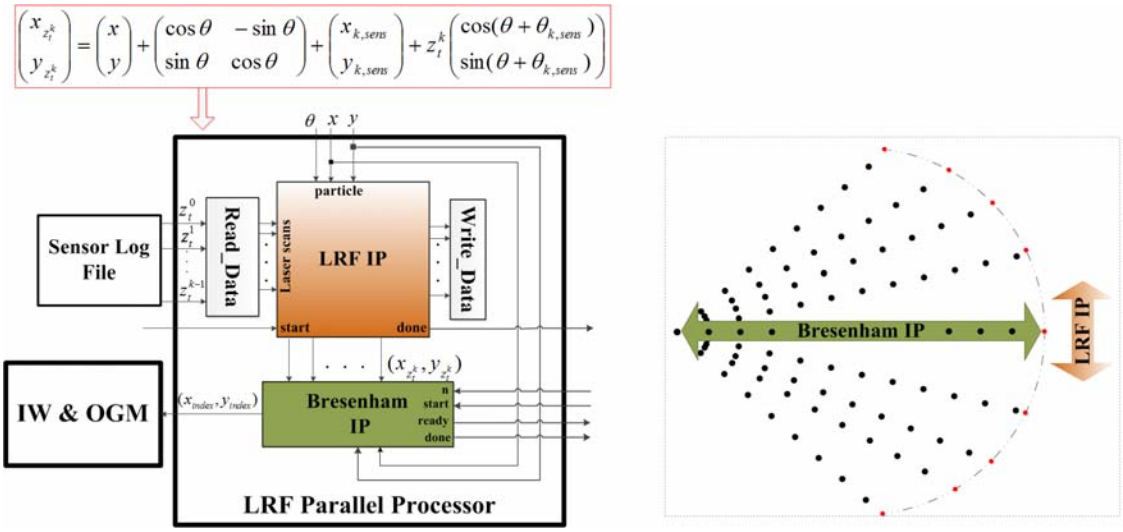


Figure 5.2: Laser scanner parallel processor

speedup of such computations with parallel hardware design is required to avoid the computational bottleneck of the PF application to SLAM.

The proposed hardware architecture of the LRF parallel co-processor is shown in Fig. 5.2, where the top module of this architecture is the laser range finder (LRF) parallel processor which contains two hardware IP cores: the LRF IP and the Bresenham IP cores. The LRF IP is the module where the parallel projection of laser scan points takes place. The Bresenham module performs the generation of the index of the grid cells along the path of the individual laser scans. These two main operations of the LRF parallel processor are also shown in Fig. 5.2 (right), where the processing of laser scanner data in hardware is performed by first generating the coordinates of the end points of the laser scans by the LRF IP core. Then using the Bresenham IP core the index of all the points between the start point of the laser (which corresponds to the particle states) are generated. These indices are used to access the gird cell in the global map, where the occupancy values are read and used in the evaluation of the weight of the particles and in updating the map of the robot environment. The Read_Data and Write_Data modules, are used for verification purposes, where the Read_Data module performs the reading of sensor log data from a file and provide it to the LRF IP module and the Write_Data module performs task such as the writing of map data to

a file. The importance weight and occupancy grid map ($IW\&OGM$) module is part of the PF hardware design explained in section 5.4.

After laser data is processed by the LRF parallel processor, a coordinate transformation is performed from the robot coordinate to a hardware coordinate in the $IW\&OGM$ module as shown in Fig. 5.3. The hardware coordinate is an integer representation of the map. However, the actual map data is stored in the map memory after a simple address mapping is performed on the map data from the hardware coordinate. An example for the transformation a single point from the robot's coordinate to the hardware coordinate and finally the corresponding memory index in the map memory is illustrated in Fig. 5.3.

5.3.1 Description of the LRF IP Core

The LRF IP core architecture shown in Fig. 5.4 mainly composed of the *Compute*, *CORDIC* and *SCB_R* modules. The *Compute* module performs the parallel projection of the laser scan end points into the global coordinate. For the projection of the laser end points, initially the sine and cosine of the bearing angles of the

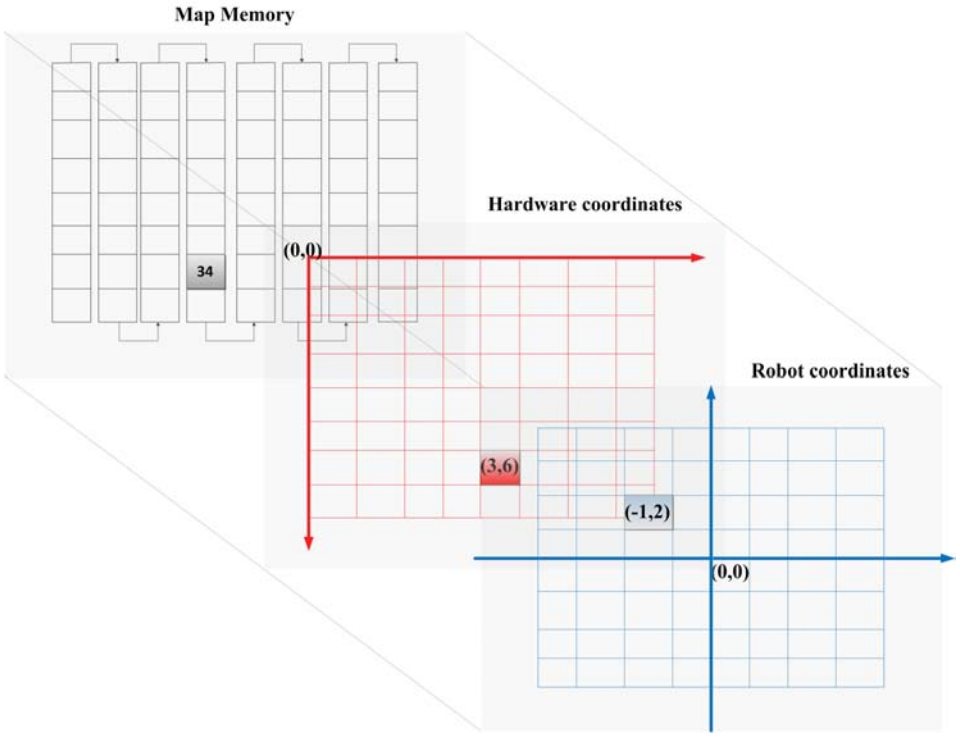


Figure 5.3: Representation of map in hardware

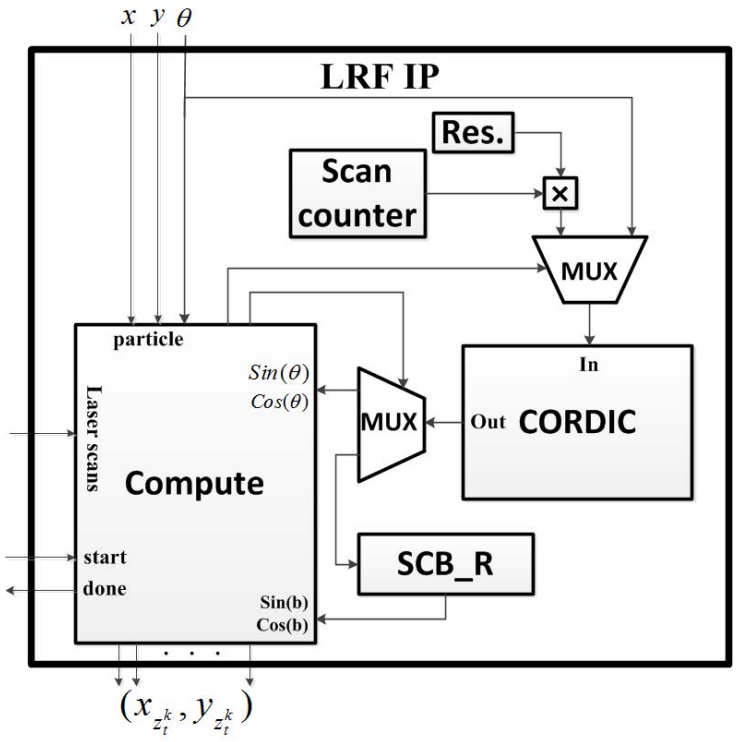
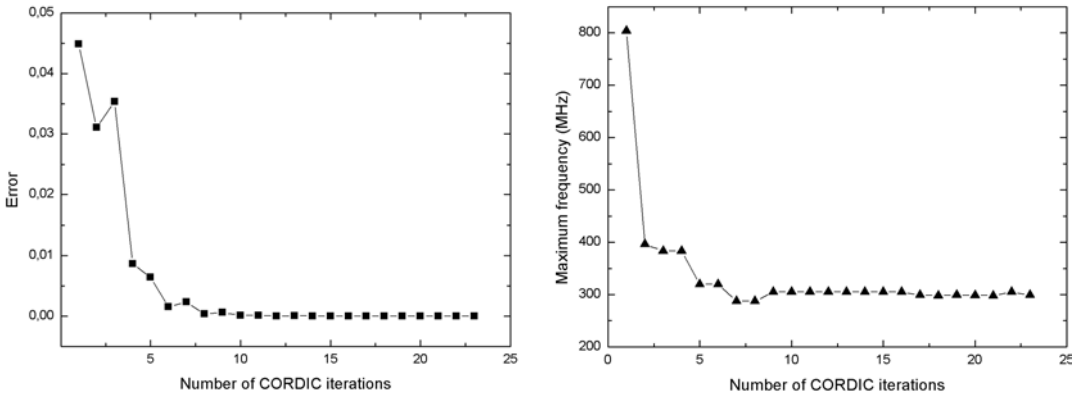


Figure 5.4: LRF IP architecture

individual scan points are computed by the Cordic module. For this computation, a product of the bearing angle resolution (*Res.*) and the index of the scan points from the *Scan counter* modules is used as an input to the *CORDIC* module and the results of the computations are stored in a register (SCB_R). Finally, based on a particle pose (x, y, θ), *SCB_R*, and *CORDIC* data all the necessary computations are performed in the *Compute* module by parallel processing of the laser scans which enables in the projection of all the laser scan points to the global coordinate in a single clock cycle.

As CORDIC is the core of the LRF IP, its performance and hardware resource requirement is mainly dependent on the configuration of the CORDIC processor (number of iterations and serial / pipelined configuration). However, a trade off in performance and hardware resources is required under these configurations. First, a study on the optimal number of CORDIC iteration is undertaken in order to identify the corresponding error in the computation. The results of this study is given in Fig. 5.5 (a), where with 10 iterations a precision of about 1.29983×10^{-4} is



(a) (b)

Figure 5.5: Effect of the number of CORDIC iterations on precision (a) and maximum frequency (b) of the LRF parallel processor

assured. Fig. 5.5 (b) shows the effect of the CORDIC iteration number on the speed of the LRF IP core. The computation of trigonometric functions using CORDIC algorithm is essential in order to obtain the maximum frequency of operation in the LRF IP core. A 10 bit CORDIC with pipelined configuration is considered in this work for minimal error and hardware resource.

5.3.2 Description on Bresenham IP Core

The Bresenham IP core is responsible for evaluating the index of the grid cells along the path of the individual laser scans. For the evaluation of the index of the grid cells, it uses the end point of the individual laser scan computed by the LRF IP core and the position (x, y) of a given particle which is input to the LRF parallel processor.

The Bresenham IP core design is based on Bresenham algorithm [1], which consists of basic operations such as addition, subtraction and bit shifting. Such basic operations are suitable for high-speed hardware implementation, and consequently it is considered a good choice for generating the index of the grid cells along the path of the laser scan for the application at hand. The input to the Bresenham IP core are two set of points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, i.e. the starting point and the end point of the laser scan point respectively.

The implementation of the Bresenham IP core is based on a finite state machine shown in Fig. 5.6 with five states: *Idle*, *Initialization*, *Bresenham*, *Stole* and *Done*

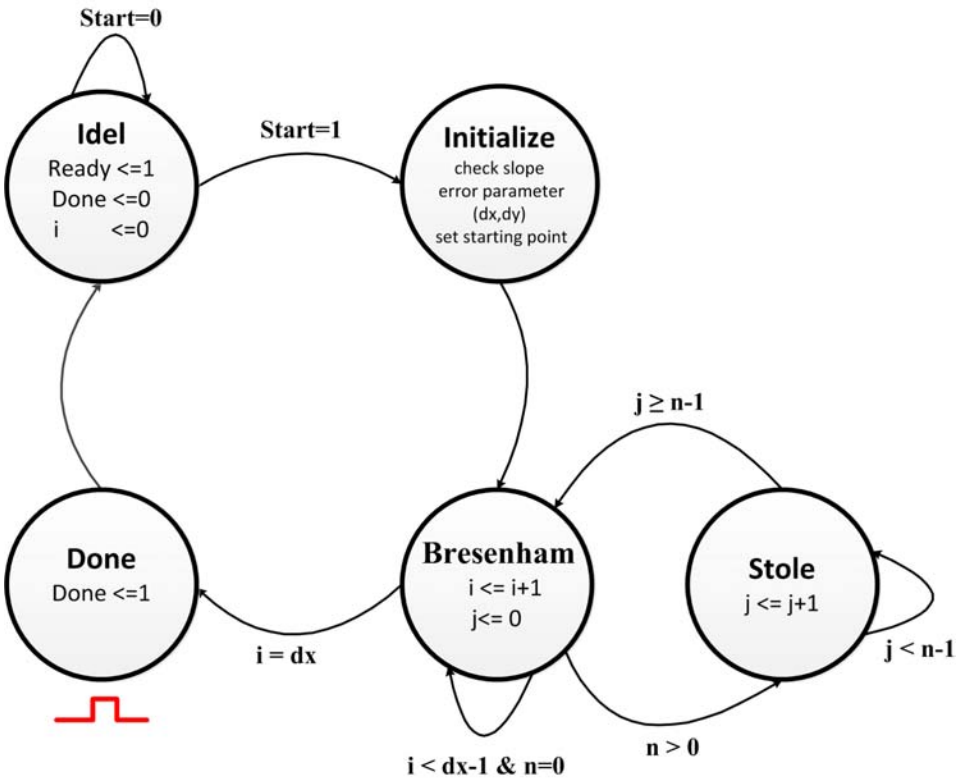


Figure 5.6: Finite state machine design for Bresenham IP core

states. In the *Idle* state the core waits until a single bit *Start* signal is asserted to 1, others wise it remains in this state. Once the *Start* bit is set to logic 1, the *Initialization* state proceeds. In this state all the necessary initialization routines to the Bresenham algorithm takes place and others parameters are also calculated based on the input points P_1 and P_2. For example, the slope of the line between the two input points is checked and swapping of the points takes place, the error parameter is calculated and starting point of the line trace is set in this state. Once it is finished it proceeds to the *Bresenham* state, where a loop is run for generating the (x, y) index of a total of dx number of grid cells along the two input points. Once all the index are generated it proceeds to the *Done* state. In this state the necessary control tick signals for the synchronization of Bresenham IP with the other hardware modules are generated, then follows the *Idle* state where it waits for the next operation. The *Stole* state is used for halting the generation of the next grid cell points for n clock cycles which is used for synchronization while using parallel Bresenham IP modules.

The exchange of data from the LRF IP to the Bresenham IP is conducted once all set of the (x, y) end points of the laser data are computed by the LRF IP core. Every clock cycle an index of a grid cell is returned by the Bresenham IP, and once the index of all grid cells are returned the next end point of the LRF IP output is considered for computation and this process continues until all the LRF IP core output end point are evaluated. As a result a large number clock cycles are normally required for evaluating the index of all grid cell along the paths of the starting point of the laser scan and the corresponding scan end points. In order to minimize the number of clock cycles, parallel processing of the end points of the LRF can be achieved by pipelining the Bresenham IP. If a total of M laser end points from the LRF IP are distributed to b parallel Bresenham IP modules, then $B_b = M/b$ laser end points are processed by each Bresenham IP module. This results in the generation of b grid cell indices at every clock cycle compared to a single grid cell with sequential processing.

Fig. 5.7 shows the parallel processing of the laser end points from the LRF IP modules for $b = 16$ parallel Bresenham IP modules. As shown in the figure, the laser scan end points from the LRF IP is partitioned into 16 sets, where the end points in each set are processed by a dedicated Bresenham IP. The set of points generated from each Bresenham IP module are then multiplexed to a corresponding section of the occupancy grid map during the weight and map update stages by the MUX unit. Instead of using a single large memory module for the representation of the occupancy grid map of the robot's environment, a total of b small map memory modules are used. Each b map module is configured for 256×256 grid cells size, which results in a map size of 1024×1024 grid cells for b memory modules. This approach allows n parallel read and write operations to be performed in two clock cycles, compared to a single read and write operation if a single map memory is considered during the occupancy grid map update step. Besides performing the multiplexing of the output points from the individual Bresenham IPs, the *MUX* unit also determines the number of scan points from the parallel Bresenham IPs falling in a given map memory. This is used by the Bresenham IPs for waiting a given

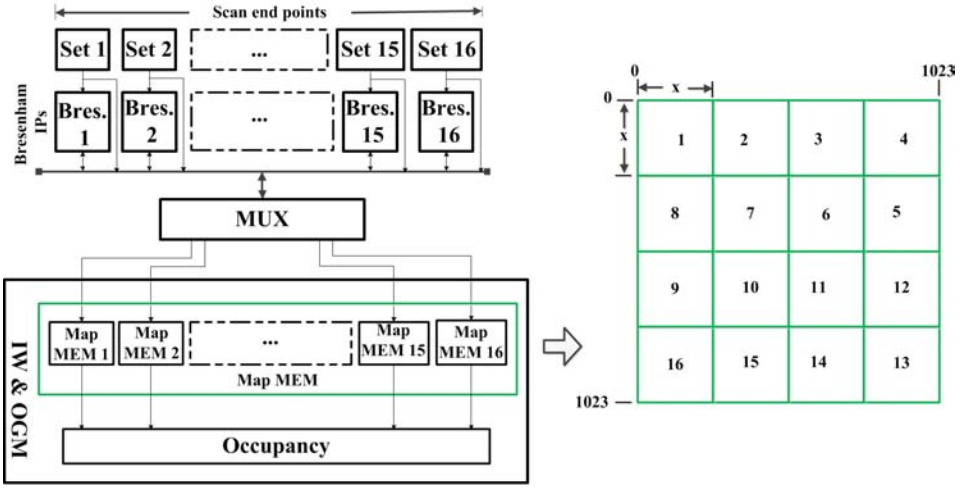


Figure 5.7: Parallel processing of laser end points with parallel Bresenham IPs (left) and partitioning of the map memory into k smaller memory modules (right)

number of clock cycles until all the corresponding points to a given map memory unit are updated before going into the generation of the index of the next grid cell.

To confirm the correct functionality of the Bresenham IP module, a plot of the trace of the grid cells along the path of the individual laser scan beams at a given time t is shown in Fig. 5.8. The data used for the verification is based on an actual laser scanner data. The figure shows the plot of the indices of all the grid cells in respect to the ground truth of a robot. The utilization of various resources of the target FPGA necessary for the LRF parallel co-processor architecture shown in Fig. 5.2 is provided in Table 5.1. The evaluation is conducted with a 32 bit fixed point number representation with 16 bit for decimal and 16 bits for fractional. The resource utilization percentage is given for the top level logic of the design (i.e. LRF parallel processor).

5.4 PF Processing Element Design

This section describes the design and implementation of the hardware architecture for the three main steps of the PF. The general block diagram of the PF processing element (PE) design is shown in Fig. 5.9. In this design the *sample* unit is responsible for generating new particle starting with a given initial particle from the *initialize* unit. While the samples are generated in the *sample* unit their respective weights

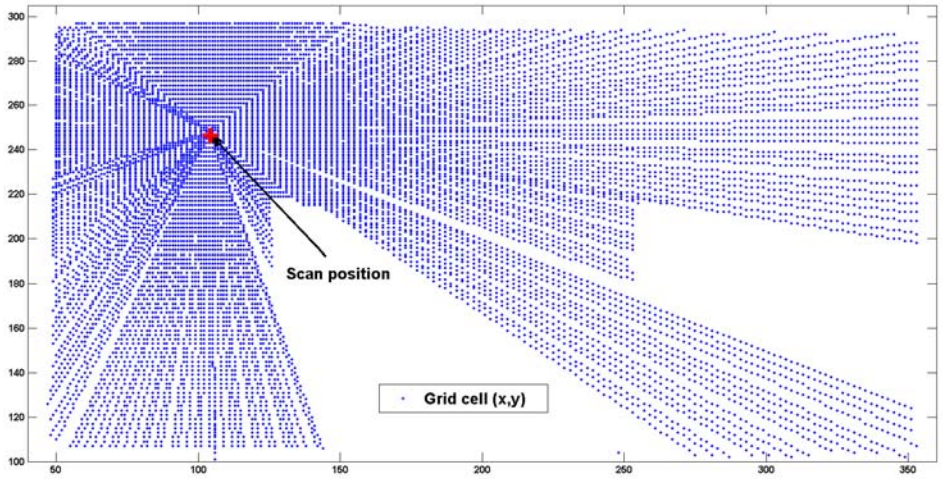


Figure 5.8: Laser scan point processed by the Bresenham IP module

are calculated by the *importance* weight unit using the available sensor measurement data. Using the weight data from the *importance* unit and a uniform random number from the random number generator (RNG) the index of the replicated particles is determined by the IMH based resampler. The particle memory (PMEM), weight memory (WMEM) and replicated particle index memory (RPI MEM) are used to store the particles, the weights and indices of the replicated particles respectively. As the PF involves many particles, one aspect in the implementation of the PF in a hardware is the correct manipulation of the particles among the PF processing steps and the memory units. With this in mind, the following subsection provide details on the hardware design of the three steps for the Grid-based Fast SLAM application.

Table 5.1: Resource unitlization for LRF parallel processor

Resources	LRF	Bresenham	Top level logic	Available	Utilization
Slice registers	2153	83	2768	407600	0%
Slice LUTs	8541	339	9255	203800	4%
DSP48E1s	52	-	60	840	7%
Block RAMs	9	-			0%

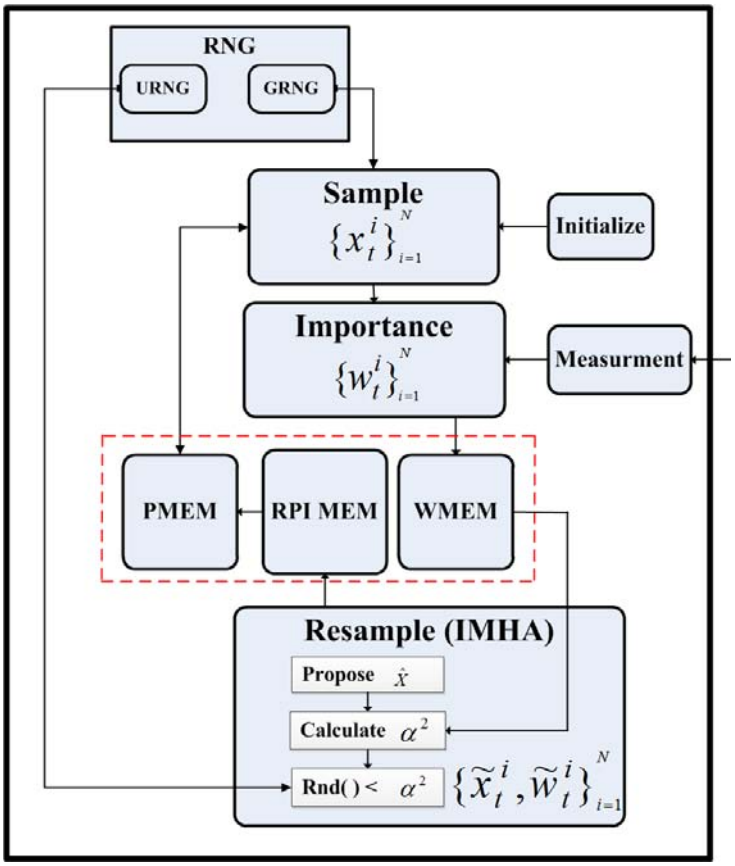


Figure 5.9: Block diagram for particle processing element (PE)

5.4.1 Sample Unit Design

The *Sample* unit is responsible for generating new set of particle instances based on previous set of particles, and their respective replication factors in the RPI MEM that is obtained from the resampling process. Before going into details of the hardware architecture for the *Sample* unit, it is important to understand how information about replication factor of particles are maintained in RPI MEM, as the *Sample* unit operation heavily depends on the manipulation of such data. Considering a total of eight particles, Fig. 5.10 shows an example of how replicated particle indexes are maintained in the RPI MEM. For illustration, in Fig. 5.10 the indexes of replicated and discarded particles are shown with solid and empty circles bullets respectively. Based on the index of the replicated particles in the RPI MEM, then the replication factor (RF) (shown on the right side of Fig. 5.10) for each particle is obtained indirectly from the RPI MEM.

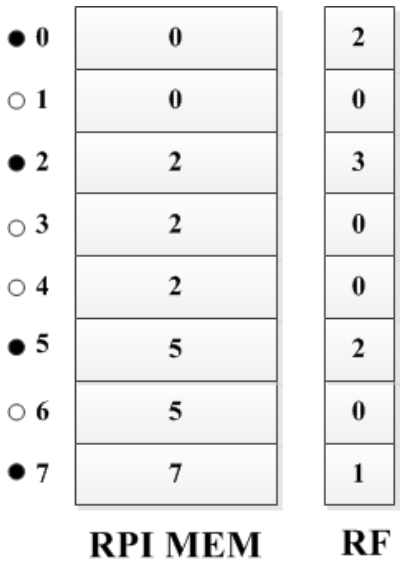


Figure 5.10: Example of particle sampling based on their replication factor in the RPI MEM

The hardware architecture for the *Sample* unit is shown in Fig. 5.11 and it is composed of different hardware sub modules. The *Motion Model* sub module, implements the Monte Carlo Localization for generating particles based on an odometry motion model of a robot explained in Chapter 2, Section 2.9.3. The inputs to this module are a Gaussian random number and odometry data. The Gaussian random numbers are generated based on a Ziggurat Gaussian random number generator (GRNG) module (chapter section). The *Sampler* sub module generates new instance of particles based on the previous particle instances from the PMEM and motion information from the *Motion Model* sub module. The *Control* module generates control signals in order to control the sampling process operations. Its design is based on a finite state machine shown in Fig. 5.12 which composed of three different states, *Initialize particles*, *Wait* and *Sampling* states. In the *Initialize particles* state, the control unit sets the write enable (*we*) signal to the PMEM to a logic level of one in order to initialize the PMEM with an initial set of particle. The write address to the PMEM is generated by the index counter (*Index Cnt*) unit. Once initialization is performed the state moves to the *Wait* state, where it waits for the start of the sampling process. The start of the sampling process is controlled by a single bit *start* signal. While this signal is asserted to a logic level of one, the finite

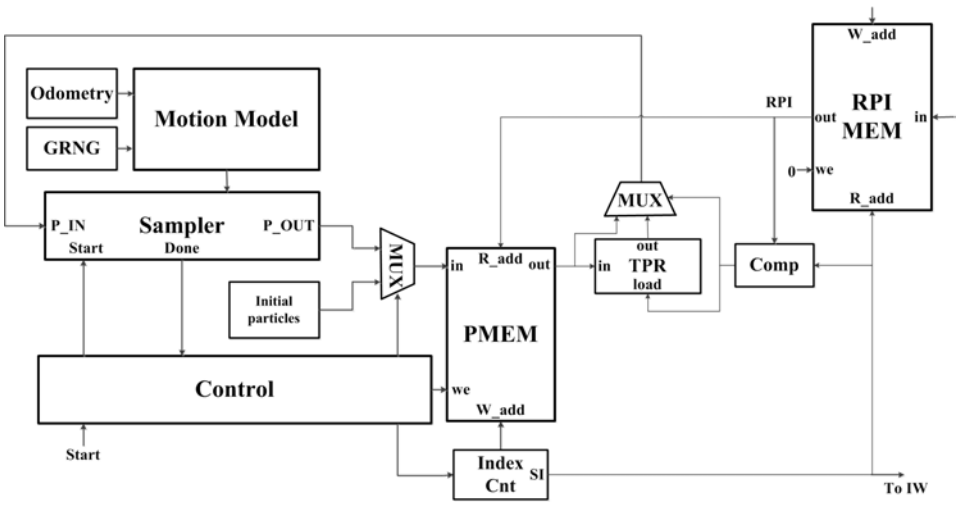


Figure 5.11: Sample unit architecture

state machine moves to the *Sampling* state and remains in the *Sampling* state until the required number of particles are generated. After all the required particles are generated, the state moves to the *Wait* state to wait for the next sampling process.

As a single memory is used for both sampled and resampled particles, avoiding the overwriting of a replicated particle in the PMEM while new particle instances

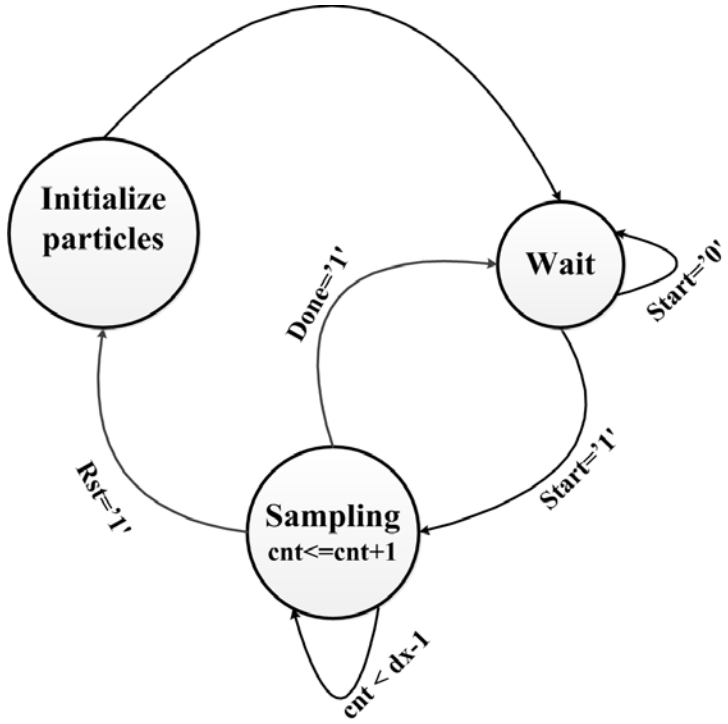


Figure 5.12: Sample control unit fsm based control sub module

are generated from previous sampled particles based on their replication factor by the *Sample* unit is critical. This task is achieved by comparing the index of a replicated particle, RPI from the replicated particle index memory (RPI MEM) and sample index (SI) from the *Index Cnt* unit using the compare (*Comp*) component shown in the *Sample* unit architecture. The result of the comparison generates a single bit signal which is used as a select and load signal to the MUX and temporary particle register (TPR) units respectively. The TPR is used to hold the value of a replicated particle until all the instances of that particle based on its replication factor are generated. The value of the TPR is updated by asserting its load signal to a logic level of one from the comp unit, while the RPI and SI have the equal value (which implies the particle at the SI index in the PMEM is a replicated particle). On the other hand, if RPI not equal to SI then a particle at the SI memory address in the PMEM is a discarded particle, which has to be replaced by a replicated particle instance.

In summary, for replicated particles with a replication factor of one, new instance of the particle is generated and written to the SI memory address in the PMEM. For particles with replication factor of greater than one, first the replicated particle is loaded to the TPR and its new instances are generated by directly using the TPR data (i.e. a single read operation is performed for the replicated particles from the PMEM). The proposed approach helps to avoid the extra requirement of storing the replication factor in another memory [2] and avoids the multiple read operation from PMEM for the replicated particle instances.

5.4.2 Importance Weight Unit Design

The importance weight unit performs the evaluation of the weight of the particles based on the most recent measurement data. For the specific application adopted in this work (SLAM), as the measurement data is obtained from laser range finder, the hardware implementation of the importance weight unit is based on the application of the parallel LRF processor design mentioned in section 5.3.

The hardware architecture of the importance weight unit is shown in Fig. 5.13, where it composed of the *Weight Update*, *Map Update*, weight memory (*W MEM*) and map memory (*Map MEM*) hardware sub modules. The *W MEM* and *Map MEM* modules are used to store the weights of the particles and the occupancy grid map of the robot environment respectively. In the architecture, only few control and data ports are shown which are used for synchronization and data exchange between the importance weight (*IW*) and occupancy grid map (*OGM*) modules.

The *Weight Update* hardware sub module calculates the weight of each particle indexed by the particle index (P_Index) from the *Sample* unit based on the laser end points in the global map coordinate from the parallel LRF processor unit. The *Weight Update* unit starts its calculation while its done input from the parallel LRF process is asserted to a logic one level, indicating that the data from the parallel LRF process is ready. The Laser End points are used to access the occupancy value of the grid cells at the end points of the current laser scan from the occupancy map memory (Map MEM).

The weight w^i of each particle is calculated by scan matching the current laser scan points and the occupied points in the global map as given by equation

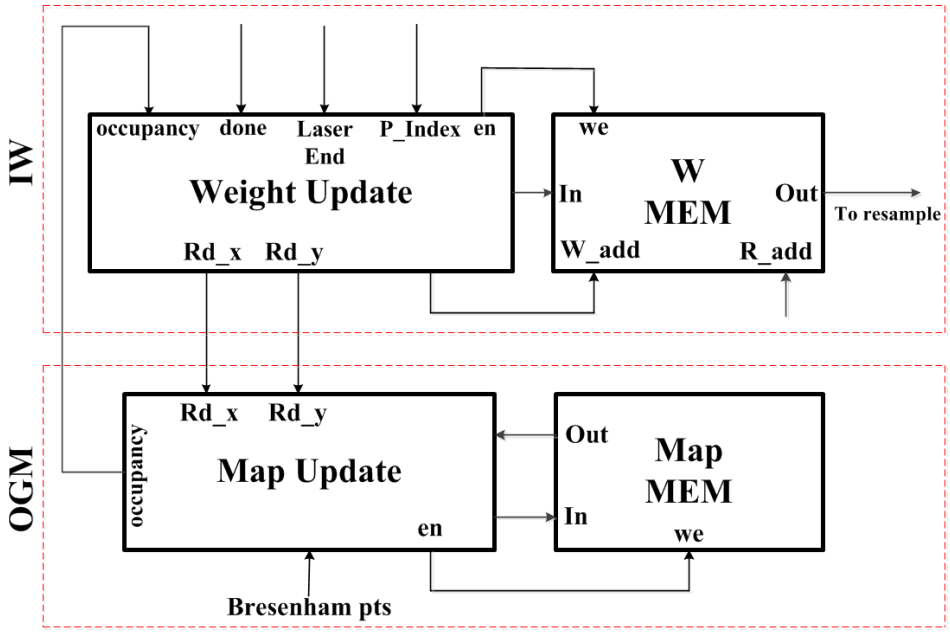


Figure 5.13: Importance weight unit architecture

5.1. Current laser scan points which fall in unoccupied or unexplored grid cells contribute a small values to the weight of the particles compared to the occupied points. Therefore, the weight of a particle is maximized while the current laser scan points match well with the occupied points in the map.

$$W^i_{Z^n_t} = \sum_{n=1}^{L} 255 - p(m_{t-1}|z^n_t) \tag{5.1}$$

Once the weight of all the particles is calculated, the global occupancy map is updated with the particle that has the highest weight. Each grid cell of the occupancy grid map corresponds to the probability of the grid cell being free (which is represented by a value of 255). For an occupied grid cell the occupancy value is represented by a value of 0 and for complete uncertainty over the occupancy of a grid cell a value of 127 is used. The map update is performed by tracing all the grid cells which lies along the path of the laser beams and updating their corresponding occupancy likelihood. However, the occupancy likelihood of those grid cells that are out side the laser's beam range of view remain unchanged. The update of the occupancy of the grid cells along the path of the laser beam is performed by using the Bresenham points from the parallel LRF processor unit in order to read and update their occupancy likelihood accordingly. And those grid cells that lies between the staring point and end point of the laser beam corresponds to unoccupied points and their occupancy likelihoods are updated by a certain factor f_{free} and for the grid cell that corresponds to the end point of the laser beam, its occupancy likelihood is updated by incrementing its previous value by a factor of $f_{occupied}$. The value $f_{free} = 0.1$ and $f_{occupied} = 0.1$ are used in this work, which are determined by heuristic tests.

5.4.3 Resample Unit Design

The hardware architecture of the resampling unit shown in Fig. 5.14 implements the IMHA algorithm given in Chapter 2, Section 2.8.5. The hardware implementation is based on four individual hardware sub modules; the *Particle index generator*, *Markov Chain Register*, replicated particles index memory (RPI MEM) and finite

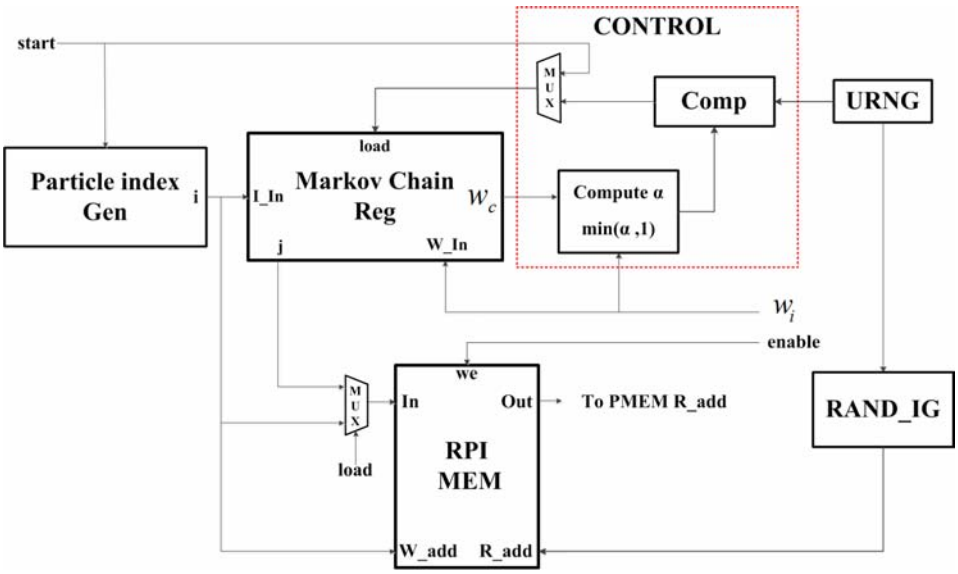


Figure 5.14: Resample unit architecture

state machine based *CONTROL* logic. The particle index generator generates the index of the particles, the *Markov Chain Register* is a register for temporary storage of the index and weight of the last accepted particle, the RPI MEM is used to store the indexes of the replicated particles and, finally the *CONTROL* unit performs the evaluation of the acceptance probability and generates appropriate control signal for the *Markov Chain Register* module.

The inputs to the resampling units are the weight of a particle, a single bit *enable* and *start* signals. The *enable* bit signal is used to control a read and write process to the replicated particle index memory unit, and it is set to logic level of one during the resampling operation to perform a write operation to the memory. The *start* signal is used to indicate the starting of the resampling operation and the initialization of the Markov chain with first particle index (i.e. $i = 0$). The initialization of the Markov chain with the first particle index is achieved by first initializing the particle index generator to zero and loading the corresponding index and weight of the particle data to the current chain register unit. For loading of the particle index and weight to the current chain register a single bit *load* signal is used by the current chain register. This *load* signal is generated by the *CONTROL* logic unit, where it is initially asserted to the value of the *start* signal by the MUX in the *CONTROL* unit.

Once initialization of the Markov chain is performed, the decision whether to accept or reject a new particle based on its weight data is performed by the *CONTROL* logic unit. The *CONTROL* logic unit uses the new and last accepted particle weights, w_i and w_c respectively, to evaluate the acceptance probability (α) value. The last accepted particle corresponds to the current chain of the Markov from the output of current chain register unit. The result of a comparison between the acceptance probability and a uniform random number is used to accepted the new particle or replicate the last accepted particle once more. The uniform random number is generated by the uniform random number generator (URNG) unit, which is implemented based on the Tausworthe uniform random number generator algorithm (see Chapter 3 of Section 3.5.3).

If the new particle is accepted, its corresponding index (i) and weight (w_i) are loaded to the current chain register by asserting the load signal to a logic level of 1. Simultaneously, the index of the accepted particle is written to the RPI MEM at the i^{th} memory address. If the new particle is rejected, the index of the last accepted particle (j) from current chain register is written to the RPI MEM at the i^{th} memory index.

After the resampling operation is done, the RPI MEM contains a total of $(N + N_b)$ indexes of the replicated particles that are going to be used in generating new particle instances by the sampling unit during the next time step of the PF operation. During the sampling step, the RPI MEM is read randomly starting from the memory address of N_b to $N + N_b - 1$ by generating a uniform random integer from the range N_b to $N + N_b - 1$. This is achieved by using the uniform random integer generator module (URAND IG), which implements the equation $I = U * (N - 1) + N_b$ to generate N uniform random integers within the range of $N_b \leq I \leq N + N_b - 1$. The values read from RPI MEM are used to fetch the state of the corresponding particles in the PMEM which are updated as per the state equation in the *Sample* unit.

5.5 Implementation and Results

In this section the results of the hardware implementation of the PF processing element (PE) and its corresponding sub-modules for the Fast SLAM application on a Xilinx Kintex-7 KC705 FPGA device are presented. The proposed system is tested on real robot data collected from a mobile robot platform. In the hardware implementation the output values of the state estimates from the PE are 32-bits wide in fixed point format. These output values were first captured for validating the system using the corresponding 32 bit floating values and plotting the results in Matlab.

5.5.1 Resource Utilization

The hardware architectures are described in VHDL. A total of $N = 1024$ particles and a grid map of size 1024×1024 grid cells with a resolution of 0.02 $meters/map\,cells$ is used in the evaluation. All the memory modules required in the PE are realized using block RAMs, and all the necessary trigonometric and others functions are implemented using a pipelined CORDIC module, and other arithmetic operations are implemented using DSP cores. The Gaussian random numbers required in the sampling step in accordance with the state space model and the uniform random numbers needed in the resampling step as per the IMHA algorithm are generated using the random number generator given in Chapter 3 of Section 3.5.3.

As the number of particles N is a critical parameter which impacts the hardware resource and estimation accuracy, a study for the optimal number of particles is determined. Fig. 5.15 and 5.16 shows the effect of the number of particles N on the RMSE and FPGA hardware resource utilization respectively. Considering the results of Fig. 5.15 and 5.16, a value of $N = 1024$ is considered as the optimal value of the particles. Table 5.2 summarizes the distribution of resources on the target platform among the various modules of the PE. The importance weight module consumes large amount of the total slices as it contains most of the logic, and most of the Block RAMs in this module are used to store the map data.

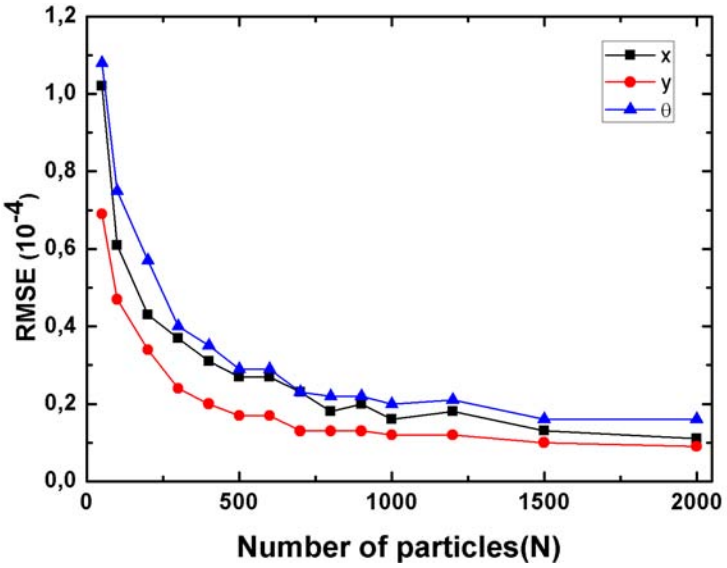
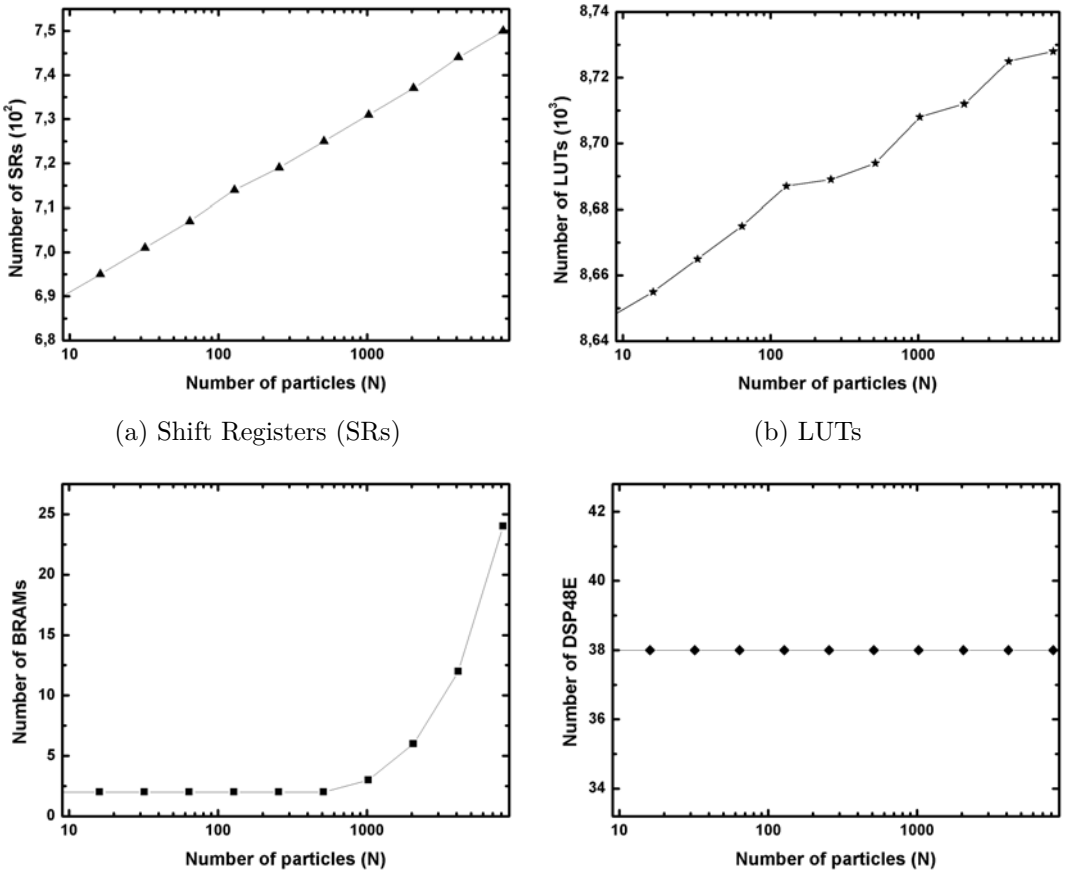


Figure 5.15: Number of particles (N) vs RMSE



(a) Shift Registers (SRs)

(b) LUTs

(c) BRAMs

(d) DSP48E

Figure 5.16: FPGA HW resource utilization with varying number of particles. The number of particles is shown in log scale

Table 5.2: Resource utilization on Xilinx Kintex-7 KC705 FPGA device

Modules		Resources							
Top Level	Sub Module	Slice Regs.		Slice LUTs		DSP48E1s		BRAMs	
Sampling	OMM	327	5	7312	4185	38	16	3	-
	SMM		96		332		16		-
	SU		-		2640		6		-
I. Weight	WU	848	58	11770	136	2	-	258	2
	MU		9		163		48		16
Resampling	-	32		20		-		1	
PE	-	1462		19116		86		260	

5.5.2 Execution Time

The execution time of the PF to the Fast SLAM application is shown in Fig. 5.17, where L_S, L_{IW} and L_R are the startup latencies in the sampling, importance weight, and resampling modules respectively. T_M is the number of clock cycles required for map update. The total number of clock cycles for one recursion of the PF is then given by $T_{PF} = L_S + L_{IW} + N + L_R + T_M$.

As the implementation of the sampling unit is based on a pipelined CORDIC for the realization of the different trigonometric functions and a Gaussian random number generator which generates a Gaussian random number every clock cycle,

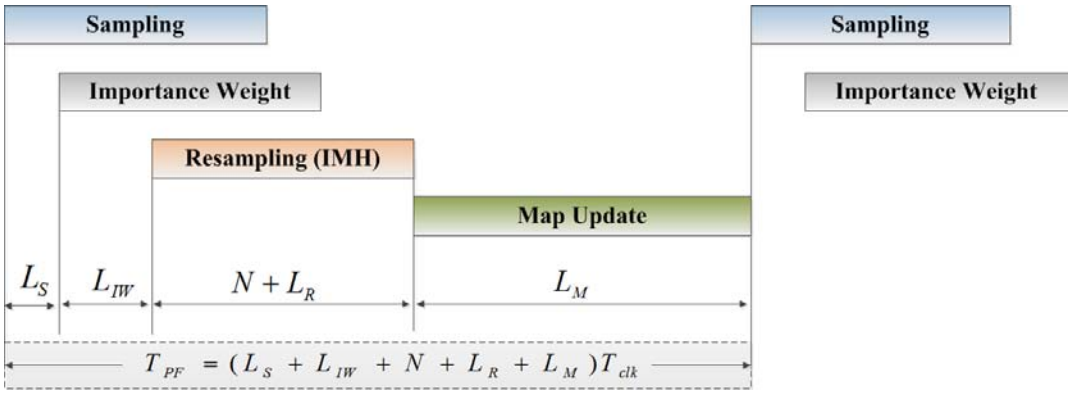


Figure 5.17: Execution time of the PF

its start up latency $L_S = 1$ clock cycle. For the importance weight unit, as the evaluation of the weight of a particle is performed based on the scan matching of the latest laser scan data (which consists of M laser beams) with the map of the robot from previous time t its startup latency is dependent on the number of the laser scan beams M.

For the implementation, the M laser scan beams are distributed among $n = 16$ parallel Bresenham IPs, which results in the processing of $B_n = M/16$ scans beams in the individual 16 parallel Bresenham IPs. As a result the latency in the importance weight is $L_{IW} = B_n$. The start-up latency of the resampling unit is $L_R = 1$ clock cycle. The map update step have a latency of $L_M = (B_n \times range\_max)/map\_resolution$, where $range\_max$ is the maximum LRF measurement distance and $map\_resolution = 0.02\, meters/map\, cells$ is the resolution of the map. The proposed system is evaluated on a real data set based on a Sick PLS LRF for the CMU-Newell-Simon Hall building available at [3] which has a total of $M = 180$ scans and a maximum range of $range\_max = 8.0 meters$.

Considering the PLS LRF data, the total number of clock cycles (T_{PF}) required for a complete iteration in hardware equals 5537 clock cycles. Using the FPGA system clock frequency of 100 MHz, the speed at which new particles can be processed is given by $100MHz/T_{PF}$. Therefore, the throughput for the implementation based on the Sick PLS LRF is $18.06kHz$. From the post place and route timing analysis report, the maximum clock frequency of the PE and its sub-modules is provided in Table 5.3. As shown in Table 5.3, the design can support clock frequency of upto 304.174 MHz.

The variations in system and measurement models, variation in sensor inputs and memory requirements for different applications of the PF, makes difficult the direct comparison of the achieved results to other PF hardware implementations and taking this into account, Table 5.4 provides a comparison on the performance results achieved for various applications. In Table 5.4, the only complete FPGA implementation of SLAM in [4] has achieved a maximum clock frequency of 143 MHz. However, this implementation is not based on PF instead on a genetic algorithm. In

Table 5.3: Maximum frequency of operation for PF computational modules

PE sub-module	Maximum frequency (MHz)
Sampling	**539.476**
OMM	-
SMM	1426.228
SU	-
Importance weight	**308.482**
WU	743.505
MU	-
Resampling	534.574
PE	**304.174**

addition, among a total of 180 laser scans based on Sick LMS-200 LRF, by using only 20 of the scans the author suggested the speedup of the particle weight computations at the cost of discarding useful measurement data. In our approach laser scans are partitioned among 16 parallel Bresenham IPs, which enables the fast processing of the whole sensor data. The relatively high throughput of 146 kHz in reference [5] results from the fact that a total of 4 parallel PEs are used instead of the single PE as in our case, furthermore, the system model for the specific application are not as complex as that of the SLAM algorithm and does not involve the processing of large sensor data and memory read / write operations as in SLAM. In general the relatively high throughput in [5–8] are due to the parallel PE implementation compared to a single PE realization, the simplicity of the computations, the non intensive sensor data processing and memory read / write operation compared to the SLAM problem. The lower throughput in our implementation resulted primarily from the high latency in the map update procedures as it involves intense memory read / write operations which limits the level of parallelism. The map update requires $L_M = 4500$ clock cycles out of the total 5537 clock cycles of the whole

Table 5.4: Comparison with other implementations

Reference	Application	FPGA Device	No. Particles	No. PE	Max. Frequency (MHz)	Throughput (kHz) (System clock frequency)
[2]	BOT	Virtex II (XC2VP50FF1152)	2048	1	118	16 (100 MHz)
[5]	Wave-agile radar tracking	Virtex 5 (XC5VSX240T)	1000	4	-	146 (100 MHz)
[6]	Maneuvering target tracking	Virtex II (XC2VP70FF1517)	1000	2	70	50 (60 MHz)
[7]	Maneuvering target tracking	Virtex II (XC2V4000)	1000	2	60	20 (not given)
[4]	SMG-SLAM	Virtex 5 (XC5VFX70T)	-	1	143.394	0.623 (100 MHz)
[8]	Tracking neural activity	Virtex 5 (XC5VSX240k)	3200	4	-	20.61 (100 MHz)
Proposed	SLAM	Kintex7 (XC7K325T)	1024	1	304.174	18.06 (**96.43**)

implementation. For fair comparison, neglecting the map update step, our implementation requires only 1037 clock cycles to perform the sampling, importance weight and resampling steps. This results in a throughput of $96.43kHz$, which is higher than even most of the parallel PF implementations given in Table 5.4. In summary, the proposed system obtains comparable performance with significantly reduced computational complexity for the SLAM problem and enables real time processing.

5.5.3 Estimation Performance

The state of the PF for the Fast SLAM algorithm is 4-dimensional due to the fact that it incorporates the pose (x, y, θ) of the robot and its map m. The input to the filter are the odometry and the laser scan data, where each input sample is processed by the PE module to produce an estimate of the pose and map at that sampling instance. The accuracy of the state estimate is measured by the RMSE error metrics between the true state and the state estimate.

The evolution of the RMSE over time for the pose of the robot is shown in Fig. 5.18, where the RMSE error is maintained at low levels through out the time. The performance of the system for the map estimation is evaluated qualitatively by comparison with a software generated map. The maps obtained from the software

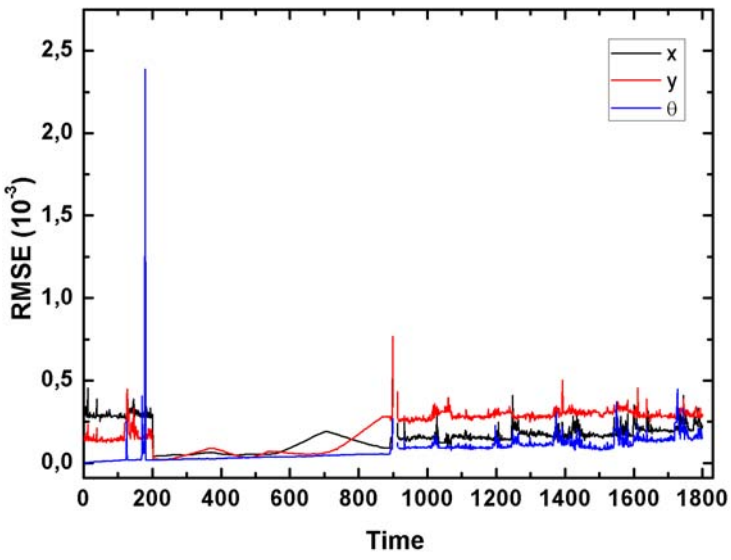
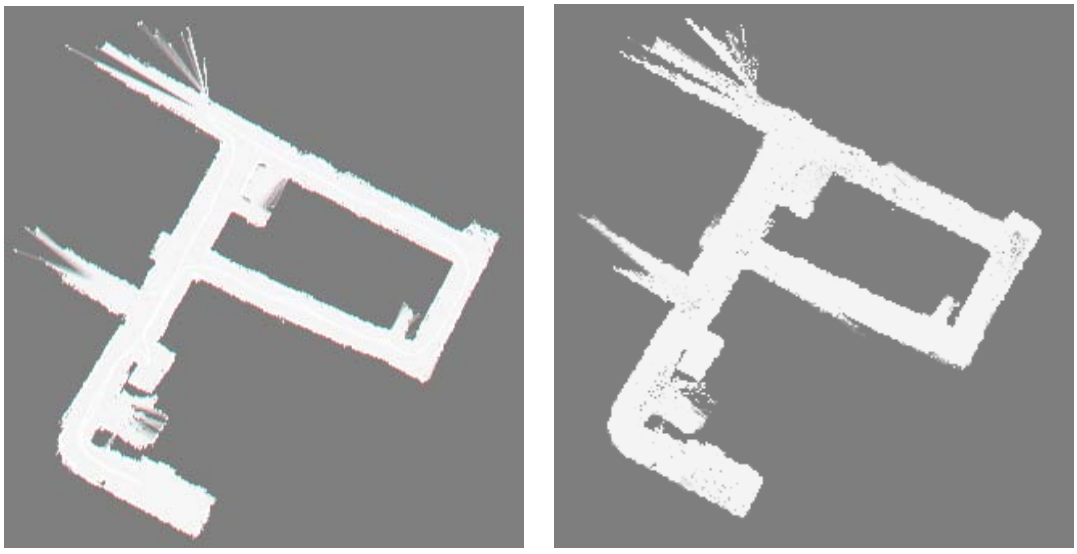


Figure 5.18: RMSE for robot pose over time for N=1024

and hardware implementations are shown in Fig. 5.19 (a) and (b) respectively. As per the results of Fig. 5.19 the quality of the SLAM solution is similar in the software and hardware implementations. The slight difference in the shape of the map between the software and hardware implementation is due to the difference in random nature of the algorithm.

In conclusion, the proposed hardware architecture has led to the development of the first hardware (FPGA) prototype for the PF applied to the SLAM problem. The proposed system is validated with an implementation on Xilinx Kintex-7 KC705 FPGA device, where 18.06 kHz throughput has been achieved and this would enables the real-time applicability of the system for different robotic applications. Taking into account the difficulty of a direct comparison of the proposed system implementation with other FPGA based implementations due to differences in models and number of particles, the proposed system obtains comparable performance results with a relatively lower resource utilization and significant reduction of computational complexity for the SLAM problem. For real-time processing of SLAM applications, the proposed implementation has an execution time of $55.37\mu s$.



(a) Software (b) Hardware

Figure 5.19: Software and Hardware generated map for CMUNewellSimonHall data set

References

[1] J. E. Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems journal* 4.1 (1965), pp. 25–30.

[2] A. Athalye, M. Bolić, S. Hong, and P. M. Djurić. "Generic hardware architectures for sampling and resampling in particle filters". In: *EURASIP Journal on Advances in Signal Processing* 2005.17 (2005), pp. 1–15.

[3] `http://radish.sourceforge.net/index.php`.

[4] G. Mingas, E. Tsardoulias, and L. Petrou. "An FPGA implementation of the SMG-SLAM algorithm". In: *Microprocessors and Microsystems* 36.3 (2012), pp. 190–204.

[5] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola. "Algorithm and parallel implementation of particle filtering and its use in waveform-agile sensing". In: *Journal of Signal Processing Systems* 65.2 (2011), pp. 211–227.

[6] S. Hong, Z. Shi, and K. Chen. "Easy-hardware-implementation MMPF for maneuvering target tracking: algorithm and architecture". In: *Journal of Signal Processing Systems* 61.3 (2010), pp. 259–269.

[7] A. Athalye, S. Hong, and P. M. Djuric. "Distributed architecture and interconnection scheme for multiple model particle filters". In: *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*. Vol. 3. IEEE. 2006, pp. III–III.

[8] L. Miao, J. J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola. "Efficient Bayesian tracking of multiple sources of neural activity: Algorithms and real-time FPGA implementation". In: *Signal Processing, IEEE Transactions on* 61.3 (2013), pp. 633–647.

6
Conclusions and Future Work

In this Chapter, first the summary and the contributions of this thesis are presented followed by the future research directions in the acceleration of the PFs computations.

6.1 Summary and Contributions

Real-time estimation of the state of dynamic systems is extremely important in different application areas. The most widely known tools for the state estimation of dynamic systems with nonlinear and non-Gaussian nature are the particle filters (PFs). PFs are sequential Monte Carlo estimation methods which operates by representing the posterior density by a set of weighted samples (particles). Compared to traditional filtering methods, such as Kalman filters, PFs offer superior performance in several important practical problems. However such an excellent performance of PFs comes at the expense of huge computational cost which inhibits their wide range applicability. As a result, it is required to accelerate the intensive computations involved in PFs through hardware to achieve real-time computations. This PhD thesis has taken a significant step in enabling this by developing efficient hardware for real-time PF in navigation application.

The contributions of this thesis can be classified into two major parts. The first part relates to the review on recent developments behind the theory and

implementations of PFs, the analysis on the computational challenges involved in the different step of the PF-SLAM and the proposal of generic PF acceleration techniques. This includes contributions from Chapters 1, 2 and, mainly 3. The second part relates to the three approaches for the PF-SLAM FPGA implementations. This includes contributions from Chapters 4 on the HW / SW co-design and Parallel HW / SW co-design FPGA implementations. The third approach, HW PF processing element (PE) design, is presented in Chapter 5. The efforts and specific contributions towards achieving the objective of developing real-time PF with the above three approaches can be summarized as follows.

- The main challenges and limitations in the use of PFs in complex real-time applications are discussed and studied. Taking into account our objective of real-time PF in navigation applications, a particular attention has been given to the SLAM technique.

- The study on the computational complexities and performances for two well known PFs (SIR and RPF) are conducted by considering their *sampling*, *importance weight* and the *resampling* steps. The RPF showed a relatively higher complexity to the SIR PF. As the resampling step is crucial in PFs hardware implementations, different available resampling techniques are studied. Among the resampling methods considered, Independent Metropolis Hastings Algorithm (IMHA) resampling resulted in the lowest computational complexity with a bottleneck free operation.

- PF hardware acceleration techniques are presented in order to speed up the underlying intensive computations involved in all the steps of the PF-SLAM. The proposed acceleration techniques include the CORDIC, Tausworthe and Ziggurat methods for evaluating computational intensive mathematical functions and random numbers (uniform and Gaussian) generation respectively. The presented random number generator resulted in the highest throughput with comparable resource utilization compared with other reported hardware implementations.

- PF-SLAM FPGA HW / SW co-design: Based on the PF hardware acceleration techniques, a PF hardware acceleration core is designed. Using the PF hardware acceleration core and soft-core MicroBlaze processor, a HW/SW PF-SLAM system is proposed. The presented HW/SW PF-SLAM system leads to an improvement in the speedup of 140×, 14.87× and 19.36× in the sampling, importance weight and resampling steps respectively. Due to the flexibility in the HW/SW co-design and the presented PF acceleration techniques, the same approach can be easily adopted to other real-time particle filtering applications. The performance the HW/SW PF-SLAM system is evaluated on standard and low cost laser scanner sensors considering that the input to the system comes mainly from such sensors.

- PF-SLAM FPGA Parallel HW / SW co-design: In the acceleration of PFs computations through parallelism, the sequential nature of PFs resampling step introduces a bottleneck and limits PFs full parallelization. To address this challenge, we present a new approach for full parallelization of the PFs with the application of a Metropolis coupled Markov Chain Monte Carlo $(MC)^3$ approach. The $(MC)^3$ approach reduces the high communication data flux which severely degrades the performance of traditional parallel PF implementations. Using the $(MC)^3$ approach parallel PF architecture is proposed. The proposed parallel PF resulted in a distributed resampling with minimum communication bottleneck and without the requirement of interconnect network among the parallel particle processors for routing particles. It also avoids the high communication costs by exchange of few data among the parallel particle processors through a central processing unit. This leads to a distributed particle filtering implementation with simplified parallel architecture. The results obtained in this study shows a speedup improvement of 3.97× in respect to a HW / SW approach. The improvement is of 72.28× when compared to a totally software implementation.

- PF-SLAM FPGA PE design: To avoid the limitations imposed on the level of parallelism for further speedup by available bus interfaces in the soft-core processor (Microblaze), generic hardware architecture by applying the PF hardware acceleration techniques are proposed for the whole PF-SLAM steps. The proposed PF-SLAM PE architecture can easily be extended to other PF applications as our design approach focuses primarily on the high-level data and control flow. The only design effort required is for designing the specific application model dependent computational units.

  As part of the PF PE design, important problems for PF in SLAM are also solved. For example, the design and implementation of a parallel laser scanner co-processor based on Bresenham line drawing algorithm has been realized. The proposed hardware architecture has led to the development of the first fully hardware prototype for the PF applied to the SLAM problem. The proposed system is validated with an implementation on Xilinx Kintex-7 KC705 FPGA device, where an execution time of 55.37μs has been achieved. This would enables the real time processing for SLAM application and applicability of the system for different robotic applications. Table 6.1provides the comparison on the execution times for the PF-SLAM realization with the three approaches.

Table 6.1: Performances Comparisons on Proposed Approaches for PF-SLAM FPGA Implementation

Approaches	HW / SW	Parallel HW / SW	HW PE
Number of Particles	100	100	1024
Execution Time (ms)	437.93	110.31	0.05537

6.2 Future Work

There are several topics that still have to be investigated and that can open new research lines. Here, the directions for this future research are presented briefly, and can be considered extension to the research performed in this thesis.

- Depending on the PFs application and the specific application models, the computational bottlenecks in PFs may vary. Future work would be the analysis of different types of PFs from the point of view of the computational complexity of the target model and its impact on the various PF steps. The main focus should be then in identifying and resolving these bottlenecks in the algorithm and architecture levels by applying the different PF hardware acceleration techniques presented in this thesis.

- Considering the fact that PF hardware acceleration techniques can be easily extended to various applications of the PFs, an interesting research direction is to automate the PFs FPGA design and implementation methodologies. This would make the design process convenient and consequently helps in the widespread use of PFs for complex real-time applications.

- The use of dynamic partial reconfiguration for the design and implementation of adaptive and variable structure of PFs. This could be useful for system where the model parameters are required to change dynamically. The application of such PF structures to practical problems and development of methods to control the adaptation in the PFs structure is a promising direction for the future to make the PFs very powerful and enable them to be applied to most complex systems.

- Parallelization of the PF PE by applying the $(MC)^3$ approach and evaluate its performance on practical real-time systems is required.

- It could also be interesting to combine the PFs with the other types of filters such as extended Kalman filter and compare the performance of the respective implemented design on FPGA. Due to their internal construction FPGAs can still accommodate complex filter structures for real-time applications.