ARQUITETURA EM *HARDWARE* DO FILTRO DE KALMAN ESTENDIDO PARA LOCALIZAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS IMPLEMENTADA EM FPGA

LUIS FEDERICO CONTRERAS SAMAME

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

ARQUITETURA EM *HARDWARE* DO FILTRO DE KALMAN ESTENDIDO PARA LOCALIZAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS IMPLEMENTADA EM FPGA

LUIS FEDERICO CONTRERAS SAMAME

Orientador: Carlos Humberto Llanos Quintero

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

Publicação: ENM.DM-86/15

BRASÍLIA-DF, 27 de Março de 2015

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

ARQUITETURA EM *HARDWARE* DO FILTRO DE KALMAN ESTENDIDO PARA LOCALIZAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS IMPLEMENTADA EM FPGA

LUIS FEDERICO CONTRERAS SAMAME

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

BANCA EXAMINADORA

APROVADA POR:

Prof. Carlos Humberto Llanos Quintero, Dr. (ENM-UnB)
(Orientador)

Prof. Daniel Mauricio Muñoz Arboleda, Dr. (ENM-UnB)
(Examinador interno)

Prof. Antonio Padilha Lanari Bo, Dr. (ENE-UnB)
(Examinador externo)

BRASÍLIA-DF, 27 de Março de 2015

FICHA CATALOGRÁFICA

CONTRERAS, L.F.
Arquitetura em *hardware* do Filtro de Kalman Estendido para localização de robôs móveis autônomos implementada em FPGA [Distrito Federal] 2015.
xiv, 99p. 210 × 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2015). Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.
Departamento de Engenharia Mecânica.

1. Localização
2. FPGAs
3. Robôs móveis
4. Filtro de Kalman
I. ENM/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

CONTRERAS, LUIS. (2015). Arquitetura em *hardware* do Filtro de Kalman Estendido para localização de robôs móveis autônomos implementada em FPGA. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-86/15, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 99p.

CESSÃO DE DIREITOS

AUTOR: Luis Federico Contreras Samame.

TÍTULO: ARQUITETURA EM *HARDWARE* DO FILTRO DE KALMAN ESTENDIDO PARA LOCALIZAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS IMPLEMENTADA EM FPGA.

GRAU: Mestre **ANO**: 2015

Luis Federico Contreras Samame
UnB, CEU-Colina Bloco K, Apartamento 105
70.904-111 Brasília-DF-Brasil

Dedicatória

Dedico este trabalho com amor e enorme gratidão
a Deus e a meus pais Dilma Samame e Federico Contreras,
presentes em todos os momentos comigo.

LUIS FEDERICO CONTRERAS SAMAME

Agradecimentos

Antes de tudo, quero agradecer a Deus pelas bênçãos derramadas em mim ao longo do período deste mestrado, permitindo superar as dificuldades apresentadas no caminho e alcançar os objetivos desejados.

Agradeço a toda a minha família, especialmente a meus pais Dilma e Federico e irmãos Janet, Victor Hugo e Álvaro por seu constante apoio e carinho ao longo da vida.

Ao Prof. Dr. Carlos Llanos, pela oportunidade, confiança e orientação durante todo este período de mestrado, assim como pela amizade oferecida, sendo mais que um mentor para mim.

À minha namorada Stephany por seu apoio e dedicação a mim, assim como a meus amigos Eder, Hernando, José, Aramiz, Mayco e Rodolfo pela amizade e disposição em ajudar sempre que necessário, tornando-se uma segunda família durante todo este tempo.

Aos companheiros, professores e funcionários do Programa de Pós-Graduação em Sistemas Mecatrônicos (PPMEC-UnB), especialmente a Sérgio Cruz e ao Prof. José M. Motta pela amizade e suas contribuições na realização deste trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Diretoria Desenvolvimento Social (DDS-UnB) e PPMEC-UnB pela auxilio financeiro.

LUIS FEDERICO CONTRERAS SAMAME

"Obstáculos são aquelas coisas assustadoras
que vemos quando desviamos o foco do nosso objetivo."

HENRY FORD

RESUMO

Este trabalho apresenta uma arquitetura em *hardware* para a implementação de um algoritmo probabilístico, especificamente o Filtro de Kalman Estendido (EKF) em versão sequencial, aplicado ao problema de localização em robótica móvel. Primeiro, desenvolveu-se um módulo de *hardware* para etapa de predição do algoritmo EKF baseada em um modelo odométrico de um robô móvel de quatro rodas deslizantes (4-SSMR). Logo, considerou-se um módulo de *hardware* para etapa de estimação do EKF baseada em um modelo de sistema medição usando um sensor LRF (do inglês *Laser RangeFinder*). Adicionalmente, um Módulo de *Hardware* Unificado (MHU) para o EKF foi projetado considerando as duas etapas do filtro (predição e estimação) em uma mesma arquitetura. Unidades em Ponto Flutuante (UPFs) foram usadas para operações aritméticas e trigonométricas necessárias para cada uma das equações do EKF. Para este caso, duas abordagens (módulos individuais e MHU) foram consideradas para a implementação do algoritmo EKF em um *kit* de desenvolvimento DE2-115da Altera (FPGA Cyclone IV, processador Nios II), aplicado à localização de uma plataforma móvel Pioneer 3AT (da companhia Mobile Robots Inc.). Finalmente, foram obtidas métricas (tempo de execução, consumo de potência e de recursos no FPGA) e comparações com outras soluções, a fim de validar o desempenho do sistema proposto e sua aplicabilidade para a área de robótica móvel. Entre os principais resultados, um tempo de execução da arquitetura em *hardware* do EKF de 3,08 μs foi obtido com um fator de aceleração minimo de 63 comparado com outras implementações em *software*.

ABSTRACT

This manuscript presents a hardware architecture to implement a probabilistic algorithm, specifically the *Extended Kalman Filter* (EKF) in a sequential version, applied to the localization problem in mobile robotics. Firstly, a hardware module for the EKF prediction stage was developed based on an odometric model of a 4-SSMR (Four Wheeled Skid-Steer Mobile Robot). Then, a hardware module for the EKF estimation stage was designed based on a measurement system model, using a LRF sensor (Laser Rangefinder). Furthermore, a Unified Hardware Module (MHU) for the EKF was designed taking into account the two EKF stages (prediction and estimation) in the same architecture. Floating-Point Units (UPFs) were used for arithmetic and trigonometric operations required for each of the EKF equations. In this case, two approaches (individuals modules and MHU) were considered for the implementation of the EKF algorithm over an Altera DE2-115 board (Cyclone IV FPGA with a Nios II processor), applied to the localization of the Pioneer 3AT robot (from Mobile Robots Inc.). Finally, metrics (execution time, FPGA resources and power consumption) and comparisons have been obtained, in order to evaluate the performance and suitability of the proposed system for the mobile robots area. Among the main results, an execution time of the *hardware* architecture for EKF of 3,08 μs was achieved with a minimum speedup factor of 63 compared to other *software* implementations.

Sumário

Lista de Figuras

Lista de Tabelas

LISTA DE SÍMBOLOS

ASIC	Application Specific Integrated Circuits
BME	Banco das Matrizes de Entrada
BMI	Banco das Matrizes Intermediárias
BMS	Banco das Matrizes de Saída
BRAM	Block RAM
CB	Connection Block
CLB	Configurable Logic Block
CORDIC	COordinate Rotation DIgital Computer
CPU	Central Processing Unit
DGPS	Differential Global Positioning System
DMA	Direct Memory Access
DSP	Digital Signal Processor
EKF	Extended Kalman Filter
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPP	General Purpose Processor
GPS	Global Positioning System
GPU	Graphics Processing Unit
HW/SW	Hardware/Software
IOB	Input Output Block
KF	Kalman Filter
LE	Logic Element
LEIA	Laboratory of Embedded Systems and Integrated Circuits Applications
LPM	Library Parameterized Modules
LRF	Laser RangeFinder
MCL	Monte Carlo Localization

MHU	Módulo de Hardware Unificado
MPSoC	Multiprocessor System on Chip
MSB	Most Significant Bit
NRE	Non-Recurring Engineering
PC	Personal Computer
PE	Processing Element
RAM	Random-Access Memory
ROM	Read-Only Memory
SB	Switching Block
SCG	Sistema de Coordenadas Global
SCR	Sistema de Coordenadas do Robô
SCSi	Sistema de Coordenadas do Sensor Si
SLAM	Simultaneous Localization And Mapping
SMG-SLAM	Scan-Matching Genetic SLAM
SoC	System on Chip
SSMR	Skid-Steering Mobile Robot
TTM	Time to Market
VHDL	VHSIC Hardware Description Language
UKF	Unscented Kalman Filter
UPF	Unidade em Ponto Flutuante

1 Introdução

1.1 Contextualização e Motivações

Nos últimos anos vem sendo desenvolvidos diversos trabalhos de pesquisa relacionados ao uso de FPGAs (do inglês ***F**ield **P**rogrammable **G**ate **A**rrays*) aplicados para a solução eficiente de diferentes tipos de problemas, muitos deles relacionados com o desenvolvimento de *plataformas computacionais embarcadas* [7, 8, 9, 10].

No âmbito de desenvolvimento de soluções na área de eletrônica, existe uma grande demanda de sistemas embarcados requeridos para realizar tarefas computacionais de alto desempenho, devendo atingir requisitos de robustez, tolerância a falhas, baixo custo e baixo consumo de energia [11]. Neste caso, os sistemas embarcados são de proposito especifico, por essa razão são considerados como sistemas computacionais especializados. Estes sistemas podem ser encontrados em plantas industriais, aparelhos médicos, indústria automotiva, plataformas para robôs, indústria aeronáutica e aeroespacial, telefonia celular, entre outras [12].

No caso, da indústria de telefonia celular, nos últimos anos pode ser observado o projeto e uso de sistemas embarcados complexos, os quais incluem internamente processadores de múltiplos núcleos (*dual-core* e *quad-core*), gerando assim uma tendência de processadores heterogêneos, contendo uma variedade de recursos especializados, tais como GPUs (do inglês ***G**raphics **P**rocessing **U**nits*) e GPPs (do inglês ***G**eneral **P**urpose **P**rocessors*). O esforço no mercado de processadores móveis tem sido sempre focado em obter menor tamanho, menor consumo de energia (e portanto, maior duração da bateria), onde todos os processadores são projetos do tipo SoC (do inglês ***S**ystem **o**n **C**hip*) e/ou MPSoC (do inglês ***M**ulti**P**rocessor **S**istem **o**n **C**hip*). Neste mercado, companhias como *Qualcomm*, *Texas Instruments* e *Nvidia* estão competindo na maioria de negócios relacionados com *smartphones* de última geração, a partir de projetos compatíveis com processadores *ARM*.

É possível também dizer que os sistemas embarcados são componentes de *hardware* e *software* integrados em uma solução mais adequada, com o objetivo de cumprir várias funcionalidades de um produto específico. Neste contexto, o projeto de sistemas embarcados usa consigo técnicas

de co-projeto de *hardware/software* (em inglês *hardware/software co-design*). Por esse motivo aparece a necessidade de se verificar quais partes do sistema a ser projetado são críticas nos quesitos de desempenho, área, consumo de potência, robustez, tolerância a falhas, flexibilidade, portabilidade, entre outras. Nesse caso, a partir de um estudo adequado podem ser determinadas partes críticas do sistema, as quais podem ser implementadas em módulos de *hardware* especializados. Para esta tarefa são usadas ferramentas de analise do perfil da aplicação (*profiling*) [13]. Uma opção para desenvolver aceleradores em *hardware* de partes críticas de um sistema é desenvolvimento de ASICs (do inglês ***A**pplication **S**pecific **I**ntegrated **C**ircuits*) ou trabalhar com plataformas reconfiguráveis como FPGAs, entre outras soluções possíveis[14]. Aqui radica uma das principais diferenças entre sistemas embarcados e sistemas computacionais de propósito geral (como o caso dos computadores pessoais), no sentido que sistemas embarcados, na maioria de casos, requerem *hardware* e *software* de propósito específico [15, 16]

Em geral, todos os tipos de sistemas embarcados são projetados para operar sob restrições de portabilidade (peso e tamanho), consumo de recursos, baixa frequência de *clock*, desempenho adequado, baixo consumo de energia e dissipação de potência [17]. Em relação à frequência de *clock*, em geral é desejável uma frequência tão baixa quanto possível, embora existem sistemas embarcados trabalhando a frequências da ordem de GHz, tal como acontece nas plataformas de telefonia celular, como é o caso do plataforma *Samsung Galaxy S® 5*, a qual roda na frequência de 2.5 GHz [18]. Entretanto, trabalhar a frequências altas termina por penalizar em alguns casos a autonomia das baterias deste tipo de sistemas, além de um incremento no consumo de energia.

No caso da restrição de dissipação de potência, conhecida também como o problema de *power-wall*, em [19] se indica a equação de potência de um simples transistor, válida por exemplo para uma CPU (do inglês ***C**entral **P**rocessing **U**nit*), escrita como:

$$P = K \times (CL) \times V^2 \times F, \tag{1.1}$$

onde P é a potência dissipada pelo transistor, K é uma constante, CL é a carga capacitiva, e V com F são a voltagem e frequência de operação respectivamente. É importante mencionar, que a Equação (1.1) mostra uma ideia que existe uma relação direta entre a dissipação de potência e a frequência de operação de um sistema projetado. Por esse motivo são desenvolvidas soluções de *hardware* que explorem o paralelismo intrínseco dos algoritmos que devem ser embarcados, permitindo assim obter implementações com bom desempenho, operando também com baixas frequências de *clock*, aliviando assim o problema de *power-wall*.

Em geral, o ciclo de vida de sistemas embarcados é cada vez menor, levando a novos desenvolvimentos que ocorrem com mais frequência para substituir a outros produtos ultrapassados.

Além disso, a complexidade dos sistemas embarcados está aumentando rapidamente. Isso leva ao fato de que para conseguir um ciclo de vida maior para sistemas embarcados seja necessário mais tempo e força de trabalho. A demanda dos consumidores para aumentar a funcionalidade se traduz diretamente em aumentar a complexidade do sistema embarcado em um chip. Este fato enfatiza a importância de conceitos bem conhecidos na engenharia de produção, tais como *Time-to-Market* (TTM), sendo o período de tempo a partir da concepção da ideia do produto até que esteja disponível para venda. TTM é muito importante em indústrias onde os produtos são ultrapassados rapidamente, como na indústria de tecnologia de ponta, e na indústria da eletrônica, microeletrônica e nanoeletrônica.

Normalmente, os sistemas embarcados são compostos de GPPs, DSPs (do inglês ***D**igital **S**ignal **P**rocessors*), GPUs, FPGAs e circuitos específicos, incluindo ASICs os quais podem implementar módulos de *hardware/software* complexos, tais como observados em dispositivos SoC (do inglês ***S**ystem **o**n **C**hip*). GPPs, GPUs e DSPs são processadores baseados na computação de fluxo de instruções, tendo as restrições típicas do gargalo de Von Neumann. Embora GPPs específicos, assim como GPUs e DSPs tenham vias de dados (*data-paths*) sofisticados, os mesmos possuem as restrições típicas do acesso a memória obrigatório de dados e instruções (tais como *fetch-instructions* e operações do tipo *read/store*). Neste contexto, os barramentos de comunicação CPU-memória se comportam com as características típicas e limitações dos canais de comunicação. Adicionalmente, existe o fenômeno de barreira de memória (*memory-wall*) que caracteriza o fato das memórias terem menor vazão para as operações de leitura/escrita que aquela exigida pelos processadores de alto desempenho [15, 19].

Por outro lado, soluções eficientes baseadas em *hardware* para sistemas embarcados podem ser conseguidas por meio de projetos ASICs, os quais podem implementar soluções baseadas no mapeamento de algoritmos diretamente em *hardware* (*clock-based solutions*). Em ASICs podem ser implementadas soluções baseadas em fluxo de dados (em vez de fluxo de instruções), as quais eliminam as restrições implícitas no modelo de Von Neumann. No entanto, a abordagem de ASICs apresenta altas despesas não recorrentes (NRE, do inglês ***N**on **R**ecurring **E**ngineering*) referentes ao seu fluxo de projeto complexo, o que o torna impróprio para TTMs curtas.

A eficiência em projetos de sistemas embarcados pode ser melhorada através de prototipagem rápida de sistemas, sendo a mesma especialmente útil quando novos desenvolvimentos de *hardware* e *software* estão sendo explorados. Nesta abordagem, o modelo de desenvolvimento para projetos de sistemas embarcados consiste, frequentemente, em uma fase de prototipagem para estudos de viabilidade, de realização do produto final e de testes, e neste caso, o modelo de desenvolvimento permite atingir um produto final rapidamente e bem testado. Em geral, a prototipagem rápida do sistema permite aos projetistas (também chamados *designers*) explorar outras alternativas de projeto e descobrir erros de projeto o mais cedo possível. Pode-se observar

que a janela TTM curta de sistemas embarcados se beneficia significativamente a partir de prototipagem rápida de sistemas. Nesse contexto, os projetos de sistemas embarcados baseados em FPGAs fornecem um sistema de prototipagem rápida, dada a flexibilidade desses dispositivos para projetos de sistemas complexos, bem como baixo consumo de energia estática e dinâmica, alto desempenho e um meio reprogramável para implementar uma variedade de aplicações eletrônicas. Neste caso, as ferramentas de projeto para FPGAs permitem desenvolver sistemas com baixos NREs, devido a que o projetista pode configurar e reconfigurar rapidamente o dispositivo com modificações necessárias. Esta reconfiguração pode ser executada ainda em tempo real, assim como em uma parte específica do dispositivo (reconfiguração dinâmica e parcial).

Adicional ao fato de permitir implementar tecnologias baseadas na abordagem de prototipagem rápida, FPGAs permitem desenvolver arquiteturas baseadas em fluxo de dados, onde operações de leitura/escrita de instruções não são necessárias [17, 19].

Por este motivo FPGAs vêm sendo aplicadas também na área de robótica, onde tem sido desenvolvidas soluções (como parte dos módulos embarcados) nos últimos anos [20, 9]. Um ponto importante é que na área de desenvolvimento de pequenos robôs (ou micro-robótica) existe uma grande demanda pelo desenvolvimento de plataformas com baixo custo, alto desempenho e baixo consumo de potência. Nessa área, as principais aplicações estão sendo focadas na aceleração da execução de algoritmos, usando FPGAs como aceleradores de *hardware*, em parte ou em todo o sistema.

Alguns dos algoritmos mais usados em robótica são os métodos probabilísticos, historicamente utilizados na área de robótica móvel, como é o caso de cadeias de Markov, métodos de Monte Carlo, algoritmos probabilísticos para SLAM (do inglês ***S**imultaneous **L**ocalization **A**nd **M**apping*), filtros de Kalman, entre outros. Esses tipos de algoritmos têm como propósito obter uma maior robustez nas tarefas de localização, mapeamento e navegação de robôs móveis, considerando as incertezas intrínsecas do ambiente e o ruído gerado pelo sistema de medição [21].

Na área da robótica móvel, trabalha-se com um grande número de sensores e plataformas de sensoriamento, os quais proporcionam arquiteturas com informações complementares ou às vezes com aspectos redundantes. Em muitos casos, os robôs móveis transportam sensores para cálculo de posição, como *encoders* e geomagnéticos, ou para a construção de mapas e auto-localização, tais como ultrassons, infravermelhos e os baseados em laser. É aqui que aparece o termo de *Fusão Sensorial*, utilizado quando se trabalha com um conjunto de informações provenientes de vários sensores, cada um deles com parâmetros diferentes de precisão e exatidão.

A fusão sensorial é amplamente utilizada na área de robótica móvel, principalmente focada para resolver problemas de localização, navegação e mapeamento. A fusão de sensores pode

ser definida como o processo de integração de dados provenientes de diferentes sensores, mesmo usando diferentes princípios físicos, para a estimação de uma grandeza física (o mensurando), tendo em conta as diversas medições. A estimação dessas variáveis ou estados, podem ser considerados como parâmetros de entrada necessários para tarefas de navegação, mapeamento (para obter modelos do ambiente), auto-localização, cartografia, planejamento e controle de trajetória.

Para o propósito deste trabalho, o algoritmo de fusão sensorial utilizado foi implementado em uma FPGA Cyclone IV (da companhia *Altera*) [22] e adaptado para a plataforma móvel Pioneer 3-AT (P3-AT da companhia *Mobile Robots Inc.*) [23] para fins de validação, sendo este último um sistema robótico com tração nas quatro rodas equipado com comunicação *Ethernet*, DGPS (do inglês ***D**ifferential **G**lobal **P**ositioning **S**ystem*), 8 sensores de ultrassom na frente e mais 8 traseiros com detecção de obstáculos de 15 cm a 7 m. O P3-AT utiliza *encoders* com correção inercial recomendado para cálculo de posição para compensar derrapagens. Neste trabalho se adicionou e se adaptou à plataforma P3-AT um sistema de medição de distancias por laser (*ladar*) para detecção de obstáculos. Neste caso, o presente trabalho é uma continuação das dissertações de mestrado [5, 24].

A relevância do desenvolvimento de projetos de robótica móvel usando arquiteturas reconfiguráveis como FPGAs radica em que estas últimas abrem a possibilidade de executar futuros trabalhos que tentem resolver problemas de robótica móvel (por exemplo, a localização e mapeamento simultâneo) aplicadas a plataformas menores, como acontece na área micro-robótica, onde se trabalham com robôs de pequeno porte e capacidade de consumo de energia limitado.

1.2 Definição do problema

Quando se usa a fusão de sensores deve-se lidar com o comportamento estatístico de cada um deles, que em alguns casos pode ser conhecido ou não. No caso de serem conhecidas as características do modelo de medição, a tarefa de fusão de sensores depende de técnicas desenvolvidas como a estimativa *a posteriori* e *a máxima verossimilhança*, adaptando os resultados da filtragem de Kalman, assim como outras teorias Bayesianas. Para o caso do Filtro de Kalman em fusão sensorial a maioria de aplicações em robótica móvel estão focadas na construção e manutenção de um modelo de ambiente para o robô móvel, assim como a monitorização da posição desse robô no ambiente. Para conseguir isso, as equações do filtro de Kalman são implementadas em *software* sobre uma plataforma de sistema embarcado baseado em um microprocessador ou microcontrolador. No entanto, a fusão de sensores também pode ser alcançada usando arquiteturas descentralizadas para obter uma maior capacidade de cálculo e comunicação, assim como para melhorar o desempenho. Neste caso, a estimativa local dos parâmetros a partir dos dados

disponíveis é feita seguida da fusão global das estimativas locais; as quais podem ser executadas por *hardware* específicos em que as equações do filtro de Kalman foram distribuídas. Para aplicações do filtro de Kalman em sistemas não lineares é utilizado a versão nomeada de Filtro de Kalman Estendido (EKF), que inclui na sua estrutura o cálculo de matrizes do tipo Jacobianas [25].

No caso de aplicações do EKF utilizando FPGAs (focado na fusão sensorial para resolver o problema de localização em robótica móvel) trabalhos anteriores tiveram que lidar com o problema de armazenamento na memória de grandes volumes de dados provenientes de diferentes sensores. Isto é ainda mais crítico, quando a dimensões das matrizes aumentam em função do número de sensores. Para esse caso, é conhecido que a complexidade computacional do algoritmo EKF aplicado para SLAM esta estabelecida por $O(n^2)$, onde n representa o número de características (*features*) [26], na qual cada característica precisa ser detectada por um sistema de medição (geralmente baseado em vários sensores). Além disso, o armazenamento prévio de dados dos sensores para a etapa da predição e estimação do EKF leva as operações matriciais (como adição/subtração, multiplicação, divisão, transposta e inversa) aumentar para dimensões maiores. Esse fato, por sua vez, requer a necessidade de projetos de arquiteturas de *hardware* que considerem requerimentos de largura de banda de memória para os EKFs implementados em FPGAs [20, 27], representando um caso específico do problema de *memory-wall* [15].

Para solucionar o problema das dimensões elevadas das matrizes geradas no EKF é usada uma abordagem *sequencial* do algoritmo, a qual foi proposta em [28], sendo as observações dos sensores processadas uma por vez. A partir de essa abordagem é possível reduzir a dimensão das matrizes, tendo impacto direto no desempenho das operações matriciais como adição, multiplicação, divisão, inversa, entre outras. Este é um fato muito interessante para implementações em FPGA, dados os seguintes aspectos: (a) FPGAs têm recursos de *hardware* limitados, que podem ser essenciais para operações em ponto flutuante, (b) requisitos de entrada e saída das FPGAs podem ser drasticamente reduzidos utilizando um enfoque sequencial do EKF, (c) a abordagem sequencial do EKF permite ao projetista usar dispositivos FPGA menores e mais baratos, (d) o uso do paralelismo intrínseco do EKF na FPGA melhora o desempenho de ambos o algoritmo EKF e a aplicação de localização global, (e) o potencial do paralelismo das FPGAs pode equilibrar o processamento dos dados em série, e (f) a utilização de pequenos dispositivos com um desempenho apropriado (por exemplo na área da micro-robótica) faz possível desenvolver soluções para estes tipos de sistemas [15].

Adicionalmente, é importante mencionar que o uso de FPGAs para embarcar algoritmos aplicados à robótica móvel tem sido um tópico desenvolvido no Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados (LEIA-UnB), como foram nos casos das dissertações de mestrado [5, 24], os quais geraram várias publicações. O presente trabalho assim como as

dissertações de mestrado mencionadas previamente, utilizaram a plataforma robótica Pioneer 3-AT (P3-AT) para validar conceitos e soluções embarcadas de técnicas de fusão sensorial, direcionando as mesmas para serem implementadas e adaptadas a pequenas plataformas móveis para futuros trabalhos.

1.3 Objetivos

1.3.1 Objetivo Geral

Como objetivo geral, esta dissertação apresenta o desenvolvimento e avaliação de uma arquitetura em *hardware* (de baixo custo em área e alto desempenho) do algoritmo de Filtro de Kalman Estendido (EKF sequencial) para o problema de localização local de robôs, considerando um sistema multi-sensorial de *encoders* e *ladar* (LRF), e implementando a arquitetura em FPGA.

Para este caso, considera-se que o robô se encontra em um ambiente de mapa conhecido, além de trabalhar com um processamento de dados *online*.

1.3.2 Objetivos Específicos

Os objetivos específicos deste trabalho são os seguintes:

(a) Obtenção do modelo do sistema correspondente à plataforma robótica móvel P3-AT.

(b) Obtenção do modelo de medição tendo em conta a disposição dos sensores no robô.

(c) Obtenção de todas as equações do algoritmo EKF, baseadas nos modelos de sistema e medição propostos para um ambiente específico.

(d) Implementação das equações que formam parte do algoritmo EKF em *software* para efeito de validação dos modelos.

(e) Adaptação de arquiteturas de operadores em ponto flutuante (desenvolvidas no LEIA-UnB) para este projeto.

(f) Desenvolvimento de uma arquitetura em *hardware* da etapa de predição do algoritmo EKF para o problema de localização em robótica móvel.

(g) Desenvolvimento de arquiteturas em *hardware* para todo o algoritmo EKF, usando diferentes estratégias, para comparação de resultados de desempenho e consumo de recursos computacionais.

(h) Implementação em *hardware* da versão sequencial de todo o algoritmo EKF, a partir da arquitetura em *hardware* da etapa de estimação do EKF (desenvolvida no LEIA-UnB) visando a redução do consumo de recursos na FPGA.

(i) Validação das arquiteturas, comparando resultados em *software* com os resultados em *hardware*.

(j) Criação de um cenário real para a plataforma Pioneer 3-AT, para efeito de validar a aplicabilidade da arquitetura proposta para localização em robótica móvel.

1.4 Resultados Alcançados e Contribuições deste Trabalho

Neste contexto, as contribuições deste trabalho são:

(a) Desenvolvimento de uma arquitetura para a versão sequencial do algoritmo completo EKF (etapas de predição e estimação), implementando a mesma em FPGA com representação em ponto flutuante. A implementação foi adaptada em uma plataforma robótica, trabalhando com um processamento de dados *online*.

(b) Validação dos resultados em termos de desempenho, consumo de potência e recursos *hardware* e sua funcionalidade em um cenário real, conseguindo a aplicabilidade do sistema proposto para robótica móvel, e sendo dirigida para pequenos robôs móveis com restrições de consumo de potência e frequências operacionais.

(c) Publicação nos anais do evento internacional SFORUM 2014 - Chip in Aracaju (Sergipe, Brasil - Setembro/14).

(d) Publicação nos anais do evento internacional ICOAI 2014 (Barcelona, Espanha - Dezembro/14) e no jornal internacional IJMLC.

(e) Publicação nos anais do evento internacional LASCAS 2015 (Montevideo, Uruguai - Fevereiro/15) e nos procedimentos da *IEEE Xplore Digital Library*.

1.5 Organização do Trabalho

O trabalho está organizado da seguinte maneira:

O capítulo 2 inicia abordando o problema da localização em robótica móvel, descrevendo os algoritmos usados para a resolução deste problema. O capítulo conclui mostrando um resumo da revisão bibliográfica sobre as diferentes arquiteturas pesquisadas.

Em seguida, o capítulo 3 começa descrevendo o modelamento da plataforma móvel usada neste trabalho (P3-AT), necessário para poder obter futuramente as equações do EKF na etapa de predição.

No capítulo 4, desenvolvem-se todas as equações para a versão sequencial do algoritmo do Filtro de Kalman Estendido, usado para localização de robôs móveis em um entorno especifico.

O capítulo 5 começa com um pequeno resumo sobre a tecnologia FPGA, bem como suas vantagens e termina apresentando as estruturas das unidades em ponto flutuante (UPFs) usadas neste trabalho.

Logo após, o capítulo 6 apresenta as arquiteturas desenvolvidas para o problema de localização em robótica móvel. São apresentadas duas abordagens para a solução do problema, bem como a estrutura das arquiteturas.

No capítulo 7, mostram-se os resultados dessas aplicações e, por último, no capítulo 8 conclui-se o trabalho com projeções de trabalhos futuros.

2 Localização de Robôs Móveis

2.1 Localização robótica

A localização em robótica móvel pode ser definida como a auto-estimação da posição de um veículo em um ambiente determinado. Como é descrito em [1], todo o processo de navegação robótica tem vinculada as seguintes quatro tarefas:

(a) Percepção: o robô deve interpretar as informações provenientes de seus sensores para extrair dados significativos.

(b) Cognição: o robô deve decidir como agir para alcançar seus objetivos.

(c) Controle de trajetória: o robô deve modular a velocidade de seus atuadores para percorrer a trajetória delineada.

(d) Localização: determinação da posição do robô.

Destas quatro funções, mencionadas anteriormente, é importante assinalar que a localização tem recebido uma atenção especial da comunidade científica nas últimas décadas, devido a sua importância para a construção de mapas de ambiente. O esquema de funcionamento da tarefa de localização é apresentado na Fig. 2.1.

A localização de robôs móveis pode ser abordada usando métodos de transformação de coordenadas [26], devido a que os mapas são considerados como um sistema de coordenada global, que é independente da *pose* (ou postura) do robô. Logo, a localização se converte em um procedimento de correlacionar o sistema de coordenada do mapa global com o sistema de coordenada local do robô móvel. A partir desta transformação de coordenadas é possível que o robô expresse a localização dos objetos de interesse com sua própria coordenada local. Desta forma é possível expressar a postura do robô mediante $x = \{x\, y\, \theta\}^T$ [26] (vide Figura 2.2).

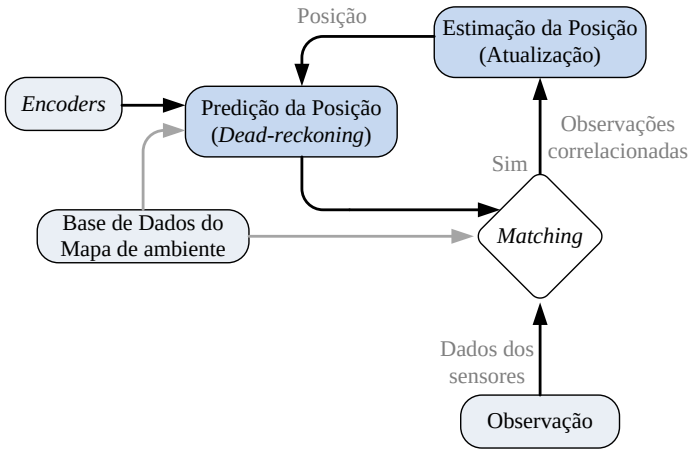


Figura 2.1. Esquema geral para localização de robôs móveis, adaptada de [1]

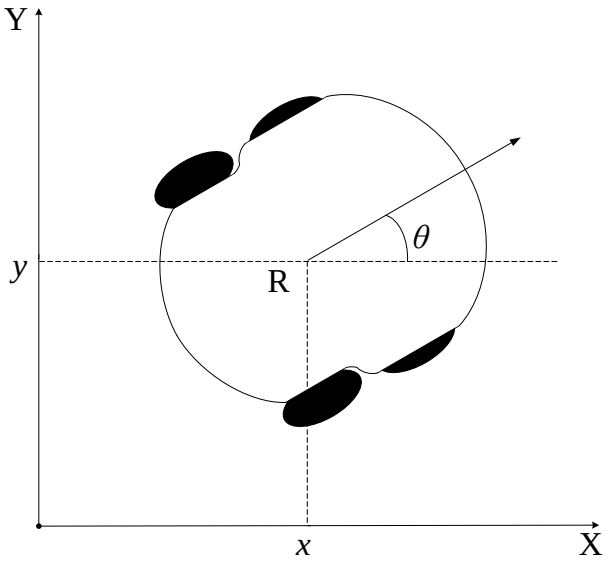


Figura 2.2. Representação da postura de um robô em sistema de coordenadas de duas dimensões

2.2 Tipos de problemas de localização e *Dead-reckoning*

Podem-se distinguir três tipos de problemas de localização [26]:

(a) *Localização local ou rastreamento de posição*: Assume-se que a posição inicial do robô é conhecida, onde o efeito do ruído associado ao movimento do robô é usualmente pequeno. A incerteza da pose do veículo muitas vezes é aproximada a uma distribuição unimodal (por exemplo, uma Gaussiana). O problema de rastreamento da posição é um problema

local, devido a que a incerteza é local e confinada a uma região próxima da verdadeira posição do móvel.

(b) *Localização global*: Neste caso, a posição inicial do robô é desconhecida. O móvel é inicialmente colocado em algum lugar no seu ambiente, mas faltando o conhecimento desta informação. Para este caso, não pode ser assumida uma limitação do erro da pose do robô, na qual as distribuições de probabilidade unimodal não são geralmente adequadas como abordagens. Este tipo de localização é mais difícil do que o rastreamento de posição; de fato, a localização global inclui à outra.

(c) *Problema robô sequestrado*: Este problema é uma variante do problema de localização global, sendo ainda mais difícil. O robô pode acreditar que sabe onde está, enquanto isso não acontece. Testes de algoritmo de localização por este tipo de problema medem a capacidade de recuperação para falhas de localização global, algo essencial para robôs verdadeiramente autônomos.

No presente manuscrito será abordado o problema de localização local, para o qual se trabalhara com uma técnica nomeada de *Dead-reckoning*, a qual permite determinar a posição atual de um robô móvel ou veículo a partir da análise das informações sobre sua posição anterior além da sua velocidade e curso conhecido. A implementação mais simples de *dead-reckoning* é conhecida como *odometria*, na qual a partir de algum tipo do odômetro embarcado no veículo é possível determinar a posição do robô móvel.

Nesse caso, uma forma de odometria muito utilizada nestes dias é baseada em codificadores ópticos diretamente acoplados aos eixos das rodas dos veículos, conhecidos também como *encoders*. Como comentário adicional, pode-se mencionar que o termo *dead-reckoning* deriva da expressão *deduced reckoning* (que em português significaria *contagem deduzida*) [6].

2.3 Algoritmos usados em localização de robôs móveis

Em seguida, serão descritos quatro dos algoritmos mais usados para o problema de localização em robótica móvel, os quais são: *Localização por GRID*, *Filtro de Partículas*, *Algoritmo de Markov* e *Algoritmo EKF*.

2.3.1 Algoritmo *GRID*

A localização por *GRID* (ou *grade* em português) é um algoritmo que consiste em discretizar o espaço, dividindo o mesmo em unidades de tamanho predefinido, que são classificadas como

ocupadas o vazias, com um determinado nível de probabilidade e confiança. Este método se baseia em um filtro de histograma para representar a *crença* (ou confiança, mencionada anteriormente) *posteriori*, ou crença posterior. A crença posterior esta dada por uma coleção de valores de probabilidades discretas. O estado da pose do robô no tempo t é denotado por x_t e a crença sobre o estado da variável x_t é denotada por $bel(x_t)$, como mostra a Equação 2.1.

$$bel(x_t) = p_{k,t}, \tag{2.1}$$

onde cada probabilidade $p_{k,t}$ é definida sobre uma célula-grade x_k.

O conjunto de todas as células-grade dá lugar a uma partição do espaço de todas as posturas, como é mostrado na Equação 2.2:

$$range(X_t) = x_{1,t} \cup x_{2,t} \cup ... x_{k,t}, \tag{2.2}$$

Em aplicações para vários ambientes internos o algoritmo GRID é usado considerando cada célula-grade de dimensões 15 cm $\times$ 15 cm correspondente às coordenadas XY (como visto na Figura 2.3), e de 5 graus para a dimensão rotacional. Neste caso, as dimensões de cada célula-grade e a partição de espaço são invariantes no tempo. A partir de uma representação mais fina é possível obter melhores resultados, no entanto isto aumenta os requisitos computacionais da aplicação [26].

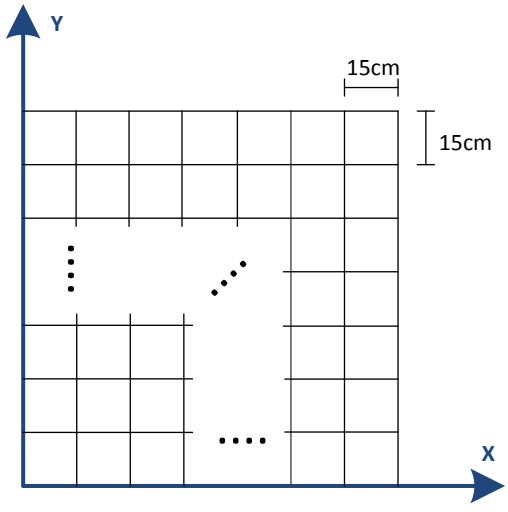


Figura 2.3. Esquema de uma representação por *grid*

O Algoritmo (1) mostra um pseudo-código para o caso de uma implementação básica. Este pseudo-código precisa de ter como entrada os valores de probabilidade discreta $p_{k,t-1}$, em conjunto com a medição mais atual z_t , o controle u_t e o mapa, denotado por m. No caso das funções

Algoritmo 1 Localização por *GRID*

1: **for all** k **do**
2: $\quad \bar{p}_{k,t} = \sum_i p_{i,t-1} f(me(x_k), u_t, me(x_i))$
3: $\quad p_{k,t} = \eta\, g(z_t, me(x_k), m)$
4: **end for**
5: **return** $p_{k,t}$

$f()$ e $g()$, elas representam o modelo do movimento e modelo de medição respectivamente. A função $me()$ retorna o centro de gravidade da célula-grade x_k. Na linha 1 do pseudo-código, define-se que a malha interna do algoritmo percorre com cada iteração todas as células (unidades predefinidas), denotando como indexador a variável k. Na linha 2 se implementa a atualização do modelo de movimento e na linha 3 a atualização da medição. Adicionalmente, é feita uma normalização para as probabilidades finais a partir do normalizador η mostrado na linha 3. Neste caso, considera-se que cada célula possui a mesma área.

A seguir um resumo desta técnica de localização é mostrado, de acordo com [26]:

(a) A técnica por *GRID* representa a crença posterior por histogramas;

(b) Pode-se fazer representações do ambiente em duas ou três dimensões;

(c) A localização por *GRID* pode globalmente localizar o robô;

(d) O tamanho da grade influencia diretamente na precisão e eficiência computacional. Células pequenas implicam em menores erros de estimação, porém em alto custo computacional e um maior tempo para localizar-se, ou seja, dificuldade na execução do algoritmo em tempo real (isto pode gerar um amplo espaço de pesquisa na área de processamento paralelo, usando GPUs e/ou FPGAs).

2.3.2 Filtro de Partículas

O Filtro de Partículas ou algoritmo de Monte Carlo (MCL, do inglês ***M**onte **C**arlo **L**ocalization*), é um método baseado na representação do estado usando um conjunto de amostras (partículas) associando a cada uma delas um peso determinado. Este método se baseia na distribuição aleatória de possíveis estados em um espaço de configurações (espaços livres para navegação em um mapa de ambiente) [29].

Neste algoritmo, cada estado é representado por uma partícula e pode ser usado um número grande de partículas. Este número de partículas interfere na precisão e velocidade de convergência, assim como no cálculo computacional. A quantidade de partículas deve ser suficiente para uma cobertura adequada do mapa. Desta forma, a cada tomada de controle do robô pelo mapa do ambiente é aplicada às partículas a partir do modelo estatístico de movimento. Além

disso, é feita uma comparação entre cada observação real do estado dos sensores e a observação das partículas.

Nesta abordagem, cada partícula recebe um peso que indica a probabilidade (conhecida também como *likelihood*) relacionada com a posição real do robô. Este peso dependerá de que próximo se encontre a leitura do sensor com o valor estimado. A partir disso, realiza-se uma nova seleção aleatória de amostras das possíveis configurações; desta vez, levando em conta os pesos das partículas. Quanto maior a *likelihood* de uma partícula, maior a probabilidade de que ela seja escolhida novamente. Com o correr do tempo, isto resulta em uma aproximação da real posição do robô, representada pela partícula com a melhor estimativa da posição. Uma vantagem deste algoritmo é que se o robô fosse colocado em outra posição, pode se auto-localizar de maneira natural, já que as partículas se espalharão pelo mapa em posteriores seleções (ou *samplings*) [30].

Algoritmo 2 Localização por Filtro de Partículas

1: $\bar{X}_t = X_t = \oslash$
2: **for** $m = 1 \rightarrow M$ **do**
3: $\quad x_t^{[m]} = f(u_t, x_{t-1}^{[m]})$
4: $\quad w_t^{[m]} = g(z_t, x_{t-1}^{[m]}, m)$
5: $\quad \bar{X}_t = X_t + \left\langle x_t^{[m]}, w_t^{[m]} \right\rangle$
6: **end for**
7: **for** $m = 1 \rightarrow M$ **do**
8: $\quad$ *projete i com probabilidade* $\propto w_t^{[i]}$
9: $\quad$ *adicione* $x_t^{[i]} \rightarrow X_t$
10: **end for**
11: **return** X_t

Uma versão básica do método de Filtro de Partículas pode ser vista no Algoritmo (2), considerado para M partículas $X_t = \{x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}\}$, onde $x_t^{[m]}$ (com $1 \leq m \leq M$) representa cada partícula que é uma estância concreta do estado x no tempo t. A Linha 3 contém amostras do modelo de movimento $f()$, usando partículas de crença atual como pontos de partida. O modelo de medição $g()$ é então aplicado na linha 4 para determinar o peso w da referida partícula de importância. A crença inicial $bel(x_0)$ é obtida gerando aleatoriamente um número M de partículas a partir da distribuição previa $p(x_0)$ e atribuindo o fator de importância uniforme M^{-1} para cada partícula. As funções $f()$ e $g()$ representam o modelo do movimento e modelo de medição respectivamente [26].

Em seguida, apresenta-se um resumo do algoritmo MCL [26]:

(a) O algoritmo MCL representa a crença posterior usando partículas;

(b) A precisão computacional está ligada diretamente ao número de partículas;

(c) O algoritmo localiza globalmente o robô;

(d) Pela adição de partículas aleatórias (*random particles*), o MCL resolve problemas de sequestro de robô (*kidnapped robot problem*);

(e) O algoritmo permite uma distribuição usando diferentes distribuições de probabilidade, ao contrário dos filtros KF/EKF (visto nas seguentes subseções) que trabalham unicamente com funções de distribuição de probabilidade Gaussianas (devido a que baseia-se no principio de verossimilhança).

2.3.3 Algoritmo de *Markov*

O algoritmo de Markov (derivado do algoritmo do filtro de Bayes) para localização, trabalha com o uso de uma *crença* probabilística no tempo $t-1$, transformando-a em uma crença no tempo t. Este método aborda os problemas da localização global, do rastreamento da posição e da mudança de posição arbitrária do robô em um ambiente estático.

Algoritmo 3 Localização por algoritmo de Markov

1: **for all** x_t **do**
2: $\quad \bar{bel}(x_t) = \int p(x_t \| u_t, x_{t-1}, m) bel(x_{t-1}) dx$
3: $\quad bel(x_t) = \eta\, p(z_t \| x_t, m) \bar{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

O Algoritmo (3) mostra o respectivo pseudocódigo básico para a localização por algoritmo de Markov. Como pode ser observado, o algoritmo se baseia em uma crença $bel()$.

Na linha 2, observa-se a primeira etapa deste método conhecida como *atualização do controle* ou *predição*. Nesta etapa, o o controle u_t é processado por meio do cálculo da crença do estado x_t baseado na crença *a priori* do estado x_{t-1}, no controle u_t e na presença de um mapa m. Ou seja, é calculada uma integral (somatória) de um produto de duas distribuições: a distribuição *a priori* assinalada a x_{t-1} e a probabilidade de x_t dado a ação de controle u_t, do estado x_{t-1} e do mapa m [26].

A linha 3 corresponde à segunda etapa do método, sendo denominada de *atualização da medição*. A mesma consiste em uma multiplicação da crença $\bar{bel}(x_t)$ pela probabilidade da medição z_t observada, tendo em conta o estado x_t e o mapa. Isto é feito para cada estado hipotético x_t. Porém, nem sempre o produto resultante será uma probabilidade, ou seja, ele não pode integrar a 1. Assim, o resultado é normalizado, em virtude de uma constante de normalização η.

É importante considerar que o algoritmo requer uma crença inicial $bel(x_0)$ no tempo $t = 0$ como condição inicial de contorno para que se consiga computar (calcular) a crença posterior.

Um resumo do algoritmo de Markov é apresentado de acordo com [26]:

(a) O algoritmo de Markov é uma variante do filtro de Bayes orientado ao problema de localização para robôs;

(b) O algoritmo representa uma técnica baseada em processos estatísticos;

(c) A localização por este método aborda o problema de rastreamento de posição, o problema da localização global, e o problema do robô sequestrado em ambientes *estáticos*.

2.3.4 Algoritmo *EKF*

Finalmente, apresenta-se como quarto algoritmo de localização, o Filtro de Kalman (KF, do inglês ***K**alman **F**ilter*). Este algoritmo matemático, foi desenvolvido por Rudolf Kalman [31], como uma técnica de filtragem e predição em sistemas lineares [26]. Esta técnica tem numerosas aplicações em tecnologia, entre as quais estão o controle de trajetória, navegação aplicada à indústria aeronáutica e espacial, além de seu uso para processamento de sinais.

O filtro de Kalman é um algoritmo capaz de realizar as inferências exatas sobre um sistema dinâmico linear de forma ótima, sendo um modelo Bayesiano semelhante ao modelo de Markov. Porém nesse caso, o espaço de estados das variáveis não observadas é contínuo e todas as variáveis (observadas e não observadas) apresentam uma distribuição normal [26].

No entanto, para o caso de sistemas não lineares uma das opções é trabalhar com uma variação do algoritmo chamado Filtro de Kalman Estendido (EKF, do inglês ***E**xtended **K**alman **F**ilter*). No Algoritmo (4) é descrito o pseudocódigo do EKF, o qual possui duas etapas:

(a) Predição (correspondentes as Linhas 1 e 2).

(b) Estimação (correspondentes as Linhas 3, 4 e 5).

Algoritmo 4 Localização EKF

1: $x_t^- = f(u_{t-1}, x_{t-1}^+)$
2: $P_t^- = F_t P_{t-1}^+ F_t^T + Q_t$
3: $G_t = P_t^- H_t^T (H_t P_t^- H_t^T + R_t)^{-1}$
4: $x_t^+ = x_t^- + G_t(z_t - h(x_t^-)$
5: $P_t^+ = (I - G_t H_t) P_t^-$
6: **return** x_t^+, P_t^+

O Filtro de Kalman Estendido utiliza um modelo dinâmico não linear de um sistema $f()$ (como é o caso das Equações do movimento do robô), entradas de controle u_t e medições z_t, considerando também na sua estrutura um modelo de medição determinado pela função não linear $h()$, com o objetivo de gerar uma estimativa das grandezas variáveis do sistema (seus estados) x_t e um incerteza associada P_t. Deste modo, a estimativa obtida com a filtragem é

mais próxima dos valores reais que a estimativa obtida utilizando-se qualquer uma das medidas individualmente, permitindo usar a técnica de *fusão sensorial*. No Algoritmo (4) G_t é conhecido como *Ganho do Kalman*; e os super-índices $-$ e $+$ indicam a informação a *priori* (predita) e *posteriori* (estimada) respectivamente. Em relação a Q_t e R_t, estas são as incertezas associadas ao ruido (branco gaussiano de média zero) de processo e medição respectivamente. Adicionalmente, na estrutura do Algoritmo (4) estão presentes os termos F_t e H_t, os quais são as matrizes Jacobianas das funções não lineares $f()$ e $h()$ respectivamente. É importante salientar que no Capítulo 4 será realizado um maior detalhamento do algoritmo EKF orientado para localização robótica.

A seguir é descrito um resumo da técnica de localização EKF [26]:

(a) A técnica se baseia em processos estatísticos;

(b) A Localização EKF é um caso especial da localização pelo algoritmo de Markov;

(c) É implementada para o problema de rastreamento de posição de robô com incerteza limitada e em ambientes com *features* distintas;

(d) A Localização EKF é recomendável para localização global ou ambientes onde a maioria dos objetos não possuem semelhança;

(e) Aplica-se para sistemas cujo comportamento é não linear;

(f) Considera valores de diferentes sensores, apropriado para fusão sensorial.

2.4 Trabalhos correlatos em implementações de algoritmos para localização robótica em FPGAs

Existem na literatura alguns trabalhos que abordam o problema de localização em robótica móvel usando FPGAs. Em [20] foi proposta uma arquitetura baseada em FPGA para o algoritmo EKF, a qual é capaz de processar mapas de 2 dimensões (2D) em tempo real (14Hz), onde o sistema estima simultaneamente um modelo do mapa de ambiente e a posição do robô com base em ambas informações de sensores odométricos e exteroceptivo. Nesse trabalho, também é mencionado que a aplicação é duas ordens de magnitude mais eficiente do que um processador de propósito geral.

Os mesmos autores em [21] mostram uma arquitetura em *hardware* para resolver o problema de SLAM (*Simultaneous Localization And Mapping*), cuja implementação foi completamente realizada em um FPGA Stratix II da Altera. A arquitetura proposta nesse trabalho possui quatro

bancos de memória para armazenamento de dados, um certo número de memória embarcada no *chip*, uma máquina de estados e quatro Elementos de Processamento (PE, do inglês ***P**rocessing **E**lements*). Os autores discutem que o número de operações em ponto flutuante para a implementação do algoritmo EKF é proporcional a n^2, sendo n o número de *features*. Eles indicam que a performance de sua arquitetura é no mínimo uma (1) ordem de magnitude melhor do que uma implementação em um PC (do inglês ***P**ersonal **C**omputer*) de última geração.

Seguindo nessa linha, em [32] se apresenta uma variação do algoritmo SLAM chamado SMG-SLAM (do inglês *Scan-Matching Genetic SLAM*). O algoritmo SMG-SLAM faz uma verificação de cada nova varredura feita por um sensor de *ladar*, com o mapa que seu sistema construiu até esse momento, a fim de encontrar a rotação e a translação do robô desde a verificação anterior, estimando assim a nova pose do robô e atualizando o mapa global em cada estimação. O sistema proposto nesse trabalho considerou um algoritmo genético, o qual foi implementado em *hardware* usando uma representação em ponto fixo. Os autores comentam que a aplicação obteve um fator de aceleração de 4,83 vezes em comparação a solução em *software*.

Em [33] se desenvolveu um co-projeto *hardware/software* aplicado a localização robótica usando FPGA (ponto flutuante), baseada no processador Nios II e em um acelerador de *hardware*. O algoritmo utilizou uma *webcam* externa para rastreamento de imagens, o que os autores chamam de *Algoritmo de Localização Absoluta*. A partir da comparação das imagens atuais com uma imagem de referência, consegue-se calcular a posição absoluta. Esta técnica tem como inconveniente a necessidade de fazer o mapeamento da imagem de dados para o cenário atual, o que introduz erro relacionado com a resolução da imagem e operações de dados, usando uma representação em número inteiro. O sistema também estima o movimento do robô, especificando os pontos inicial e final. Não obstante, o método não garante ainda uma navegação contínua e em tempo real para o robô.

Adicionalmente, em [5] se desenvolveu uma arquitetura em *hardware* para a implementação da etapa de *estimação* (nesse trabalho a etapa é chamada de correção e atualização) de algoritmo EKF, considerando uma versão sequencial do mesmo. Para a obtenção dessa arquitetura, primeiro foi necessário desenvolver um modelo para o sistema de medição para efeito de definir a função não-linear $h()$, presente no algoritmo EKF. A arquitetura mencionada foi projetada considerando uma Máquina de Estados Finitos (FSM, do inglês ***F**inite **S**tate **M**achine*), a qual por sua vez usa operadores de aritmética de ponto flutuante, necessários para fazer a fusão de dados provenientes de diferentes sensores tais como ultrassom e *ladar* (LRF). Nesse trabalho, foi considerado que o robô se encontrava parado, tendo conhecimento da estrutura do mapa de ambiente e também da sua posição dentro desse mapa. Além disso, o sistema trabalhou com um processamento de dados *offline*.

É relevante salientar, que para o caso do trabalho de dissertação apresentado neste documento

será usado como ponto de partida a arquitetura da etapa de *estimação* desenvolvida em [5], fazendo uma extensão da mesma com objetivo de obter toda a arquitetura em *hardware* do EKF tanto para a etapa de *predição* (a partir de um modelo odométrico) quanto para a etapa de *estimação*, alem de considerar que o robô estará em movimento trabalhando com processamento de dados *online*.

Por outro lado, existem trabalhos focados em implementações do filtro de Kalman em FPGAs, considerando as vantagens do paralelismo intrínseco do algoritmo e o potencial das arquiteturas reconfiguráveis para a sua aplicação. Em todos os casos aparece a complexidade de lidar com operações matriciais. Este é o caso de [34], onde se apresenta uma solução aproximada do Filtro de Kalman (KF) usando a expansão de Taylor, implementada em um FPGA Virtex-4 da Xilinx. O trabalho tenta eliminar a operação de inversão matricial presente na equação do ganho (veja Equação (4.9)) a partir da expansão das series de Taylor com cálculos matriciais. Essa implementação em *hardware* utiliza três (3) módulos: *multiplicação de matriz*, *adição/subtração de matriz* e *soma ponderada de matrizes*. Os autores mencionam outra forma de implementação do algoritmo KF baseada no algoritmo Bierman-Thorton [35, 36]. Além disso, os resultados foram comparados com uma implementação convencional em C++ do KF, demonstrado que a sua proposta do KF aproximado tem um melhor desempenho em consumo de *hardware* e *throughput* do que as outras implementações.

Com respeito ao tema de fusão sensorial, em [10] foi realizado uma arquitetura em *hardware* para a técnica de fusão de sensores, cuja implementação foi embarcada em uma placa de desenvolvimento DE0-Nano com um FPGA Cyclone IV. A técnica foi aplicada para a estimação de distâncias usando como sensores um sensor de ultrassom e de infravermelho. Os autores relatam uma velocidade na ordem de 524 vezes mais rápido comparado com o algoritmo implementado em *software*.

Como é visto nos trabalhos anteriores explicados, está presente o objetivo de alcançar implementações reais de algoritmos (EKF, SLAMs entre outros) em FPGAs, usando representação em ponto flutuante ou fixo, e procurando sempre o baixo consumo potência e de recursos de *hardware* para obter um ótimo desempenho. Considerando isto, a abordagem sequencial do EKF surge como uma opção para ser usada em plataformas reconfiguráveis de FPGAs, isto devido às dimensões pequenas das matrizes, o que faz fácil a execução das suas respectivas operações aritméticas.

2.5 Discussão das técnicas de localização e trabalhos correlatos

A partir dos resumos apresentados dos algoritmos mencionados na Seção 2.3 foi feita uma comparação entre cada técnica, como mostra a Tabela 2.1; onde uma quantidade maior de símbolos $*$ significa que o algoritmo tem uma maior presencia da respectiva característica da primeira coluna.

Tabela 2.1. Comparação das diferentes implementações para localização

	EKF - Markov	**F. de Partículas**	**GRID**
Medições	*Landmarks*	Linhas de medição	Linhas de medição
Ruído (Medição)	Gaussiana	Qualquer	Qualquer
Posterior	Gaussiana	Partículas	Histograma
Eficiência (Memória)	****	***	*
Eficiência (Tempo)	****	***	*
Implementação	***	****	*
Resolução	***	**	**
Robustez	*	***	***
Localização Global	Não	Sim	Sim

Tendo em mente que futuramente será trabalhado um projeto mais complexo, o qual abordará o problema da localização e mapeamento simultâneo (SLAM), aplicado à área de micro-robótica, onde serão utilizados robôs de pequeno tamanho e consumo, decidiu-se usar como algoritmo de localização ao filtro EKF pelos seguintes motivos:

(a) O algoritmo servirá como base para o algoritmo SLAM nos futuros trabalhos;

(b) A estrutura do algoritmo considera a técnica de fusão sensorial;

(c) O algoritmo é indicado para predição de estados em Sistemas Dinâmicos Não Lineares.

Adicionalmente, foi realizada uma comparação entre os trabalhos correlatos (discutidos na Seção) na área de implementação em FPGAs de algoritmos de localização aplicados à robótica móvel (vide Tabela 2.2).

Para concluir, a partir dos trabalhos correlatos discutidos neste capítulo, conclui-se que ainda não foi implementada uma abordagem sequencial do algoritmo EKF em plataformas FPGAs, que considere ambas, as etapas de predição e estimação, usando representação em ponto flutuante com aplicação na localização de robôs móveis.

Tabela 2.2. Comparação de trabalhos correlatos usando FPGAs aplicados à robótica móvel

	Trab. [20, 21]	Trab. [32]	Trab. [33]	Trab. [5]	Presente Trabalho
Representação numérica	Flutuante	Fixo	Flutuante	Flutuante	Flutuante
Algoritmo	EKF-SLAM	SMG-SLAM	Localização absoluta	EKF-Localização	EKF-Localização
Arquitetura em HW (de *Predição* EKF)	Não	-	-	Não	Sim
Arquitetura em HW (de *Estimação* EKF)	Sim	-	-	Sim	Sim
Sistema de medição	Câmera	Câmera	LRF	LRF	LRF
Execução *online*	Sim	Sim	Não	Não	Sim
FPGA	EP2S9OF102OC4 (Celoxica RC250)	XC5VFX70T (Xilinx Virtex-5)	Starter Kit (Altera Cyclone III)	EP4CE115F29C7 (Altera Cyclone IV)	EP4CE115F29C7 (Altera Cyclone IV)

2.6 Conclusões do Capítulo

Neste capítulo foram apresentados os conceitos de localização para robótica móvel, assim como discutidos alguns dos algoritmos usados nessa área, os quais foram: a *Localização por GRID*, *Localização por Filtro de Partículas*, *Localização por algoritmo de Markov* e *Localização por algoritmo EKF*.

A partir da discussão das técnicas de localização, decidiu-se usar como algoritmo de localização o filtro EKF, onde a presente proposta também foi comparada com outros trabalhos correlatos (vide Tabela 2.2).

3 Modelagem da plataforma robótica móvel

3.1 Introdução

Neste capítulo a modelagem cinemática de uma plataforma robótica é formulada, tendo em conta que vai se trabalhar com um robô móvel de quatro rodas deslizantes, que será denominado neste trabalho de *4-SSMR* (do inglês *four wheel* ***S****kid-****S****teering* ***M****obile* ***R****obot*).

Hoje em dia podem ser encontrados mecanismos de acionamento baseados em deslizamento em diversas aplicações relacionadas com as áreas de agricultura, mineração e militar. Este tipo de sistemas de movimentação também é útil para robôs móveis em terrenos acidentados, com aplicações de campo, tais como exploração planetária, detecção minas terrestres, resgate, entre outras [37]. Além disso, como foi mencionado anteriormente, o sistema de locomoção supracitado é também utilizado pela plataforma experimental de pesquisa Pioneer 3-AT [23] (vide Figura 3.1), usada neste trabalho.

Figura 3.1. Plataforma experimental Pioneer 3-AT

A direção de um *SSMR* se baseia na condução diferencial do par de rodas em cada lado (as velocidades relativas dos lados esquerdo e direito) do robô móvel [2].

Para tarefas de controle, cálculo de trajetória e navegação é necessário obter um modelo do sistema (ou chamado também modelo do *processo*), devido a que o mesmo definirá a função não linear $f()$ usada no algoritmo EKF (vide Equação (4.1)).

3.2 Trabalhos relacionados com modelagem de 4-SSMRs

Nesta seção são mencionados alguns trabalhos relacionados com a modelagem de robôs do tipo 4-SSMR. Este o caso de [2], onde um modelo matemático de um robô móvel de quatro rodas deslizantes é apresentado de uma forma sistemática. Neste caso, assumiu-se a existência de derrapagem nas rodas a partir de considerar as restrições não-holonômicas na modelagem. Nesse trabalho, o robô é considerado como um subsistema constituído por níveis cinemáticos, dinâmicos e de acionamentos. Em seguida, um projeto de um controlador cinemático com base no algoritmo introduzido por [38, 39] é mostrado. Logo, foi desenvolvida uma extensão da lei de controle cinemático nos níveis dinâmicos e de motor por meio da análise de Lyapunov e a técnica *backstepping*. Para validar o algoritmo concebido, foram discutidos os resultados de simulações para os casos de rastreamento de trajetória e *set-point*.

Seguindo com o tema de modelagem de robôs SSMR, em [40] desenvolveu-se um método para obter experimentalmente um modelo cinemático otimizado para veículos SSMRs, baseado na limitação dos centros instantâneos de rotação (CIRs) de rodas no plano de movimento.

Este último modelo foi usado como base em [37] para melhorar o controle de movimento em tempo real e o *dead-reckoning* para este tipo de veículos, considerando os efeitos de derrapagem, mas sem introduzir a complexidade de cálculos dinâmicos nas iterações. Esse trabalho forneceu mais informações sobre este método de modelagem proposto para veículos SSMRs, o qual foi aplicado com sucesso para a plataforma robótica de pesquisa Pioneer 3-AT, com diferentes tipos de pneus e tipos de terreno.

Outros trabalhos relacionados com a modelagem de plataformas 4-SSMRs têm sido propostos, embora em todos os casos, os mesmos apresentam abordagens baseados em [2].

A partir dos trabalhos supracitados, decidiu-se usar como base para o presente trabalho a modelagem proposta em [2], devido a que o mesmo considera as restrições não-holonômicas no modelo. Também será considerado no modelo de sistema, adotado nesta dissertação, a abordagem de [37], já que nesse artigo consideraram-se os efeitos de derrapagem a partir de cálculos cinemáticos, os quais foram testados e validados na plataforma robótica Pioneer 3-AT (usada também na presente pesquisa).

3.3 Modelo do Sistema

Para começar com a modelagem da plataforma SSMR, assume-se que o robô realiza um movimento plano e que o mesmo está localizado em um sistema global $Xg - Yg$ (ver Figura 3.2) com coordenadas generalizadas, segundo o mostrado na Equação (3.1):

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{3\times1} , \tag{3.1}$$

onde $\boldsymbol{q}$ é o vetor de posição generalizadas.

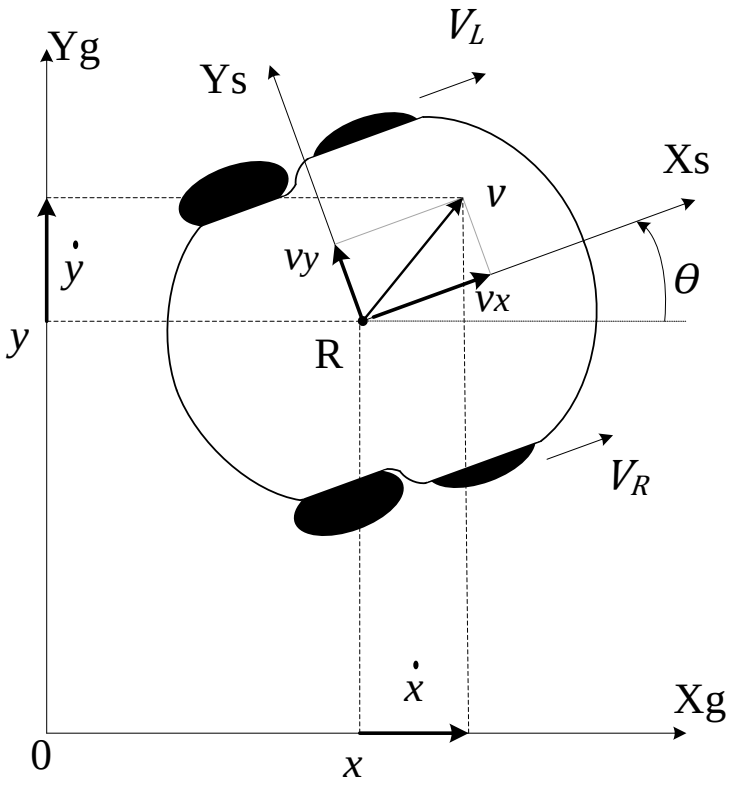


Figura 3.2. Cinemática da plataforma móvel, adaptado de [2]

A seguir, representa-se o vetor de velocidade do centro de massa (ponto R) no sistema local do robô $Xs - Ys$ como $\boldsymbol{v} = [v_x, v_y]^T$, onde v_x e v_y são a velocidade longitudinal e lateral do robô [39]. Então, a partir da Figura 3.2 e utilizando matrizes de rotação, pode-se determinar a equação de movimento cinemático, tal como mostrado na Equação (3.2):

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_{3\times1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}_{3\times3} . \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix}_{3\times1} , \tag{3.2}$$

onde $\dot{\boldsymbol{q}}$ denota vetor de velocidade generalizada e w é a velocidade angular do robô móvel.

Adicionalmente, devem-se considerar algumas restrições no modelamento do robô. Nesse contexto, a Figura 3.3 mostra a relação geométrica que existe entre os vetores de velocidades

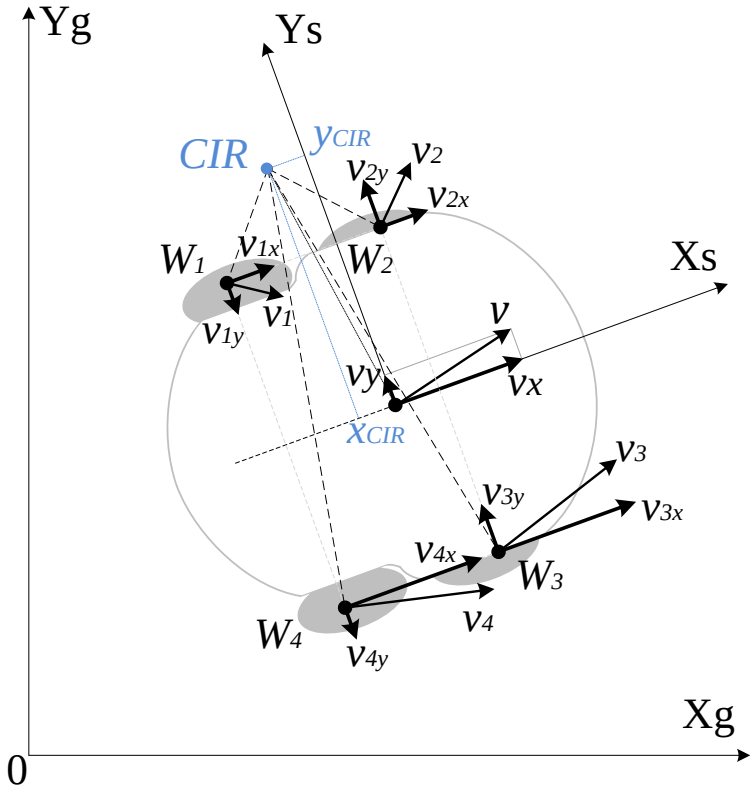


Figura 3.3. Perfil de velocidades na plataforma SSMR, adaptado de [2]

de cada uma das rodas do SSMR com respeito ao *Centro Instantâneo de Rotação* (CIR), cujas coordenadas no sistema local do robô $Xs - Ys$ são (x_{CIR} , y_{CIR}). Estes vetores de velocidades estão associados aos pontos de contato das rodas do robô ($W1$, $W2$, $W3$ e $W4$) com o plano $Xg - Yg$. Da Figura 3.3, e a partir de [41], pode-se obter a expressão mostrada na Equação (3.3):

$$V_L = v_{1x} = v_{2x},\ V_R = v_{3x} = v_{4x},\ V_F = v_{2y} = v_{3y},\ V_B = v_{1y} = v_{4y}, \tag{3.3}$$

onde V_L e V_R são as velocidades lineares longitudinais das rodas do lado esquerdo e direto; por outro lado V_F e V_B representam as velocidades laterais das rodas frontais e traseiras, respectivamente.

Além disso, outro ponto que se deve considerar no modelamento do SSMR é a restrição não-holonômica [42] seguinte, mostrada na Equação (3.4):

$$v_y = w.x_{CIR}, \tag{3.4}$$

Para movimentos retos, da Figura 3.3 e da Equação (3.4) pode-se concluir que v_y é igual a

zero (para $w = 0$). A partir da Equação (3.2), tem-se a expressão mostrada na Equação (3.5):

$$v_y = -\sin(\theta)\dot{x} + \cos(\theta)\dot{y}, \tag{3.5}$$

Substituindo a Equação (3.5) na Equação (3.4) e reorganizando os termos é possível expressar a restrição não-holonômica na expressão conhecida como a forma Pfaffian, mostrada na Equação (3.6):

$$A(\boldsymbol{q})\dot{\boldsymbol{q}} = 0,$$

$$\begin{bmatrix} -\sin(\theta) & \cos(\theta) & -x_{CIR} \end{bmatrix} \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0, \tag{3.6}$$

Para o caso da velocidade generalizada $\dot{\boldsymbol{q}}$, a mesma pode ser escrita da forma indicada na Equação (3.7:

$$\dot{\boldsymbol{q}} = S(\boldsymbol{q})\eta, \tag{3.7}$$

onde η é o vector de entrada ou controle para a cinemática definido por:

$$\eta = \begin{bmatrix} v_x \\ w \end{bmatrix}_{2\times 1}, \tag{3.8}$$

e S tem a seguinte expressão:

$$S(\boldsymbol{q}) = \begin{bmatrix} \cos(\theta) & -x_{CIR}\sin(\theta) \\ \sin(\theta) & x_{CIR}\cos(\theta) \\ 0 & 1 \end{bmatrix}_{3\times 2}, \tag{3.9}$$

a qual satisfaze a condição da Equação (3.10), considerando que o vetor $\dot{\boldsymbol{q}}$ esta sempre em um espaço nulo de A.

$$S^T(\boldsymbol{q})A^T(\boldsymbol{q}) = 0, \tag{3.10}$$

Desta maneira, a Equação Cinemática (3.9) pode ser representada pela Equação (3.11).

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_{3\times 1} = \begin{bmatrix} v_x.\cos(\theta) - w.x_{CIR}.\sin(\theta) \\ v_x.\sin(\theta) + w.x_{CIR}.\cos(\theta) \\ w \end{bmatrix}_{3\times 1}, \tag{3.11}$$

Seguidamente, considerando um modelo simétrico, a seguinte Equação (3.12) mostra a relação entre o vector $[v_x, v_y, w]^T$ e as velocidades de deslocamento linear longitudinais do robô $[V_L, V_R]^T$ (baseado em [37]).

$$\begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix}_{3\times1} = (\alpha/D) \begin{bmatrix} D/2 & D/2 \\ x_{CIR} & x_{CIR} \\ -1 & 1 \end{bmatrix}_{3\times2} \cdot \begin{bmatrix} V_L \\ V_R \end{bmatrix}_{2\times1}, \tag{3.12}$$

onde α é o fator de correção para as velocidades de deslocamento linear nominais e D pode ser identificado de maneira experimental para garantir uma alta precisão do modelo. Para obter os parâmetros cinemáticos (D, α), pode-se utilizar um experimento proposto em [40]. O experimento consiste em que o robô deve fazer uma rotação pura em torno do eixo Z tendo velocidades com o mesmo valor, mas de sinal contrário para as entradas V_L e V_R. Em seguida, aplicar a Equação (3.13), onde ϕ é o ângulo real rotado pelo robô. Este termo D pode ser nomeado de distancia efetiva entre as rodas direita e esquerda.

$$D \approx \frac{\int V_R dt - \int V_L dt}{\phi}, \tag{3.13}$$

O parâmetro α pode ser ajustado mediante a fórmula da Equação (3.14), onde d representa a medida da distância real percorrida em um movimento linear.

$$\alpha \approx \frac{2d}{\int V_R dt - \int V_L dt}, \tag{3.14}$$

Estes procedimentos oferecem aproximadamente uma solução para o modelo cinemático, considerando o deslocamento feito pelas rodas. Da Equação (3.12), pode-se derivar o vetor de entrada ou controle, que contem a velocidade longitudinal e angular de acordo com a seguinte relação (somente se não ocorrer nenhum deslizamento longitudinal), conforme a Equação (3.15).

$$\eta = \begin{bmatrix} v_x \\ w \end{bmatrix}_{2\times1} = \begin{bmatrix} \alpha.\frac{V_R+V_L}{2} \\ \alpha.\frac{V_R-V_L}{D} \end{bmatrix}_{2\times1}, \tag{3.15}$$

Finalmente, baseado nas Equações (3.11) e (3.15), é possível obter uma equação cinemática alternativa que governa o comportamento da plataforma móvel a partir de outras entradas de controle, neste caso as velocidades lineares das rodas V_L e V_R:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_{3\times1} = \begin{bmatrix} (\alpha.\frac{V_R+V_L}{2}).\cos(\theta) - x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\sin(\theta) \\ (\alpha.\frac{V_R+V_L}{2}).\sin(\theta) + x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\cos(\theta) \\ (\alpha.\frac{V_R-V_L}{D}) \end{bmatrix}_{3\times1}, \tag{3.16}$$

3.4 Conclusões do Capítulo

Neste capítulo, foi apresentado o conceito de robô móvel de quatro rodas deslizantes (4-SSMR), assim como alguns dos trabalhos correlatos sobre a modelagem deste tipo de robôs.

A partir desses trabalhos foi desenvolvido o modelo de sistema usado nesta dissertação, assumindo as restrições não-holonômicas (que depende do parâmetro x_{CIR}) e os efeitos de derrapagem (que depende dos parâmetros α e D) nas rodas próprias do movimento do robô. Também foi considerado que o movimento do veiculo é realizado em um plano com um sistema de coordenadas $Xg - Yg$, onde a equação cinemática de movimento é obtida de modo matricial, tendo como vetor de entrada as velocidades lineares das rodas de cada um dos lados do robô.

4 Localização EKF

4.1 O Filtro de Kalman Estendido

A principal contribuição deste capítulo é o desenvolvimento das equações da etapa de predição do algoritmo EKF. Do ponto de vista das equações do modelo de medição, este trabalho adota o modelo proposto em [5], o qual foi também desenvolvido no LEIA-UnB.

A fim de apresentar as equações do EKF, este capítulo também discute alguns aspectos importantes sobre o tema do Filtro de Kalman (KF), sendo uma das ferramentas de estimação mais utilizadas na atualidade. A partir de sua recursividade, o KF consegue estimar o estado de um sistema dinâmico com presença de ruído. Este algoritmo gera estimativas dos valores reais de medição, assim como dos valores associados a estas medições. Além de disso, o algoritmo estima a incerteza do valor predito e calcula uma média ponderada entre os valores predito e medido, onde o maior peso é atribuído ao valor com menor grau de incerteza. Estas estimativas tendem a estar mais próximas dos valores reais do que as medições iniciais, devido a que a incerteza da filtragem é menor do que qualquer um dos valores individualmente [26].

A Figura 4.1 mostra a estrutura geral do Filtro de Kalman e suas duas fases: (a) predição e (b) estimação.

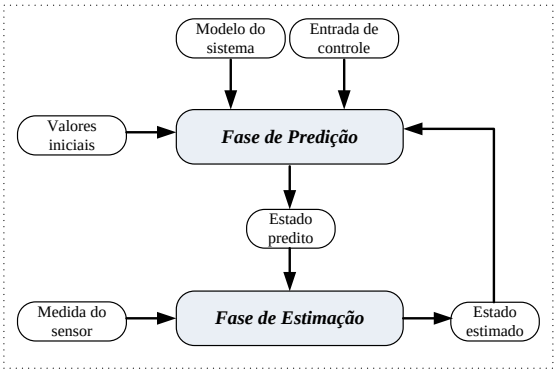


Figura 4.1. Estrutura Geral do Filtro de Kalman

A Filtragem de Kalman apresenta extensões e generalizações, como são os casos do Filtro de

Kalman Estendido ou EKF (do inglês ***E**xtended **K**alman **F**ilter*) e UKF (do inglês ***U**nscented **K**alman **F**ilter*). Para a tarefa de localização foi escolhida aquela relativa ao EKF (vide Capítulo 2), tendo em conta que:

(a) O algoritmo é orientado para sistemas dinâmicos não lineares;

(b) O algoritmo considera a técnica de fusão de sensores;

(c) O algoritmo servirá como base para o algoritmo SLAM, aplicado à área de micro-robótica nos futuros trabalhos.

Como foi mencionado no Capítulo 2, o Filtro de Kalman é um excelente estimador de estados para sistemas lineares; no entanto na realidade a maioria de sistemas são não-lineares o qual faz difícil sua aplicação prática. Este o caso de um robô móvel, o qual se movimenta em um ambiente determinado, onde se verifica a limitação do KF para estimar o estado seguinte de posição do robô. É aqui que o EKF se apresenta como uma opção a partir de seu enfoque de linearização de sistema.

Considera-se um sistema não-linear representado pelas Equações (4.1) e (4.2):

$$x_k = f(x_{k-1}, u_{k-1}, e_{k-1}), \tag{4.1}$$

$$z_k = h(x_k) + v_k, \tag{4.2}$$

onde $f()$ é uma função não-linear do *sistema do processo* e $h()$ uma função não- linear do *sistema de medição*. Além disso, essas funções podem ser usadas para propagar ambos o vetor de estados x_k e o vetor de saída z_k. Por outro lado, $f()$ também depende de um vetor de controle u_{k-1} e um vetor associado ao ruído do processo e_{k-1}, enquanto que v_k está associado ao ruído de medição.

Cabe mencionar que a principal abordagem do EKF é a *linearização* sobre o sistema. Assume-se uma função $f()$ não-linear, na qual a função densidade de probabilidade (ou *fdp*) projetada através desta função é tipicamente não Gaussiana, devido à não-linearidade em $f()$ gera uma distorção na forma Gaussiana do próximo estado. A partir da linearização em torno da média da Gaussiana obtemos uma aproximação de $f()$ com uma função linear. Logo ao projetar a *fdp* através desta aproximação linear se obtêm que a natureza Gaussiana da crença posterior se conserva. Uma vez feita a linearização, o modo de tratamento da crença do sistema obtido é equivalente à do KF simples. [26].

Com respeito à linearização, o EKF utiliza um método de *Expansão de Taylor* de primeira ordem [43], o qual constrói uma aproximação linear da função $f()$ a partir de seu valor e a sua primeira derivada parcial da função $f()$, como é mostrado na Equação 4.3.

$$f'(x_k, u_k, e_k) = \frac{\partial f(x_k, u_k, e_k)}{\partial x_k}, \tag{4.3}$$

As linearizações de $f()$ e $h()$ em relação às variáveis de estado e ao ruído do processo são usadas pelo algoritmo EKF, recebendo o nome de matrizes *Jacobianas* as quais são mostradas nas Equações 4.4, 4.5 e 4.6.

$$F = Df(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, \tag{4.4}$$

$$H = Dh(x) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \frac{\partial h_n}{\partial x_2} & \cdots & \frac{\partial h_n}{\partial x_n} \end{bmatrix}, \tag{4.5}$$

$$W = Df(e) = \begin{bmatrix} \frac{\partial f_1}{\partial e_1} & \frac{\partial f_1}{\partial e_2} & \cdots & \frac{\partial f_1}{\partial e_n} \\ \frac{\partial f_2}{\partial e_1} & \frac{\partial f_2}{\partial e_2} & \cdots & \frac{\partial f_2}{\partial e_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial e_1} & \frac{\partial f_n}{\partial e_2} & \cdots & \frac{\partial f_n}{\partial e_n} \end{bmatrix}, \tag{4.6}$$

Estas matrizes Jacobianas F e H não são constantes no EKF, no entanto se elas forem avaliadas em um valor específico do vetor de estados, $x = x_0$, as matrizes tornam-se constantes [43]. No caso discreto do EKF, a etapa da *predição* pode ser definida pelas Equações (4.7) e (4.8). No caso da etapa da *estimação*, a mesma fica estabelecida pelas Equações (4.9), (4.10) e (4.11).

Predição:

$$1^a : X_k^- = f(X_{k-1}^+, u_{k-1}, e_{k-1}), \tag{4.7}$$

$$2^a : P_k^- = F P_{k-1}^+ F^T + W Q_{k-1} W^T, \tag{4.8}$$

Estimação:

$$3^a : G_k = P_k^- H^T \left[H P_k^- H^T + R \right]^{-1}, \tag{4.9}$$

$$4^a : X_k^+ = X_k^- + G_k \left[Z_k - h(X_k^-) \right], \tag{4.10}$$

$$5^a : P_k^+ = P_k^- - G_k H P_k^-, \tag{4.11}$$

É possível observar que as Equações (4.7) a (4.11) incluem operações matriciais de adição, subtração, multiplicação, divisão e inversão, além de usar transpostas. A Tabela 4.1 exibe a notação empregada neste trabalho para o algoritmo EKF.

Tabela 4.1. Descrição dos símbolos usados no algoritmo EKF

Símbolos	**Descrição**
X	Pose do Robô
u	Entrada ou controle para Pose do Robô
e	Erro de entrada para Pose do Robô
P	Covariância da Pose do Robô
F	Jacobiana do Movimento do Robô
W	Jacobiana do Ruído do Movimento do Robô
Q	Covariância do Ruído do Movimento Permanente
G	Ganho do EKF
H	Jacobiana da Medição
R	Covariância do Ruído da Medição Permanente
Z	Medição do Sensor
f()	Função Não-Linear do Processo
h()	Função Não-Linear da Medição
k-1	Tempo/Estado Anterior
k	Tempo/Estado Atual
-	Informação *a priori* ou *predita*
+	Informação *a posteriori* ou *estimada*

4.2 Desenvolvimento das equações do Algoritmo Sequencial do EKF

Como foi descrito na seção anterior, as equações do algoritmo EKF demandam intenso cálculo computacional e tempo, devido ao fato de incluir operações matriciais tais como soma, subtração, multiplicação e inversão. Além disso, o aumento dimensional das matrizes intensifica este problema [44]. Por essa razão, optou-se por utilizar a versão sequencial do algoritmo EKF, onde se processa uma medida do sensor a cada instante de tempo [9]. A outra opção é trabalhar com todos os sensores no mesmo instante de tempo chamada neste caso como versão *cheia* (*full*) do algoritmo, mas só aumentaria a dimensão matricial nas equações.

Para conseguir o desenvolvimento do algoritmo em modo sequencial é necessário se ter o modelo do sistema de processo, descrito no Capítulo 3, o qual definirá a função não-linear $f()$, e da mesma maneira modelar o sistema de medição a fim de se obter a função $h()$.

4.2.1 Etapa de Predição do EKF

Já que o interesse do presente trabalho está focado em conhecer a posição (x , y), assim como a orientação angular θ do robô, então definiram-se as mesmas como as variáveis de estado (vide Equação (4.12)):

$$X_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{3\times 1} , \tag{4.12}$$

Baseado na Equação (3.16), e usando o método de integração de Euler, é possível obter o modelo de sistema discreto necessário para a 1^a equação EKF (Equação (4.7)), a qual pode ser escrita como mostrado na Equação (4.13):

$$1^a \textit{equação EKF:} \quad X_k^- = f(X_{k-1}^+, u_{k-1}, e_{k-1}),$$

$$\begin{bmatrix} x_k^- \\ y_k^- \\ \theta_k^- \end{bmatrix}_{3\times 1} = \begin{bmatrix} x_{k-1}^+ + \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\cos(\theta_{k-1}^+) - x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\sin(\theta_{k-1}^+)] \\ y_{k-1}^+ + \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\sin(\theta_{k-1}^+) + x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\cos(\theta_{k-1}^+)] \\ \theta_{k-1}^+ + \Delta T.[(\alpha.\frac{V_R-V_L}{D})] \end{bmatrix}_{3\times 1} , \tag{4.13}$$

onde ΔT é o tempo de amostragem e $u_k = [V_L, V_R]^T$ é o vetor de entrada (informação fornecida pelos *encoders*). Os erros das velocidades de deslocamento linear V_L e V_R podem ser definidos como V_{Le} e V_{Re} respectivamente (vide Equação (4.14)):

$$V_L = V_{Lc} + V_{Le}, \quad V_R = V_{Rc} + V_{Re}, \tag{4.14}$$

$$e_k = \begin{bmatrix} V_{Le} \\ V_{Re} \end{bmatrix}_{2\times 1}, \tag{4.15}$$

onde neste caso V_{Lc} e V_{Rc} são as velocidades lineares corretas de cada roda, sendo que o vetor de erro e_k (descrito na Equação (4.15)) representa as incertezas do modelo odométrico, assumindo um ruído do tipo Gaussiano com média zero.

No entanto, na Equação (4.13) a função não-linear $f(.)$ será descomposta em uma adição dos vetores x_{k-1}^+ e M_{k-1} (ver Equação (4.16)) devido a que este último vetor será usado no Capítulo 6.

$$X_k^- = X_{k-1}^+ + M_{k-1},$$

$$\begin{bmatrix} x_k^- \\ y_k^- \\ \theta_k^- \end{bmatrix}_{3\times 1} = \begin{bmatrix} x_{k-1}^+ \\ y_{k-1}^+ \\ \theta_{k-1}^+ \end{bmatrix}_{3\times 1} + \begin{bmatrix} \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\cos(\theta_{k-1}^+) - x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\sin(\theta_{k-1}^+)] \\ \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\sin(\theta_{k-1}^+) + x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\cos(\theta_{k-1}^+)] \\ \Delta T.[(\alpha.\frac{V_R-V_L}{D})] \end{bmatrix}_{3\times 1}, \tag{4.16}$$

onde M_{k-1} é representado pela Equação (4.17).

$$M_{k-1} = \begin{bmatrix} \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\cos(\theta_{k-1}^+) - x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\sin(\theta_{k-1}^+)] \\ \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\sin(\theta_{k-1}^+) + x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\cos(\theta_{k-1}^+)] \\ \Delta T.[(\alpha.\frac{V_R-V_L}{D})] \end{bmatrix}_{3\times 1}, \tag{4.17}$$

A seguir o desenvolvimento da 2^a equação EKF (Equação (4.8)) é tratado. Para isso, devem ser obtidas as matrizes Jacobianas F e W, cujas representações são mostradas nas Equações (4.18) e (4.19).

$$F = \frac{\partial f()}{\partial X_k^+} = \begin{bmatrix} \frac{\partial f_1}{\partial x_k^+} & \frac{\partial f_1}{\partial y_k^+} & \frac{\partial f_1}{\partial \theta_k^+} \\ \frac{\partial f_2}{\partial x_k^+} & \frac{\partial f_2}{\partial y_k^+} & \frac{\partial f_2}{\partial \theta_k^+} \\ \frac{\partial f_3}{\partial x_k^+} & \frac{\partial f_3}{\partial y_k^+} & \frac{\partial f_3}{\partial \theta_k^+} \end{bmatrix}_{3\times 3} = \begin{bmatrix} 1 & 0 & \Delta T.[-(\alpha.\frac{V_R+V_L}{2}).\sin(\theta_k^+) - x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\cos(\theta_k^+)] \\ 0 & 1 & \Delta T.[(\alpha.\frac{V_R+V_L}{2}).\cos(\theta_k^+) - x_{CIR}.(\alpha.\frac{V_R-V_L}{D}).\sin(\theta_k^+)] \\ 0 & 0 & 1 \end{bmatrix}_{3\times 3}, \tag{4.18}$$

$$W = \frac{\partial f()}{\partial e_k} = \begin{bmatrix} \frac{\partial f_1}{\partial V_{Le}} & \frac{\partial f_1}{\partial V_{Re}} \\ \frac{\partial f_2}{\partial V_{Le}} & \frac{\partial f_2}{\partial V_{Re}} \\ \frac{\partial f_3}{\partial V_{Le}} & \frac{\partial f_3}{\partial V_{Re}} \end{bmatrix}_{3\times2} = \Delta T.\alpha. \begin{bmatrix} [\frac{\cos(\theta_k^+)}{2} + x_{CIR}.\frac{\sin(\theta_k^+)}{D}] & [\frac{\cos(\theta_k^+)}{2} - x_{CIR}.\frac{\sin(\theta_k^+)}{D}] \\ [\frac{\sin(\theta_k^+)}{2} - x_{CIR}.\frac{\cos(\theta_k^+)}{D}] & [\frac{\sin(\theta_k^+)}{2} + x_{CIR}.\frac{\cos(\theta_k^+)}{D}] \\ [\frac{-1}{D}] & [\frac{1}{D}] \end{bmatrix}_{3\times2}, \tag{4.19}$$

As outras matrizes necessárias para a implementação da 2^a equação EKF estão descritas nas Equações (4.20) e (4.21).

$$P_k = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta^2 \end{bmatrix}_{3\times3}, \tag{4.20}$$

$$Q = \begin{bmatrix} \sigma_{V_L}^2 & 0 \\ 0 & \sigma_{V_R}^2 \end{bmatrix}_{2\times2}, \tag{4.21}$$

onde P é a matriz de covariância de erro de posição do robô e Q representa a matriz de covariância de ruído do processo permanente.

4.2.2 Etapa de Estimação do EKF

Na etapa estimativa do algoritmo EKF, aplicado ao problema de localização em robótica móvel, considera-se que o robô se movimenta em um ambiente especifico. A Figura 4.2 descreve o robô Pioneer 3AT posicionado no sistema global com orientação θ, sendo que a representação dos símbolos é mostrada na Tabela 4.2.

Para projetar esta etapa do EKF, vai se supor que é conhecido o mapa do ambiente onde o robô está localizado. Neste caso, o mapa tem a forma específica de uma linha (por exemplo, uma parede), como mostrado na Figura 4.3.

Para cada medição do *feixe* de laser do sensor LRF (vide Figura 4.3), os dados são equiparados (*matched*) com o mapa. Dado que foi implementada a versão sequencial do algoritmo EKF, as medições foram realizadas em forma também sequencial. Isto significa que uma estimação do vetor de estados é efetuada por cada medição. Desta forma, o tamanho das matrizes são reduzidas, assim como a complexidade da implementação em *hardware*. As Equações (4.22) e (4.23) representam exemplos de medições sequenciais e cheias, respectivamente, para o vetor Z_k associado ao sistema de medição.

$$Z_k = \begin{bmatrix} u_i \\ v_i \end{bmatrix}_{2\times1}, \tag{4.22}$$

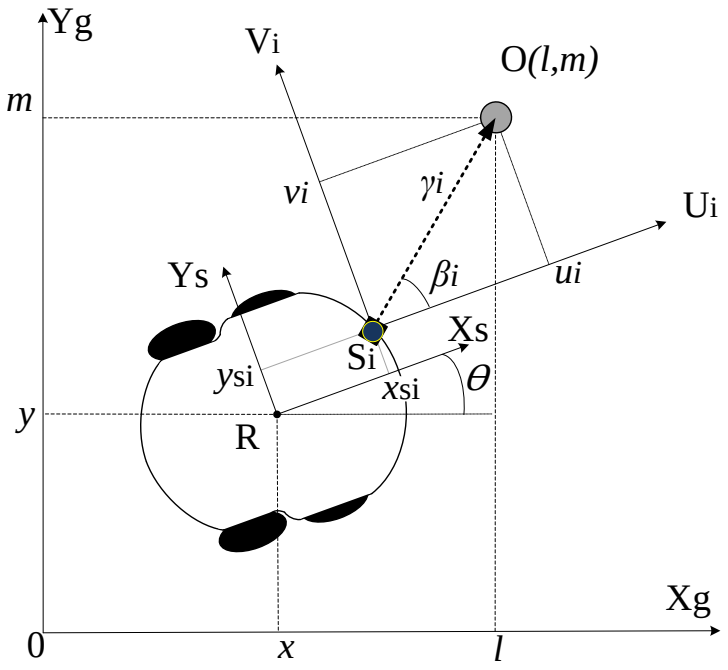


Figura 4.2. Robô Pioneer 3-AT posicionado em um sistema de coordenadas de duas dimensões

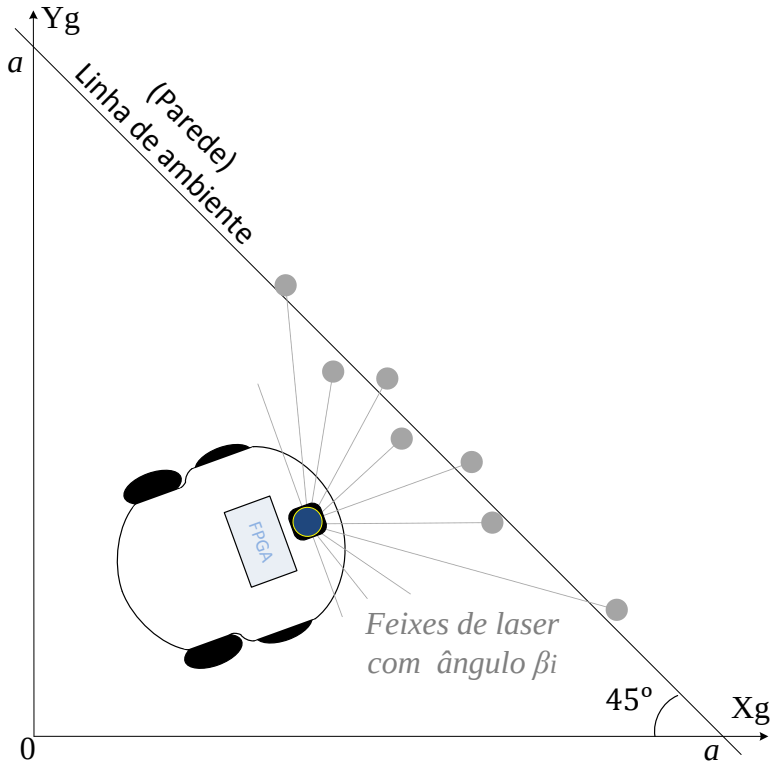


Figura 4.3. Robô Pioneer 3-AT posicionado em um ambiente de linha

Tabela 4.2. Descrição dos símbolos usados na representação da posição do Pioneer 3-AT

Símbolos	Descrição
X-Y	Sistema de Coordenadas Global - SCG
(x,y)	Posição do Robô no SCG
θ	Orientação do Robô no SCG
Xs-Ys	Sistema de Coordenadas do Robô - SCR
Xsi-Ysi	Posição do Sensor i no SCR
Ui-Vi	Sistema de Coordenadas do Sensor Si - SCSi
γ_i	Distância entre o objeto O e o sensor Si
β_i	Ângulo de medição
(ui,vi)	Posição do objeto O no SCSi
(l,m)	Posição do objeto O no SCG

$$Z_k = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{bmatrix}_{2n\times 1}, \tag{4.23}$$

Em cada iteração do algoritmo, a posição global do objeto detectado O (*feature*) é calculada a partir do estado predito X_k^-, e logo enviada ao algoritmo para seu processamento. As coordenadas deste ponto O são l no eixo-Xg e m no eixo-Yg como mostra a Figura 4.4. A através de uma transformação de coordenadas do objeto $O - feature$, do sistema de coordenadas SCSi ($Ui - Vi$) para o SCG ($Xg - Yg$), é possível descobrir estes parâmetros (l, m), cujas representações são dadas pelas Equações (4.24) e (4.25):

$$\begin{aligned} l_k = & \frac{a - (y_k^- + x_{si}.\sin(\theta_k^-) + y_{si}.\cos(\theta_k^-))}{\tan(\theta_k^- + \beta_i) + 1} + \\ & + \frac{(x_k^- + x_{si}.\cos(\theta_k^-) - y_{si}.\sin(\theta_k^-)).\tan(\theta_k^- + \beta_i)}{\tan(\theta_k^- + \beta_i) + 1}, \end{aligned} \tag{4.24}$$

$$m_k = a - l_k, \tag{4.25}$$

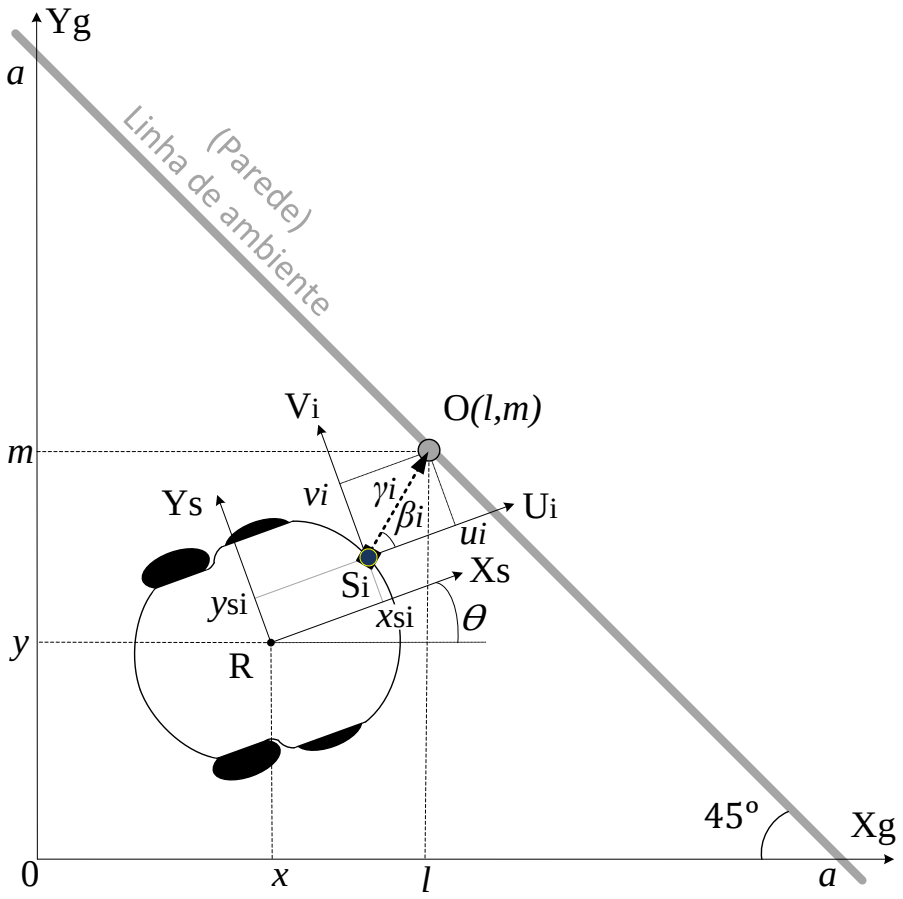


Figura 4.4. Robô, ambiente de linha e coordenadas a considerar

onde a esta relacionado ao ponto (a,0) no sistema de coordenadas SCG, correspondente à interseção da linha de ambiente com o eixo-Xg. É importante indicar que as expressões das Equações (4.24) e (4.25) são somente para um mapa de ambiente especifico da forma da Figura 4.4. Esta forma de ambiente foi escolhida só para efeito de validação inicial do sistema proposto para aplicações de localização robótica. Para outro tipo de mapas de ambiente deve ser feito o recálculo das expressões dos parâmetros l_k e m_k.

Por outro lado, R é a matriz de ruído permanente da medição, sendo intrínseca de cada sensor, que neste caso será o *ladar* (LRF). A mesma pode ser representada pela variância da medição em cada eixo do SCSi (vide Equação (4.26)).

$$R = \begin{bmatrix} \sigma_{ui}^2 & 0 \\ 0 & \sigma_{vi}^2 \end{bmatrix}_{2\times 2}, \tag{4.26}$$

Em seguida, para resolver o problema de localização, usando a versão sequencial, foi realizada uma transformação de coordenadas do objeto O, do sistema de coordenadas SCG ($Xg-Yg$) para

o SCSi ($Ui - Vi$), para desta maneira obter a função não-linear $h(X_k^-)$, a qual está representada pelas Equações (4.27) e (4.28).

Modelo do Sistema de Medição:

$$h(X_k^-) = \left\{ \begin{bmatrix} cos(\theta_k^-) & sin(\theta_k^-) \\ -sin(\theta_k^-) & cos(\theta_k^-) \end{bmatrix}_{2\times2} \cdot \left[\begin{pmatrix} l_k \\ m_k \end{pmatrix}_{2\times1} - \begin{pmatrix} x_k^- \\ y_k^- \end{pmatrix}_{2\times1} \right] \right\} - \begin{bmatrix} x_{si} \\ y_{si} \end{bmatrix}_{2\times1}, \tag{4.27}$$

ou,

$$h(X_k^-) = \begin{bmatrix} (l_k - x_k^-) \cdot cos(\theta_k^-) + (m_k - y_k^-) \cdot sin(\theta_k^-) - x_{si} \\ (m_k - y_k^-) \cdot cos(\theta_k^-) - (l_k - x_k^-) \cdot sin(\theta_k^-) - y_{si} \end{bmatrix}_{2\times1}, \tag{4.28}$$

A matriz Jacobiana H foi obtida aplicando a Equação (4.29), dando como resultado a Equação (4.30).

$$H = \frac{\partial h()}{\partial X_k} = \begin{bmatrix} \frac{\partial h_1}{\partial x} & \frac{\partial h_1}{\partial y} & \frac{\partial h_1}{\partial \theta} \\ \frac{\partial h_2}{\partial x} & \frac{\partial h_2}{\partial y} & \frac{\partial h_2}{\partial \theta} \end{bmatrix}_{2\times3}, \tag{4.29}$$

$$H = \begin{bmatrix} -cos(\theta_k) & -sin(\theta_k) & -(l_k - x_k) \cdot sin(\theta_k) + (m_k - y_k) \cdot cos(\theta_k) \\ sin(\theta_k) & -cos(\theta_k) & -(l_k - x_k) \cdot cos(\theta_k) - (m_k - ty_k) \cdot sin(\theta_k) \end{bmatrix}_{2\times3}, \tag{4.30}$$

Finalmente, a Tabela 4.3 exibe as dimensões das matrizes utilizadas no algoritmo sequencial EKF.

4.3 Conclusões do Capítulo

Neste capítulo foram explicados os conceitos de Filtro de Kalman Estendido (EKF) aplicado ao problema de localização robótica, assim como o modo do algoritmo que será implementado (em *hardware*) neste trabalho. O modo escolhido para abordar este problema foi a versão sequencial do mesmo (EKF sequencial), com a ideia de reduzir a dimensão das matrizes e facilitar a implementação em *hardware*.

Com respeito à obtenção das equações do algoritmo EKF, a etapa de *predição* foi baseada no modelo do robô desenvolvido no Capítulo 3, quanto a etapa de *estimação* foi baseada no modelo de medição desenvolvido em [5].

Tabela 4.3. Dimensão matricial dos símbolos usados no algoritmo sequencial EKF

Símbolos	**Dimensão Matricial**
X	3x1
u	2x1
e	2x1
P	3x3
F	3x3
W	3x2
Q	2x2
G	3x2
H	2x3
R	2x2
Z	2x1
$f()$	2x1
$h()$	2x1

Para a execução correta do algoritmo, considera-se que robô se movimenta em um ambiente de mapa conhecido previamente e que o processamento dos dados (informação dos sensores) será em linha (*online*). Esta será à abordagem para solucionar o problema de localização, que será discutida com maior detalhe no Capítulo 7.

5 Aspectos sobre FPGAs e das Unidades em Ponto Flutuante utilizadas neste trabalho

5.1 Introdução

Como foi discutido no Capítulo 1, este trabalho está focado na implementação de arquiteturas de *hardware* para o problema de localização em robótica móvel. Para tanto, foi escolhida uma plataforma reconfigurável, baseada em um FPGA da Altera. Entretanto, devido a que FPGAs atuais não incluem operadores aritméticos em ponto flutuante, para esta tarefa deveram ser utilizadas bibliotecas de ponto flutuante previamente projetadas em trabalhos anteriores desenvolvidos no Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados (LEIA-UnB).

A escolha da aritmética de ponto flutuante para representação numérica justifica-se levando em conta os requerimentos de precisão e alta faixa dinâmica presentes em diversos campos da engenharia, sendo este também o caso da robótica, onde além dos requerimentos mencionados anteriormente se tem que lidar com o fato de tratar com grandes volumes de dados provenientes de diferentes sensores quase em tempo real, o qual aumenta a criticidade. Desta forma, aplicações embarcadas para robótica podem contar com cálculos de alto desempenho em termos de precisão, faixa dinâmica, entre outros aspectos. Um formato de alta precisão indica poucos erros de quantização na implementação final, enquanto que um formato com baixa precisão indica implementações de alta velocidade e reduções em área e consumo de potência [45]. Por outro lado, o termo de faixa dinâmica, refere-se ao fato de conseguir representar números reais muito pequenos e muito grandes para operações aritméticas [3].

Em relação a tamanho da palavra, é comum encontrar na literatura implementações de *hardware* em ponto flutuante de 32 e 64 *bits*. A escolha do tamanho de palavra usado para o presente trabalho aritmética foi de 32 *bits* pelas seguintes razões: (a) é a mais utilizada dentro da comunidade cientifica e (b) apresenta baixo consumo de recursos. É importante também comentar, que existem implementações usando tamanho de palavras de 27 *bits*, o quais oferecem uma faixa dinâmica similar com implementações de 32 *bits*, no entanto apresentam um aumento

no erro associado comparado com às de 32 *bits*, como é analisado em [3], gerando outra razão para o uso deste tamanho de palavra (32 *bits*).

Em seguida, são apresentados conceitos sobre FPGAs e lógica reconfigurável, e a seguir uma descrição de uma biblioteca de operações aritméticas em ponto flutuante é apresentada, a qual foi desenvolvida previamente em diferentes trabalhos de pesquisa feitos no laboratório LEIA-UnB, e que foram utilizadas e adaptadas durante o desenvolvimento deste trabalho.

5.2 Arquiteturas Reconfiguráveis FPGA

Os FPGAs (do inglês ***F**ield **P**rogrammable **G**ate **A**rrays*) são dispositivos lógicos que permitem a sua reconfiguração após da sua fabricação, os quais contém componentes lógicos programáveis chamados de elementos lógicos (LEs, do inglês ***L**ogical **E**lements*), conectados via interconexões reconfiguráveis hierárquicas [46].

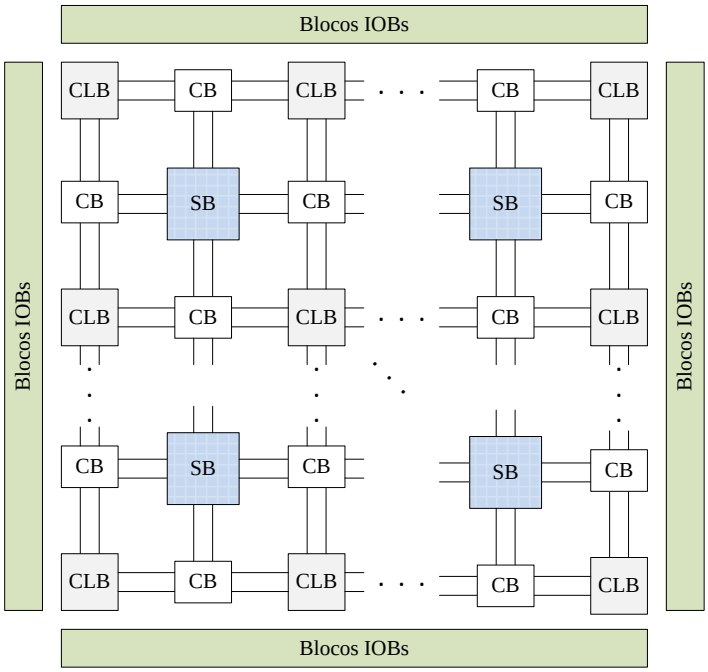


Figura 5.1. Estrutura geral de um FPGA

A estrutura geral de um FPGA está ilustrada na Figura 5.1. onde os três componentes básicos são: blocos lógicos configuráveis (CLBs, do inglês ***C**onfigurable **L**ogic **B**locks*), blocos de entrada e saída (IOBs, do inglês ***I**nput **O**utput **B**locks*) e interconexões reconfiguráveis. Um CLB consiste em uma matriz de comutação configurável, alguns circuitos de seleção tais como multiplexores, e *flip-flops*. Os IOBs são circuitos responsáveis pelo interfaceamento das saídas provenientes dos CLBs. Os mesmos são basicamente *buffers*, que funcionaram como um porto bidirecional de entrada/saída do FPGA.

As interconexões reconfiguráveis podem ser classificadas basicamente em dois grupos: (a) blocos de conexão (CB, do inglês ***C**onnection **B**lock*) e (b) blocos de comutação (SB, do inglês ***S**witching **B**lock*), mostrados na Figura 5.1. Os CBs permitem que as entradas e saídas dos blocos lógicos sejam atribuídas a blocos de entrada e saída (IOBs). Por outro lado os SBs permitem que um sinal de um IOB se ligue a outro IOB. Geralmente, a configuração é estabelecida por programação interna das células de memória estática, que determinam funções lógicas e conexões internas implementadas no FPGA entre os CLBs e os IOBs. O processo de escolha das interconexões é chamado de roteamento [47].

Atualmente, os FPGAs mais modernos incorporam blocos de DSPs (Digital Signal Processors), de gestão de relógios e RAMs (do inglês ***R**andom **A**ccess **M**emory*), estas últimas também conhecidas como BRAMs (do inglês ***B**lock **RAM**s*), presentes por exemplo nas plataformas Xilinx.

A seguir, é possível citar como principais vantagens dos FPGAs:

(a) Velocidade de processamento - implementações baseadas em fluxo de dados (ao invés de fluxo de instruções) e executadas diretamente em *hardware*;

(b) Flexibilidade em *hardware* - Alto poder de ajuste a funções específicas, mediante reconfiguração estática ou dinâmica (arquiteturas reconfiguráveis);

(c) Baixo custo;

(d) Rápido desenvolvimento de protótipos;

5.3 Unidades em Ponto Flutuante - (UPFs)

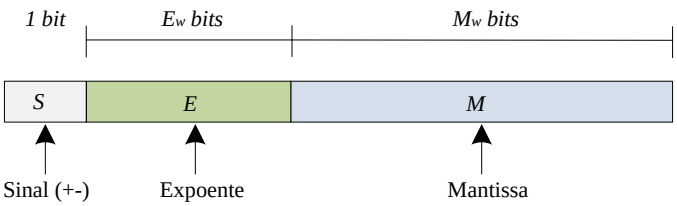


Figura 5.2. Estrutura da representação numérica no padrão IEEE-754

Nesta seção apresenta-se uma breve descrição das arquiteturas em *hardware* para operações aritméticas em ponto flutuante de precisão simples desenvolvidas por [48, 49, 4]. Estas unidades de operações aritméticas em ponto flutuante utilizaram a representação padrão IEEE-754 da

Figura 5.2). Este padrão permite ao projetista trabalhar tanto com precisão simples (32 *bits*) e dupla (64 *bits*) como também com outros formatos adaptados aos requerimentos de precisão de faixa dinâmica de uma aplicação especifica, como o caso da representação de 27 *bits* [3].

É importante ressaltar que as bibliotecas de operações aritméticas em ponto flutuante foram concebidas para plataformas Xilinx, por o qual em [5] foram realizadas algumas modificações, especificamente na arquitetura de soma/subtração para adequá-la aos FPGAs da Altera. O código original realizava uma operação de deslocamento de *bits* usando os operadores *sll* e *srl*. Devido à incompatibilidade dos operadores com o tipo de sinal usado no código, foram criados dois processos para deslocamento de *bits* (um à direta e outro à esquerda) como mostra a Figura 5.3.

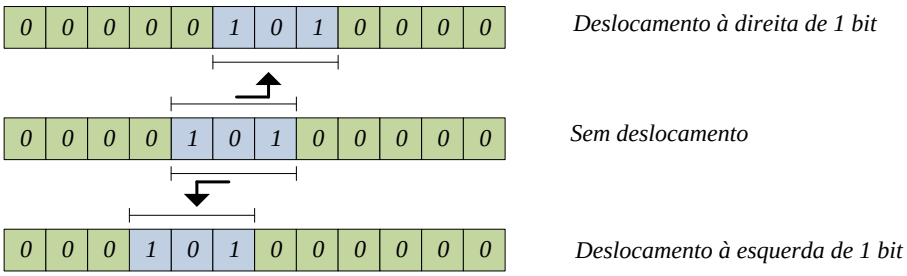


Figura 5.3. Exemplo de uma operação de deslocamento de *bit*

Esses deslocamentos fazem parte do processo para calcular o alinhamento das mantissas segundo a diferença dos expoentes (correspondente à representação padrão IEEE-754 da Figura 5.2), ou seja, dependo do valor da diferença entre os expoentes dos operandos A e B, este será o número de *bits* deslocados. Com essa correção, obteve-se sucesso na compilação dos códigos para plataformas da Altera.

5.3.1 Somador/Subtrator em Ponto Flutuante

O Somador/Subtrator em ponto flutuante apresenta um conjunto de passos, os quais são os seguintes:

(a) Separar as entradas sinal, exponente e mantissa, para logo verificar se elas são zero, infinito ou uma representação não correspondente ao padrão IEEE-754. Logo adicionar o *bit* escondido (ou *hidden*) à mantissa.

(b) Comparar as duas entradas: uma operação lógica de deslocamento à direita é feita sobre

o menor dos dois números. O número de *bits* correspondentes à mantissa deslocados à direita, baseado na diferença do exponente, e essa diferença é o resultado preliminar do cálculo do exponente. Finalmente, somar/subtrair as mantissas atuais.

(c) Logo, se o *bit* mais significativo (MSB, do inglês ***M**ost **S**ignificant **B**it*) MSB é 1, então não é necessária normalização. Do contrario é preciso deslocar à esquerda a mantissa alcançada até que seu MSB seja 1. Cada deslocamento feito deve estar acompanhado de um decremento do exponente atual em 1. Como ultimo passo, concatenar o sinal, exponente e mantissa para ter o resultado final.

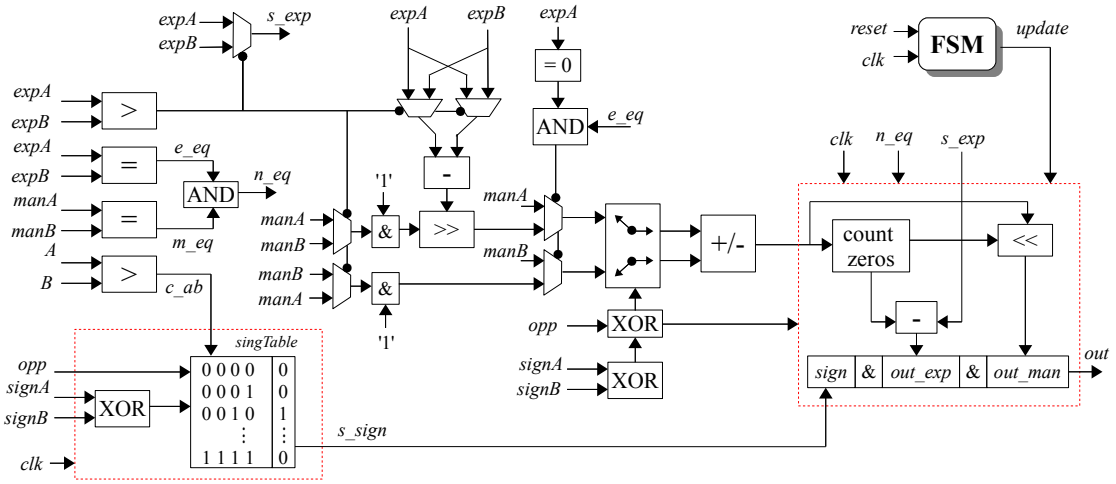


Figura 5.4. Arquitetura em *hardware* da unidade soma/subtração *FPadd*.
Fonte: Muñoz, 2012 [3]

A arquitetura em *hardware* da unidade somador/subtrator (*FPadd*) é descrita na Figura (5.4). Nela os blocos em linha pontilhada correspondem a processos sensíveis ao relógio. Essa arquitetura faz uso de dois ciclos de relógio para realizar a operação de soma/subtração em aritmética de ponto flutuante. No primeiro ciclo de relógio é feito o primeiro passo correspondente à verificação dos argumentos de entrada (se são zero, infinito ou uma representação padrão IEEE-754 não válida). As comparações em paralelo também são feitas neste ciclo de relógio. Finalmente, no segundo ciclo do relógio é realizado o ultimo passo de normalização e concatenação. Neste caso, uma FSM controla o processo de atualização do resultado final, o qual é registrado usando *flip-flops* [3].

5.3.2 Multiplicador em Ponto Flutuante

No caso da multiplicação em ponto flutuante, os passos são os seguintes:

(a) Separar as entradas sinal, exponente e mantissa, para logo verificar se elas são zero, infinito

ou uma representação não correspondente ao padrão IEEE-754. Logo adicionar o *hidden bit* à mantissa.

(b) Multiplicar as mantissas, somar exponentes e determinar o sinal produto.

(c) Logo, é feita normalização y concatenação do sinal, exponente e mantissa dos resultados finais, similar ao terceiro passo do operador Somador/Subtrator explicado anteriormente.

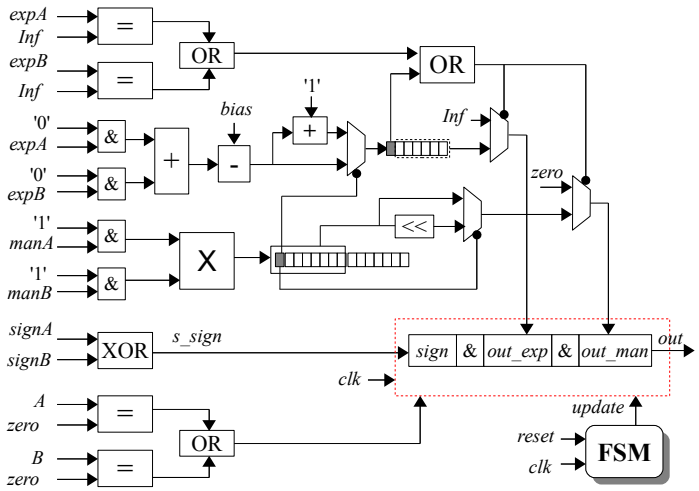


Figura 5.5. Arquitetura em *hardware* da unidade multiplicação *FPmul*.
Fonte: Muñoz, 2012 [3]

A Figura 5.5 descreve a arquitetura de *hardware* da unidade de multiplicação (nomeada *FPmul*), o qual é feito em dois ciclos de relógio. No primeiro ciclo, é realizada a identificação e verificação dos operandos com o objetivo de saber se são zero, infinito ou uma representação padrão IEEE-754 não válida. Além disso, o sinal, expoente preliminar e multiplicação das mantissas são computadas. A operação lógica *xor* é usada para comparar o sinal dos argumentos de entrada para determinar o resultado do sinal. É importante notar aqui que os expoentes são adicionados com um *bit* extra indicando *overflow*. Portanto, o valor resultante tem uma polarização dupla e o expoente preliminar é calculado subtraindo-se o *bias* da adição do expoente. Adicionalmente nesta arquitetura a mantissa de M_w *bits* com o *bit* implícito são multiplicadas, resultando em uma palavra de tamanho $2(M_w + 1)$ que é truncada nos primeiros $M_w + 2$ *bits* [3].

No segundo ciclo de relógio, é avaliado se o MSB do resultado da multiplicação das mantissas é 1. Este *bit* controla um multiplexador que aborda o resultado da mantissa final. Além disso, se o MSB do resultado da multiplicação das mantissas é um expoente preliminar, então deve ser adicionado por 1. Seguidamente, o resultado do expoente é avaliado com o objetivo de identificar uma possível representação de um número infinito, de não ser assim, se procede à concatenar os resultados de expoente e mantissa. Similar à Subseção 5.3.1, o processo de atualização do

resultado final é controlado a partir de uma FSM [3].

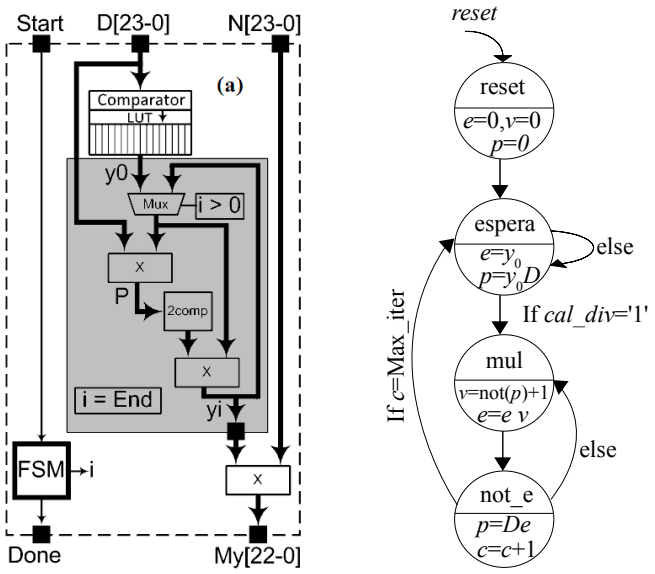
5.3.3 Divisão em Ponto Flutuante

Um algoritmo generalizado para calcular a divisão inclui os passos de abaixo.

(a) Seja X e Y números reais representados no padrão IEEE-754, onde X representa o dividendo e Y o divisor.

(b) Separar as entradas sinal, exponente e mantissa, para logo verificar se elas são zero, infinito ou uma representação inválida no padrão IEEE-754. Adicionar o *bit hidden* à mantissa.

(c) Calcular o resultado da mantissa usando o algoritmo de Newton-Raphson para divisão. Paralelo a isso, se deve verificar do resultado *Expoente* (X) - *Expoente* (Y) + *Bias*, e avaliar o sinal do resultado.



(a) arquitetura (b) FSM

Figura 5.6. Arquitetura em *hardware* da unidade de divisão baseado no algoritmo Newton-Raphson. Fonte: Muñoz, 2012 [3]

Como foi mencionado anteriormente, para a operação de divisão será usado o algoritmo de Newton-Raphson, o qual assume duas entradas N e D de $n - bits$, que satisfaçam $1 \leq N$, $D < 2$, a partir de uma aproximação inicial para $y_0 = 1/D$. As Equações 5.1a e 5.1b devem ser executadas de uma forma iterativa [50].

$$p = Dy_i \tag{5.1a}$$

$$y_{i+1} = y_i(2 - p) \tag{5.1b}$$

Após a $i-ésima$ iteração, multiplicando $N \cdot y_{i+1}$ é encontrada uma aproximação para N/D. A Figura 5.6 mostra a arquitetura em *hardware* deste algoritmo com sua respectiva descrição de FSM. [3].

5.3.4 CORDIC

O algoritmo CORDIC (do inglês ***Co**ordinate **R**otation **Di**gital **C**omputer*) é um método iterativo baseado em deslocamento-adição (*add-shift*) com o objetivo de calcular a rotação de um vetor bidimensional e o comprimento e ângulo de um vetor. O CORDIC é um algoritmo simples e eficiente para calcular funções hiperbólicas e trigonométricas. Segundo [4], as equações do CORDIC iterativo podem enunciadas como segue:

$$X_{i+1} = X_i - m \cdot \sigma_i \cdot 2^{-i} \cdot Y_i \tag{5.2}$$

$$Y_{i+1} = Y_i + \sigma_i \cdot 2^{-i} \cdot X_i \tag{5.3}$$

$$Z_{i+1} = Z_i - \sigma_i \cdot \theta_i \tag{5.4}$$

onde, i é o índice de interação, θ_i são micro-rotações pré computadas, σ_i é a direção da rotação e m indica o tipo de coordenada.

Das Equações (5.2) a (5.4), é possível notar que cada iteração do algoritmo precisa computar três operações em ponto flutuante soma/subtração e uma busca na memória ROM (do inglês ***R**ead **O**nly **M**emory*), usado para armazenar as micro-rotações pré computadas. A arquitetura em *hardware* do algoritmo, nomeada FP-CORDIC, apresenta condições iniciais diferentes para cada modo de funcionamento, já seja como função *seno*, *cosseno* e *arco-tangente* [3].

Com respeito à arquitetura do algoritmo, ela é composta por quatro unidades: *redução de argumento*, *máquina de estados finitos*, *micro-rotação* e *normalização*. A estrutura base da unidade de micro-rotação são três unidades *FPadd* paralelas (para soma e subtração, vide Figura 5.7). As operações 2^{-n} são feitas subtraindo o índice n da micro-rotação atual para o expoente (E) de X e Y. Após, é realizada uma concatenação com a mantissa e o sinal de X e Y. As condições iniciais para as variáveis X, Y e Z, dependo do modo de operação desejada (função

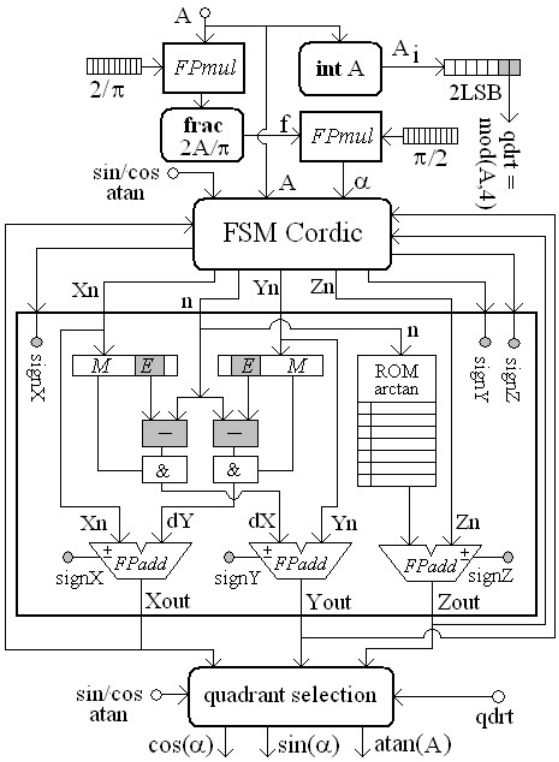


Figura 5.7. Arquitetura de *hardware* FP-CORDIC para cálculo das funções *sin*, *cos* e *atan*. Fonte: Muñoz, 2010 [4]

seno, *cosseno* ou *arco-tangente*) são determinadas pela unidade *FSM*. Esta unidade também controla o processo de retorno e o sentido das rotações, enquanto, um sinal *ready* indica uma saída válida após N micro-rotações [4].

Adicionalmente, tem-se uma unidade dedicada à redução do argumento ao quadrante para os cálculos de *seno* e *cosseno* usando as seguintes expressões matemáticas $\alpha = \delta\pi/2$ e $\delta = (2\theta/\pi) - \kappa$. Nesta unidade, o argumento A é multiplicado por $2/\pi$ e então, a unidade *fracFP* separa as partes inteira (i) e fracional (f), para logo, a partir do cálculo $\delta\pi/2$ obter o argumento reduzido. Terminada a iteração N, a unidade *normalização* realiza a seleção do quadrante no caso do modo de rotação (*seno* e *cosseno*). Lembrar que os dois *bits* menos significativos (LSB, do inglês ***L**east **S**ignificant **B**it*) da parte inteira (i) do argumento reduzido correspondem ao valor $mod(k, 4)$, que é usado para endereçar as saídas de um dos seguinte valores: $sin(a), cos(a)$ [4].

5.4 Conclusões do Capítulo

Neste capítulo foi apresentada uma descrição básica sobre os FPGAs, indicando as suas principais vantagens para uso em sistemas embarcados, bem como as arquiteturas de *hardware* das unidades de cálculo aritmético e trigonométrico em ponto flutuante necessárias para este

trabalho (soma/subtração, multiplicação, divisão e *seno*/*cosseno*).

Além disso, foram detalhadas as alterações realizadas no código das UPFs para seu correto funcionamento em FPGAs da Altera, já que as mesmas foram desenvolvidas inicialmente para plataformas da Xilinx.

Adicionalmente, foi discutido o tema de precisão nas operações aritméticas necessárias para a área de robótica em geral. Pelo uso da representação em ponto flutuante de 32 *bits*, fica garantida a precisão necessária nas operações aritméticas.

6 Metodologia Proposta

Neste capítulo são apresentadas duas metodologias para implementar o algoritmo EKF orientado ao problema de localização em robótica móvel. Para tanto, se desenvolveu uma a arquitetura para a etapa de *predição* do EKF e adicionalmente foi usado como ponto de partida a arquitetura da etapa de *estimação* desenvolvida em [5], fazendo uma extensão da mesma com objetivo de desenvolver uma arquitetura em *hardware* unificada (*predição* + *estimação*) do algoritmo EKF aplicado a localização robótica. O sistema embarcado está conformado por um processador Nios II, o barramento Avalon, módulos de *hardware* propostos e outros módulos auxiliares.

Cabe mencionar que a partir dos trabalhos correlatos discutidos (vide Capítulo 2), foi possível concluir que ainda não se desenvolveu na literatura em plataformas FPGAs uma abordagem sequencial do algoritmo EKF, que considere ambas etapas do filtro (*predição* e *estimação*), usando representação em ponto flutuante de 32 *bits* aplicado a localização em robótica móvel, e que trabalhe com processamento de dados *online*.

6.1 Primeira abordagem para a solução do problema de localização: Implementação em FPGA usando módulos de *hardware* individuais para cada etapa EKF

Nesta seção, é apresentada uma arquitetura denominada de *módulo de hardware* para a etapa de *predição* (Equações (4.7),(4.8)), que utiliza uma representação em ponto flutuante para os cálculos matriciais intensivos do EKF sequencial. Este módulo foi conectado a outro módulo correspondente à etapa de *estimação* (vide Equações (4.9),(4.10)e(4.11))previamente desenvolvido em [5], permitindo assim a implementação completa do algoritmo EKF em *hardware*.

Neste caso, tanto as matrizes Jacobianas quanto a função não linear $h()$ não estão mapeadas em *hardware*; pois seus elementos são previamente calculados em *software* (usando o processador Nios II e a arquitetura CORDIC), sendo endereçados às arquiteturas em *hardware*. O cálculo do *seno* e *cosseno*, neste caso, é realizado pela arquitetura CORDIC (vide Capítulo 5, Subseção

5.3.4) para acelerar o computo das matrizes M, F, W, H e $h()$. Estas matrizes são recalculadas para cada iteração do algoritmo EKF. A Figura (6.1) mostra a implementação em FPGA usando módulos individuais associado a cada etapa do filtro EKF.

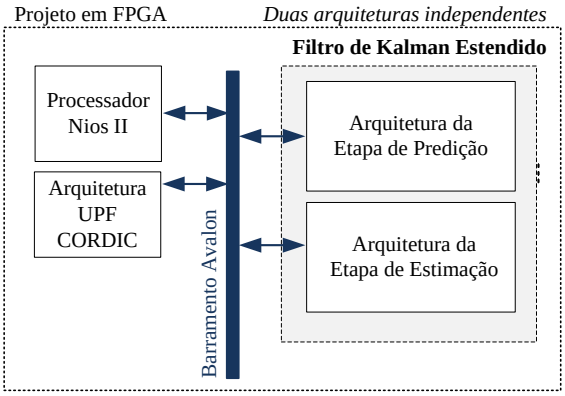


Figura 6.1. Primeira implementação em FPGA com duas arquiteturas independentes: *Predição e Estimação*

6.1.1 A Arquitetura da Etapa de *Predição*

A etapa de *predição* inclui basicamente operações matriciais, tais como multiplicação e adição (veja Equações (4.7) e (4.8)). As Equações (6.1) a (6.6) descrevem a decomposição do processo em seis passos, e para atingi-lo como um todo uma FSM foi projetada, a fim de controlar as operações em ponto flutuante envolvidas na arquitetura em *hardware*.

$$Passo\ 1 : X_k^- = X_{k-1}^+ + M_{k-1} \tag{6.1}$$

$$Passo\ 2 : O_{3\times3}^1 = P_{k-1}^+ \cdot F^T \tag{6.2}$$

$$Passo\ 3 : O_{2\times3}^2 = F \cdot O_{3\times3}^1 \tag{6.3}$$

$$Passo\ 4 : O_{2\times3}^3 = Q \cdot W^T \tag{6.4}$$

$$Passo\ 5 : O_{3\times3}^4 = W \cdot O_{2\times3}^3 \tag{6.5}$$

$$Passo\ 6 : P_k^- = O_{3\times3}^2 + O_{3\times3}^4 \tag{6.6}$$

onde a 1^a equação EKF esta representa pela Equação (6.1). Logo, as Equações (6.2) a (6.5) são os passos correspondentes para computar a 2^a equação EKF.

Por exemplo, pode ser observado na Equação (6.2) que o primeiro passo para computar a 2^a equação EKF é uma multiplicação de matrizes ($P_{k-1}^+ \cdot F^T$), como mostrado na Equação (6.7).

$$O^1_{3\times3} = \begin{bmatrix} p^+_{11} & p^+_{12} & p^+_{13} \\ p^+_{21} & p^+_{22} & p^+_{23} \\ p^+_{31} & p^+_{32} & p^+_{33} \end{bmatrix}_{3\times3} \cdot \begin{bmatrix} F^T_{11} & F^T_{12} & F^T_{13} \\ F^T_{21} & F^T_{22} & T^T_{23} \\ F^T_{31} & F^T_{32} & T^T_{33} \end{bmatrix}_{3\times3} \tag{6.7}$$

O resultado da multiplicação gera uma nova matriz O^1 com dimensão 3×3, cujo os elementos são representados pelas Equações (6.8) a (6.16).

$$o^1_{11} = p^+_{11} \cdot F^T_{11} + p^+_{12} \cdot F^T_{21} + p^+_{13} \cdot F^T_{31} \tag{6.8}$$

$$o^1_{12} = p^+_{11} \cdot F^T_{12} + p^+_{12} \cdot F^T_{22} + p^+_{13} \cdot F^T_{32} \tag{6.9}$$

$$o^1_{13} = p^+_{11} \cdot F^T_{13} + p^+_{12} \cdot F^T_{23} + p^+_{13} \cdot F^T_{33} \tag{6.10}$$

$$o^1_{21} = p^+_{21} \cdot F^T_{11} + p^+_{22} \cdot F^T_{21} + p^+_{23} \cdot F^T_{31} \tag{6.11}$$

$$o^1_{22} = p^+_{21} \cdot F^T_{12} + p^+_{22} \cdot F^T_{22} + p^+_{23} \cdot F^T_{32} \tag{6.12}$$

$$o^1_{23} = p^+_{21} \cdot F^T_{13} + p^+_{22} \cdot F^T_{23} + p^+_{23} \cdot F^T_{33} \tag{6.13}$$

$$o^1_{31} = p^+_{31} \cdot F^T_{11} + p^+_{32} \cdot F^T_{21} + p^+_{33} \cdot F^T_{31} \tag{6.14}$$

$$o^1_{32} = p^+_{31} \cdot F^T_{12} + p^+_{32} \cdot F^T_{22} + p^+_{33} \cdot F^T_{32} \tag{6.15}$$

$$o^1_{33} = p^+_{31} \cdot F^T_{13} + p^+_{32} \cdot F^T_{23} + p^+_{33} \cdot F^T_{33} \tag{6.16}$$

Por exemplo, para implementar a Equação (6.8), uma FSM foi projetada, realizando três multiplicações e duas adições ambas em ponto flutuante. Nesse caso, a estrutura FSM é composta por cinco estados: *waiting*, *mult1*, *mult2*, *multadd* e *add*. Como mostrado na Figura (6.2), a FSM está descrita na forma de uma FSM de *Mealy*, onde a saída para cada transição é omitida para simplificar a sua representação.

O primeiro estado da FSM é denominado de *waiting*, o qual espera um sinal de *start* do processador Nios II para inicializar a operação ou o fim de um passo anterior. O estado *mult1* realiza a primeira operação de multiplicação da esquerda para a direita na Equação (6.8). O estado *mult2* realiza a segunda operação de multiplicação. O estado *multadd* computa a terceira multiplicação e em paralelo faz a adição entre o resultado da primeira e segunda multiplicação. Por fim, o estado *add* realiza a operação de adição entre os resultados da primeira adição e o da terceira multiplicação. De modo similar e concorrentemente, as Equações (6.9) a (6.16) são computadas usando a mesma FSM.

Pode-se observar que os passos dois e três da etapa de *predição* (ver Equações (6.2) e (6.3)) são realizadas pelos mesmos passos da FSM e com a mesma operações em ponto flutuante de duas adições e três multiplicações. Entretanto, os passos quatro e cinco (ver Equações (6.4) e (6.5)) precisam de menos operações aritméticas, uma adição e duas multiplicações. Finalmente,

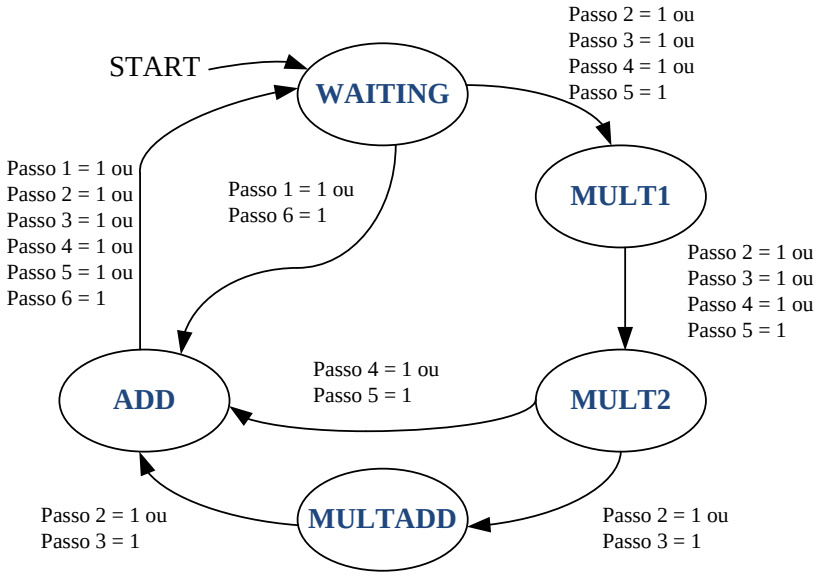


Figura 6.2. FSM usada na arquitetura para computar as equações da etapa de *predição*

os passos um e seis (ver Equações (6.1) e (6.6)) somente usam uma operação de adição em suas execuções.

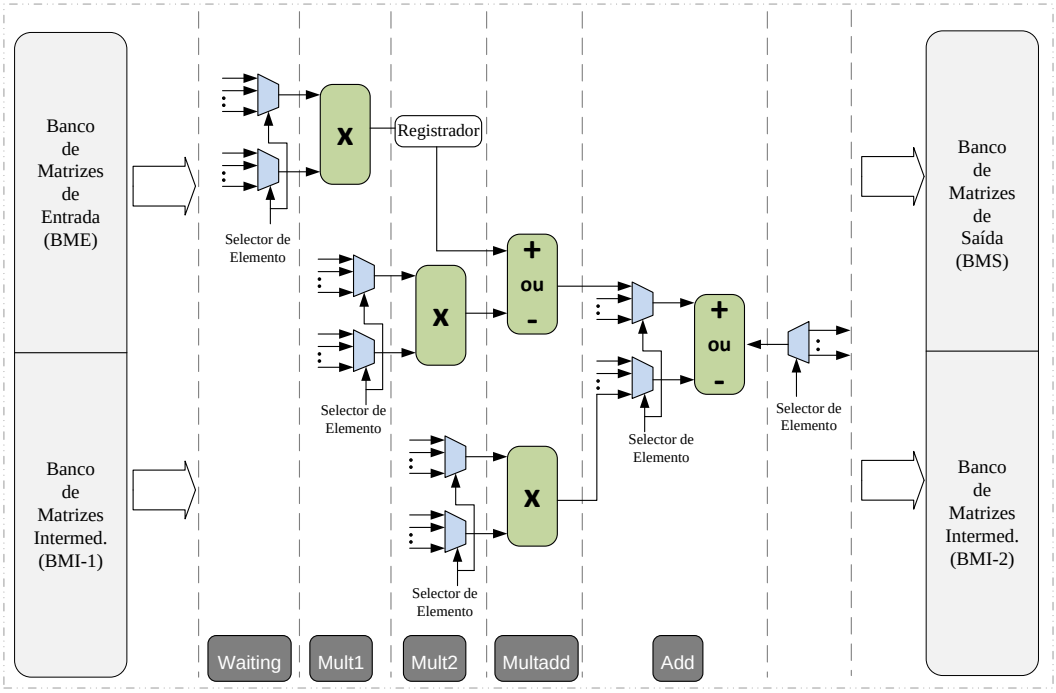


Figura 6.3. Escalonamento que gera o caminho de dados para computar a etapa de *predição*

A Figura (6.3) mostra o escalonamento (em inglês *scheduling*) das operações na arquitetura, incluindo um módulo UPF multiplicador e um somador/subtrator. Pode-se observar para a Equação (6.2) que todas as operações em ponto flutuante de um único elemento da matriz são realizados de forma sequencial (com cinco estados), mas os nove elementos da matriz (nas Equações (6.8) a (6.16)) são calculados ao mesmo tempo (em paralelo), replicando o mesmo caminho de dados (em inglês *datapath*) da arquitetura (ver Figura (6.4)). É importante deixar claro que os módulos UPF são reutilizados para cada estado, como acontece para os estados *mult1*, *mult2* e *multadd* que reutilizam a UPF de multiplicação.

A Figura (6.3) também mostra três bancos de registros da matrizes usadas na arquitetura implementada (BME, BMI e BMS). Esses registros podem ser acessados a qualquer momento, para leitura e escrita de operações, a partir do processador Nios II através do barramento Avalon (ver Figura 6.1). Nesse caso, funções especiais no Nios II estão disponíveis para executar esses processos de leitura e escrita nas matrizes. Depois de preencher o Banco das Matrizes de Entrada (BME), um comando de inicio (*start*) é enviado para a arquitetura, a fim de iniciar o processo. Sempre que o processo for concluído, os valores a partir do Banco das Matrizes de Saída (BMS) podem serem lidos. A arquitetura também inclui registros intermediários ou temporais, que são representados pelo Banco das Matrizes Intermediárias (BMI). Para a etapa de *predição* o BME é composto por $X_{k-1}^{+}, M_{k-1}, P_{k-1}^{+}, F, Q$ e W; o BMS é composto por X_k^{-} e P_k^{-} e o BMI por O^1, O^2, O^3 e O^4.

Um importante detalhe no projeto refere-se ao caminho de dados. A Figura (6.4) exibe 9 blocos de caminho de dados, cada um deles representando a computação de um único elemento de matriz. O número de caminho de dados é devido ao fato de a dimensão da maior matriz resultante durante o processo no passo dois ser 3×3. De forma análoga esta arquitetura apresentada é utilizada para os outros passos (Equações (6.1) a (6.6)), considerando a mesma FSM da Figura (6.2) com um caminho de dados da forma da Figura (6.3), variando somente no numero de estados que requer o passo.

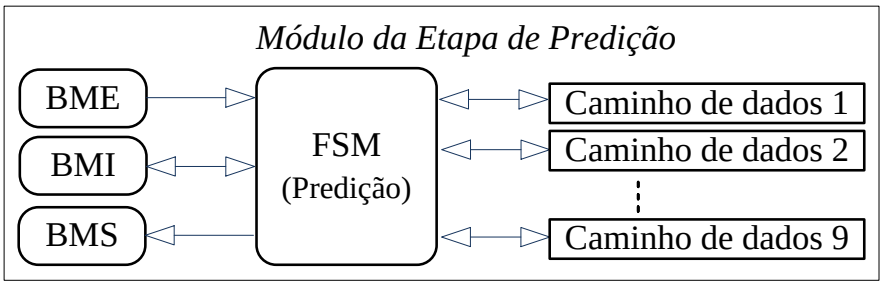


Figura 6.4. Arquitetura da etapa de *predição*

6.1.2 A Arquitetura da Etapa de *Estimação*

Seguidamente, será apresentada uma descrição da arquitetura em *hardware* para a etapa de *estimação* desenvolvida por [5], devido a que neste trabalho de mestrado faremos uso dele. Este *módulo de hardware* será conectado com o módulo da etapa de *predição* que foi explicado na Subseção 6.1.1, com o fim de ter uma implementação completa do algoritmo EKF em *hardware*.

Para a sua implementação, a etapa de *estimação* (vide Equações (4.9), (4.10) e (4.11)) foi decomposta em onze passos, como é visto nas Equações de (6.17) a (6.27).

$$Passo\ 1 : O^1_{3\times 2} = P^-_k \cdot H^T \tag{6.17}$$

$$Passo\ 2 : O^2_{2\times 2} = H \cdot O^1_{3\times 2} \tag{6.18}$$

$$Passo\ 3 : O^3_{2\times 2} = O^2_{2\times 2} + R \tag{6.19}$$

$$Passo\ 4 : O^4_{2\times 2} = inv(O^3_{2\times 2}) \tag{6.20}$$

$$Passo\ 5 : G_k = O^1_{3\times 2} \cdot O^4_{2\times 2} \tag{6.21}$$

$$Passo\ 6 : O^5_{2\times 1} = Z_k - h(X^-_k) \tag{6.22}$$

$$Passo\ 7 : O^6_{3\times 1} = G_k \cdot O^5_{2\times 1} \tag{6.23}$$

$$Passo\ 8 : X^+_k = X^-_k + O^6_{3\times 1} \tag{6.24}$$

$$Passo\ 9 : O^7_{3\times 3} = G_k \cdot H \tag{6.25}$$

$$Passo\ 10 : O^8_{3\times 3} = O^7_{3\times 3} \cdot P^-_k \tag{6.26}$$

$$Passo\ 11 : P^+_k = P^-_k - O^8_{3\times 3} \tag{6.27}$$

Como é mostrado na Figura (6.5), a FSM para esta etapa está descrita na forma de máquina de *Mealy*, onde a saída para cada transição é omitida para simplificar a sua representação.

Pode-se observar que todos os passos exceto o passo quatro precisam de operações aritméticas em ponto flutuante, tais como adições e multiplicações (vide Equações (6.17) a (6.19) e (6.21) a (6.27)), por essa razão a FSM é composta basicamente pelos seguintes estados: *waiting*, *mult1*, *mult2*, *multadd* e *add*. Entretanto, a quarta etapa (ver Equação (6.20) representa uma operação de inversão de matriz, que é realizada como $adjunta(O^3)/determinante(O^3)$ (na qual usa-se uma operação de divisão). Portanto, um estado de divisão (*div1*) foi adicionado à FSM (ver Figura 6.5) para realizar a divisão entre a *adjunta*() e o *determinante*(). É importante lembrar que o determinante é calculado usando as operações de multiplicação e subtração que são realizadas no estado *multadd* e *add* (ver as Figuras (6.5) e (6.6)).

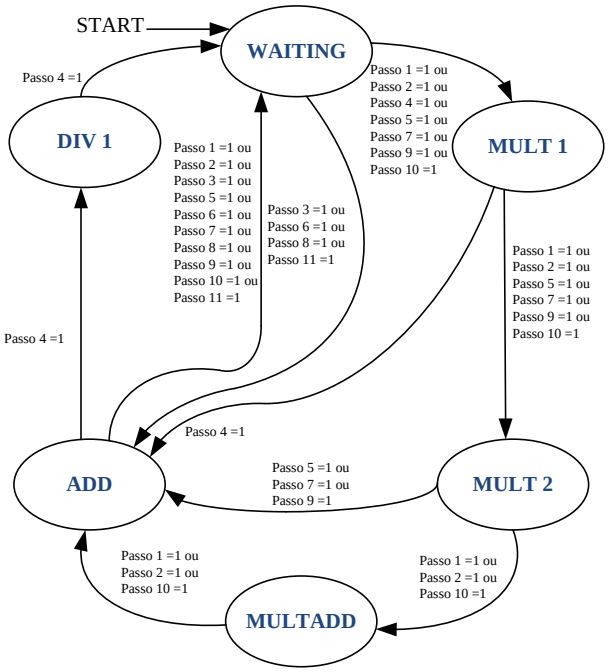


Figura 6.5. FSM usada na arquitetura para realizar o algoritmo de *estimação* do EKF, adaptada de [5]

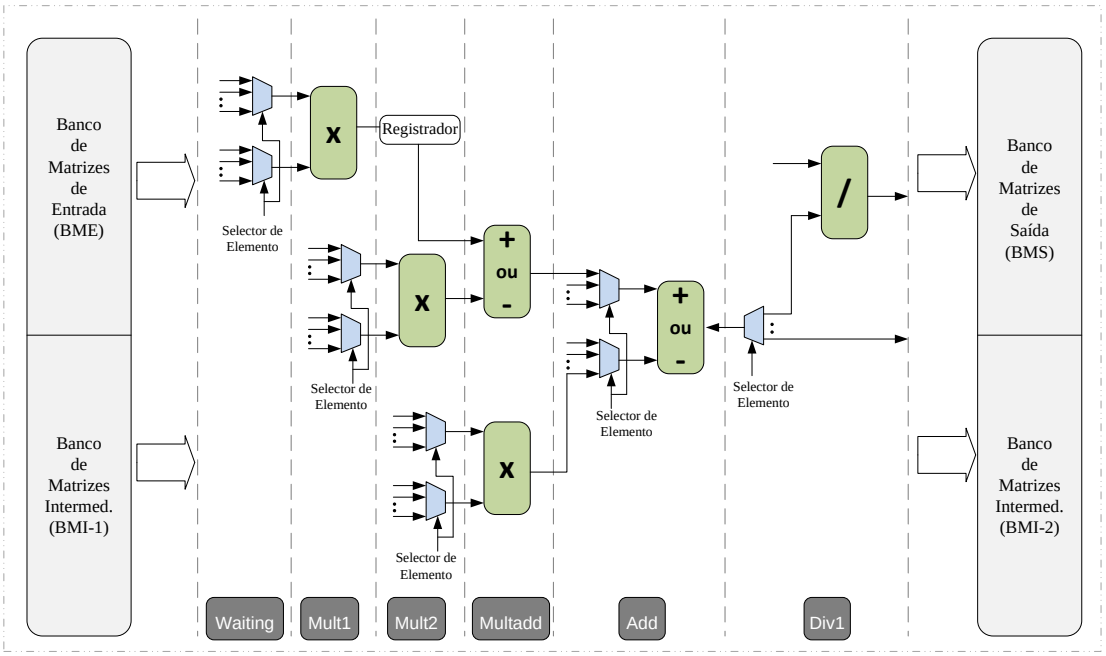


Figura 6.6. Escalonamento que gera o caminho de dados para computar a etapa de *estimação*, adaptado de [5]

A Figura (6.6) mostra o caminho de dados para computar a etapa de *estimação*. Nesse caso, o BME é composto por P_k^-, H, R, Z_k, $h()$ e X_k^-; o BMS é composto por G_k, X_k^+ e P_k^+ e o BMI

é composto por $O^1, O^2, O^3, O^4, O^5, O^6, O^7$ e O^8. Como aconteceu também para arquitetura da etapa de *predição*, as UPFs serão reutilizadas para cada estado.

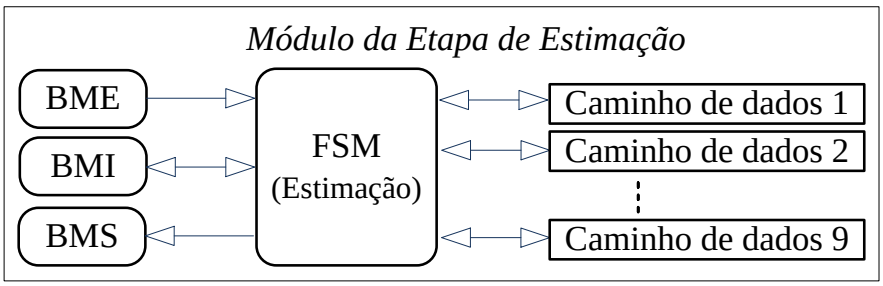


Figura 6.7. Arquitetura da etapa de *estimação*, adaptada de [5]

A Figura (6.7) exibe a arquitetura da etapa de *estimação*. Aqui, também há 9 blocos de caminhos de dados, onde cada um deles representa o caminho de dados para computar um único elemento de uma matriz. Isso se deve ao fato que a maior dimensão matricial usada na computação da etapa de *estimação* do EKF é 3×3. Vale lembrar que os caminhos de dados 5 a 9 não contem blocos de divisão porque o número de elementos na inversão matricial do processo é quatro (devido a que a dimensão da inversão matricial é 2×2), logo, somente os caminhos de dados 1 a 4 possuem blocos de divisão.

Tabela 6.1. Recursos de blocos em ponto flutuante para a primeira abordagem

Arquitetura	Somador/Subtrator	Multiplicador	Divisor
Módulo da etapa de Predição	9	9	-
Módulo da etapa de Estimação	9	9	4
Total	18	18	4

Por fim, pode-se observar que a Figura (6.8) mostra toda a arquitetura geral do projeto em FPGA considerando a arquitetura do EKF e outros componentes usados, tais como o processador Nios II e a UPF CORDIC. A Tabela (6.1) apresenta o número de blocos em ponto flutuante requerido para a implementação de cada arquitetura.

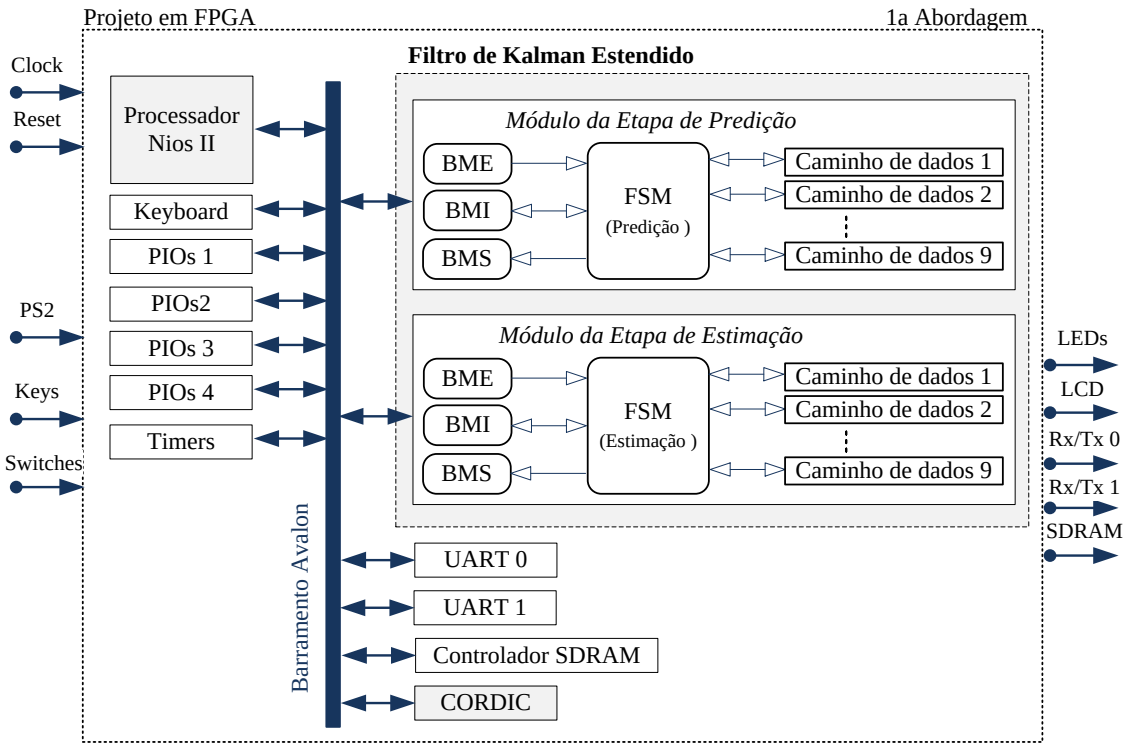


Figura 6.8. Projeto de FPGA com a arquitetura do EKF usando módulos individuais para cada etapa

6.2 Segunda abordagem para a solução do problema de localização: Implementação em FPGA usando um Módulo de *Hardware* Unificado

A segunda abordagem para a solução do problema de localização consistiu no desenvolvimento de um ***M**ódulo de **H**ardware **U**nificado* (*MHU*), devido a que este tipo de abordagem reduz o numero de operações de leitura/escrita que as arquiteturas em *hardware* realizam via o barramento de comunicação (neste caso Avalon), conseguindo por tanto um melhor desempenho do sistema. Este *MHU* foi projetado se considerando a integração de todas as equações do algoritmo EKF em somente uma arquitetura de *hardware*.

Neste caso, a abordagem é mostrada na Figura (6.9), em que a comunicação entre o processador Nios II e arquitetura em *hardware* é simplificada. Essa abordagem também reduz os recursos de *hardware* requeridos, em termos de blocos em ponto flutuante como multiplicadores, divisores e somadores/subtratores. Isso acontece devido a que é possível compartilhar o mesmo caminho de dados para a implementação em *hardware* das cinco equações do algoritmo EKF (vide Equações (4.7) e (4.11)), o que resulta na redução do número de operações em ponto

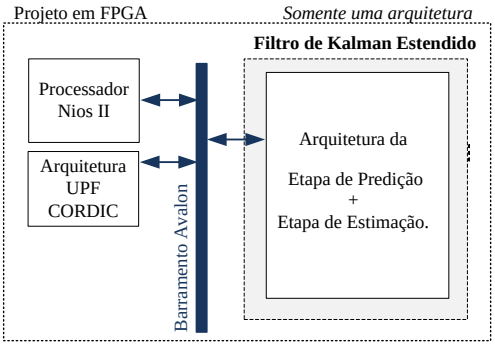


Figura 6.9. Segunda implementação em FPGA com somente uma arquitetura

flutuante e, consequentemente, na área do FPGA. Vale ressaltar que, assim como na primeira abordagem, o vetor M, as matrizes Jacobianas e função não linear $h()$ não estão incluídas no *hardware*; seus elementos são previamente calculados em *software* (usando o Nios II e a arquitetura CORDIC) e endereçado à arquitetura em *hardware*. O cálculo do *seno* e *cosseno*, neste caso, é realizado pela arquitetura UPF CORDIC para acelerar o computo de M, F, W, H e $h()$.

A implementação do algoritmo EKF foi dividida em dezessete passos, como são mostradas nas Equações (6.28) a (6.44).

$$Passo\ 1 : X_k^- = X_{k-1}^+ + M_{k-1} \tag{6.28}$$

$$Passo\ 2 : O_{3\times3}^1 = P_{k-1}^+ \cdot F^T \tag{6.29}$$

$$Passo\ 3 : O_{2\times3}^2 = F \cdot O_{3\times3}^1 \tag{6.30}$$

$$Passo\ 4 : O_{2\times3}^3 = Q \cdot W^T \tag{6.31}$$

$$Passo\ 5 : O_{3\times3}^4 = W \cdot O_{2\times3}^3 \tag{6.32}$$

$$Passo\ 6 : P_k^- = O_{3\times3}^2 + O_{3\times3}^4 \tag{6.33}$$

$$Passo\ 7 : O_{3\times2}^5 = P_k^- \cdot H^T \tag{6.34}$$

$$Passo\ 8 : O_{2\times2}^6 = H \cdot O_{3\times2}^5 \tag{6.35}$$

$$Passo\ 9 : O_{2\times2}^7 = O_{2\times2}^6 + R \tag{6.36}$$

$$Passo\ 10 : O_{2\times2}^8 = inv(O_{2\times2}^7) \tag{6.37}$$

$$Passo\ 11 : G_k = O_{3\times2}^5 \cdot O_{2\times2}^8 \tag{6.38}$$

$$Passo\ 12 : O_{2\times1}^9 = Z_k - h(X_k^-) \tag{6.39}$$

$$Passo\ 13 : O^{10}_{3\times1} = G_k \cdot O^{9}_{2\times1} \tag{6.40}$$

$$Passo\ 14 : X^{+}_{k} = X^{-}_{k} + O^{10}_{3\times1} \tag{6.41}$$

$$Passo\ 15 : O^{11}_{3\times3} = G_k \cdot H \tag{6.42}$$

$$Passo\ 16 : O^{12}_{3\times3} = O^{11}_{3\times3} \cdot P^{-}_{k} \tag{6.43}$$

$$Passo\ 17 : P^{+}_{k} = P^{-}_{k} - O^{12}_{3\times3} \tag{6.44}$$

onde se considera que apos a execução do primeiro passo (Equação 6.28) o resultado X^{-}_{k} é levado ao Nios II, devido a que a função não linear $h(X^{-}_{k})$ necessita deste dado para ser calculada e depois ser endereçada na arquitetura em *hardware*, já que como pode ser visto na Equação 6.39 esta função não linear é usada no passo doze.

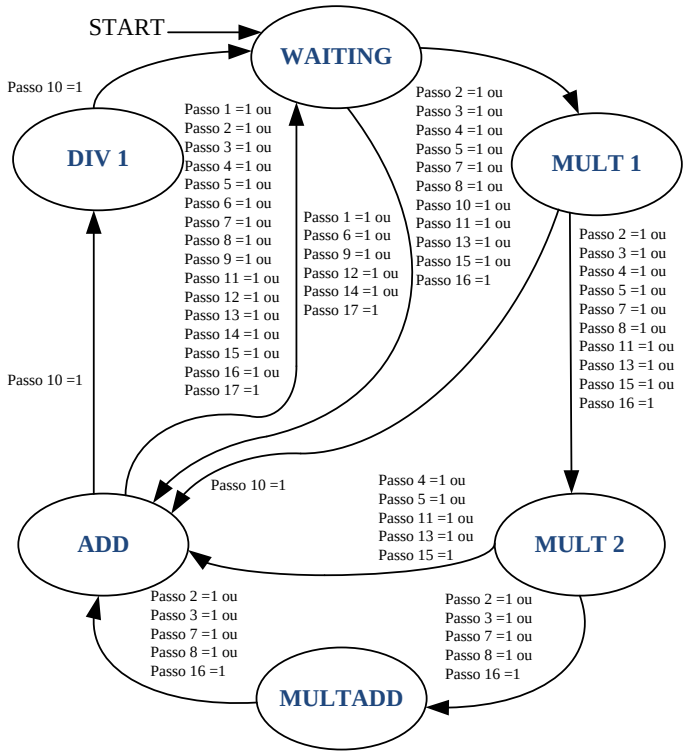


Figura 6.10. FSM usada na arquitetura para realizar o algoritmo do EKF

A estrutura da FSM usada na arquitetura do algoritmo EKF é exibida na Figura (6.10) e pode-se observar que ela é composta de seis estados, nomeados de *waiting*, *mult1*, *mult2*, *multadd*, *add* e *div1*. Já a Figura (6.11) exibe o caminho de dados para computar a FSM. Na arquitetura EKF, o BME é composto por $X^{+}_{k-1}, M_{k-1}, P^{+}_{k-1}, F, Q, W, H, R, Z$ e $h()$; o BMS é composto por $X^{-}_{k}, P^{-}_{k}, G_k, G_k, X^{+}_{k}$ e P^{+}_{k}; e o BMI é composto por $O^1, O^2, O^3, O^4, O^5, O^6, O^7, O^8, O^9, O^{10}, O^{11}$ e O^{12}.

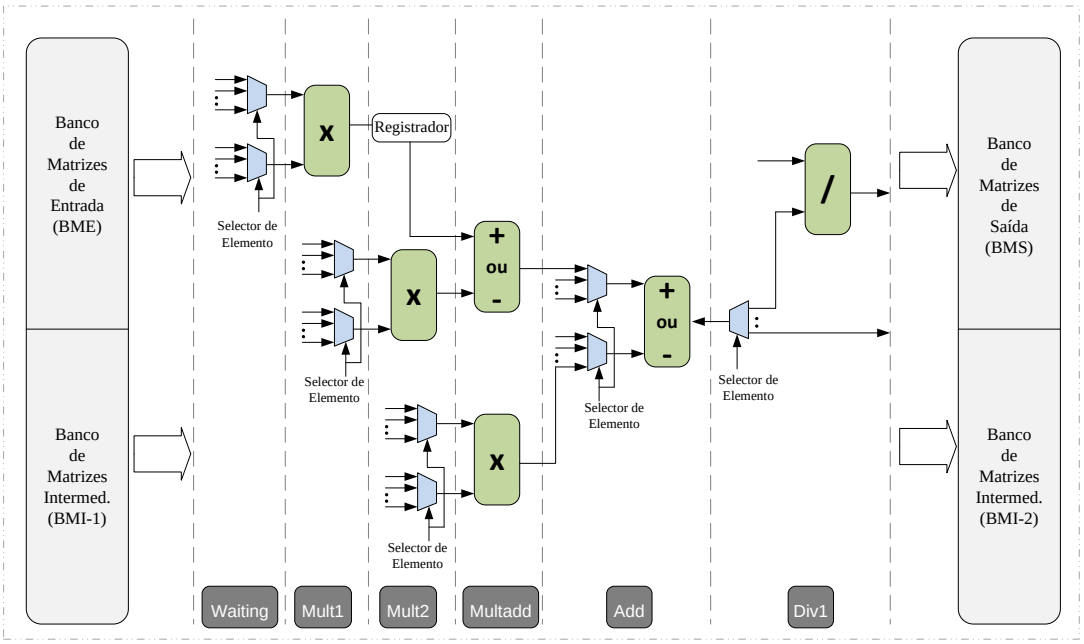


Figura 6.11. Escalonamento que gera o caminho de dados para computar o algoritmo do EKF usando uma abordagem MHU

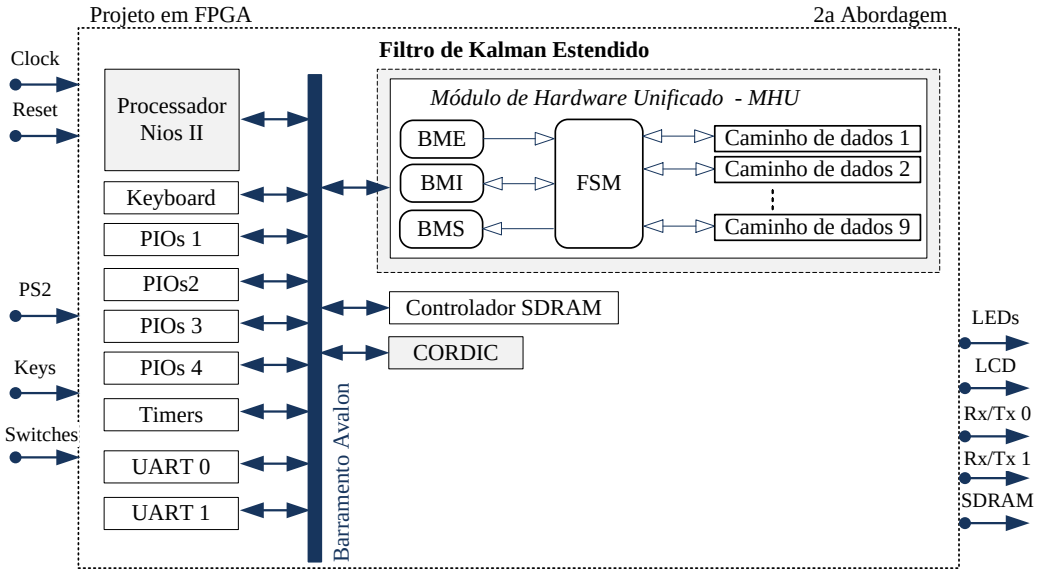


Figura 6.12. Projeto de FPGA com a arquitetura do algoritmo do EKF usando uma abordagem MHU

Pode-se observar que a Figura (6.12) mostra todo o projeto geral em FPGA incluindo a arquitetura do algoritmo EKF. Nesse caso, há 9 blocos de caminhos de dados, devido a que a maior dimensão matricial usada na computação do EKF é 3×3, onde cada um dos caminhos são necessários para calcular cada elemento das matrizes. Também neste caso, os caminhos de

dados 5 a 9 não contem blocos de divisão porque o número de elementos na inversão matricial do processo é quatro (já que a dimensão da inversão matricial é de 2×2). Desta maneira os únicos caminhos de dados que possuem blocos de divisão são os quatro primeiros (*datapaths* 1 a 4).

Finalmente, a Tabela (6.2) apresenta o número de blocos em ponto flutuante requerido pela implementação do *Módulo de Hardware Unificado*.

Tabela 6.2. Recursos de blocos em ponto flutuante para a segunda abordagem

Arquitetura	Somador/Subtrator	Multiplicador	Divisor
MHU para EKF	9	9	4

6.3 Conclusões do Capítulo

Neste capítulo se apresentaram duas abordagens de implementação das equações do Filtro de Kalman Estendido na versão sequencial. Na primeira delas, considerou-se uma arquitetura para o EKF usando módulos de *Hardware* individuais para a etapa de *predição* e *estimação*, enquanto que na segunda abordagem foi apresentada uma única arquitetura que computa todo o algoritmo, nomeada de MHU.

Para a primeira abordagem (módulos individuais) pode-se destacar a sua modularidade, ou seja, pode-se utilizar no projeto uma dessas arquiteturas. Isto é útil se visamos FPGAs de baixa densidade (elementos lógicos), onde uma parte do algoritmo pode ser executada em *software* e a outra em *hardware*. Para a segunda abordagem (MHU) podemos destacar a diminuição no consumo de *hardware* e desempenho, bem como no uso de unidades em ponto flutuante, conforme visto na Figura (6.13).

Desta maneira, comparando com [5], onde se desenvolveu uma arquitetura especificamente para a etapa de *estimação*, o presente trabalho tem a avantajem de conseguir a completa implementação do EKF, considerando ambas etapas (*predição* e *estimação*) e usando qualquer uma das duas abordagens propostas.

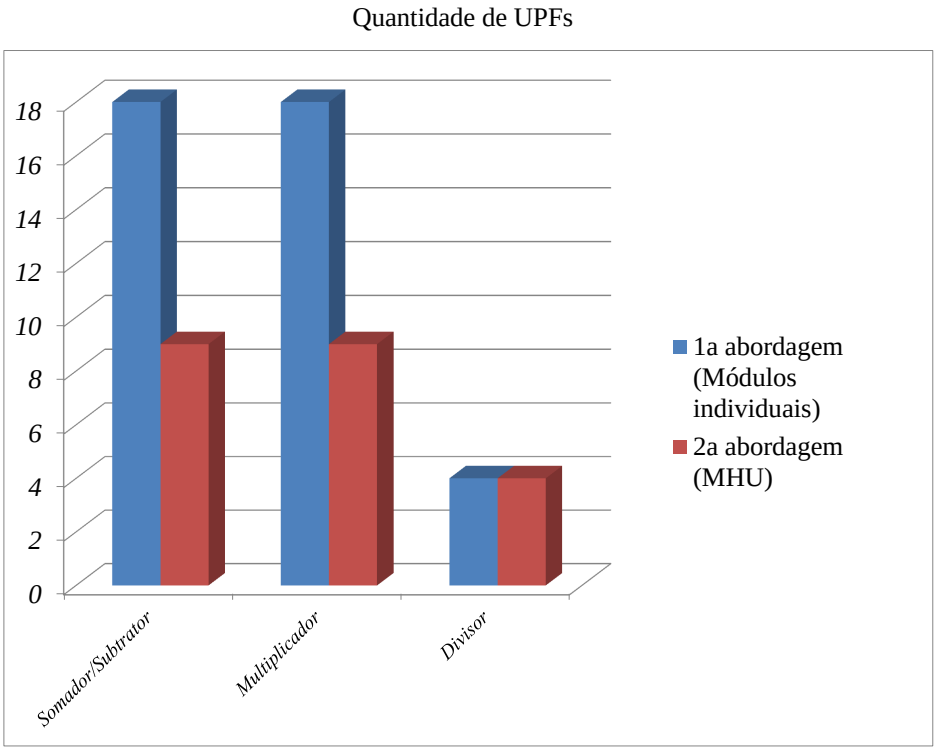


Figura 6.13. Comparativo de consumo de UPFs entre a 1^a e a 2^a abordagem

7 Resultados da Implementação

Neste capítulo são apresentados os resultados experimentais obtidos através da primeira e segunda abordagens para a solução em *hardware* do problema de localização em robótica móvel (vide Capítulo 6), levando em consideração recursos de *hardware* e desempenho. As arquiteturas propostas foram escritas na linguagem de programação VHDL (do inglês ***V****HSIC* ***H****ardware* ***D****escription* ***L****anguage*) e mapeadas usando o *software* Quartus II [51]. As mesmas foram validadas no *kit* de desenvolvimento DE2-115, que possui uma Cyclone IV EP4CE115F29C7N [22] (FPGA de baixo custo).

7.1 Resultados de síntese

Os resultados de recursos de *hardware* em um dispositivo FPGA EP4CE115F29C7 (família Cyclone IV E) para as arquiteturas propostas usando ambas abordagens são descritas na Tabela 7.1 bem como o consumo de potência, o qual foi estimado a partir da ferramenta PowerPlay Power Analyzer do *software* Quartus II.

Tabela 7.1. Recursos de *hardware* para as arquiteturas EKF propostas

Arquitetura	**LEs**	**DSPs**	**Fmáx (MHz)**	**Potência (mW)**
Abordagem 1:				
Módulo de *predição* EKF	15843 (14%)	63 (12%)	49,86	183,93
Módulo de *estimação* EKF(*)	22089 (19%)	175 (33%)	49,08	201,13
Usando módulos individuais para o EKF				
(Mod. *Predição* + Mod. *Estimação*)	37932 (33%)	238 (45%)	48,00	385,06
Abordagem 2:				
Usando o enfoque MHU para o EKF	27889 (24%)	175 (33%)	49,91	235,74

* desenvolvido em [5]

Como pode ser visto, na primeira abordagem, a arquitetura que obteve melhor desempenho foi a correspondente ao módulo de *predição* EKF, já que ela só contem operações matriciais do tipo *transposta*, *soma* e *multiplicação*. Como consequência, teve-se um menor consumo de elementos lógicos (15843 LEs) e maior frequência de operação (49,86 MHz). Nesta mesma abordagem, a arquitetura do módulo de *estimação* obteve um menor desempenho. Isso é devido

a que a arquitetura além de ter operações matriciais como transposta, soma e multiplicação, conta também com operações de inversão matricial, a qual precisa de 4 unidades de divisão para a sua execução, aumentando assim o número de elementos lógicos (LEs) e o caminho entre a entrada e saída do sinal, fazendo com que a frequência máxima que a arquitetura possa rodar diminua (49,08 MHz).

Já para a segunda abordagem (enfoque MHU), o número de elementos lógicos é maior (27889), porém se for feita a somatória do número de LEs das arquiteturas da 1^a implementação (37932), tem-se uma redução de aproximadamente 26%. Isso é válido também para o consumo de DSPs (do inglês ***D**igital **S**ignal **P**rocessor*), a segunda implementação consome 175 DSPs enquanto que a soma das duas arquiteturas da primeira abordagem dá 238 DSPs, ou seja, uma redução de 26%. Neste caso, a frequência é levemente maior (49,91 MHz) que da arquitetura do módulo de *predição*.

Com respeito ao consumo de potência, entre a primeira e a segunda abordagem, a última leva uma grande vantagem, pois a soma das potências das duas arquiteturas (módulos de *predição e estimação*) é quase 63% maior do que a arquitetura unificada (MHU).

Tabela 7.2. Recursos de *hardware* para as arquiteturas EKF propostas em outras famílias de FPGA Altera

Arquitetura	FPGA	LEs (*)	DSP (*)	Fmáx (MHz)
Módulo de Predição EKF	DE0_Nano	15953 (71%)	63 (48%)	72,42
	Arria II GX	11653 (6%)	36 (5%)	127,5
	Stratix IV GX	11679 (6%)	36 (3%)	145,39
Módulo de Estimação EKF [5]	DE0_Nano	25461 (114%)	132 (100%)	-
	Arria II GX	16505 (8%)	100 (14%)	62,64
	Stratix IV GX	16685 (9%)	100 (11%)	64,03
EKF usando enfoque MHU	DE0_Nano	31832 (143%)	132 (100%)	-
	Arria II GX	19819 (10%)	100 (14%)	96,58
	Stratix IV GX	19874 (14%)	100 (11%)	118,61

Por último, a Tabela 7.2 apresenta uma série de resultados em relação à compilação das arquiteturas apresentadas para diferentes tipos de FPGAs da Altera. Nesta tabela é possível observar os resultados para consumo de elementos lógicos (LEs), DSPs e frequência máxima de operação da arquitetura. As compilações foram direcionadas para os FPGAs Cyclone IV EP4CE22F17C6N (FPGA de baixo custo, disponível no *kit* de desenvolvimento DE0_Nano), Arria II GX EP2AGX260FF35I3 (FPGA de médio porte) e Stratix IV GX EP4SGX230KF40C2 (FPGA de alto desempenho).

7.2 Simulação Comportamental

As arquiteturas propostas neste trabalho foram simuladas usando a ferramenta computacional ModelSim-Altera Start Edition [52]. A partir do uso de módulos *testbenchs* (escritos em VHDL) nas simulações, conseguiu-se realizar a verificação do correto funcionamento das arquiteturas propostas, tanto para a primeira quanto para a segunda abordagem.

A Tabela 7.3 mostra o desempenho em termos de ciclos de relógio para cada arquitetura. A simulação comportamental das mesmas é apresentada na Figura 7.1, sendo que os valores que aparecem estão em formato hexadecimal padrão IEEE-754. Neste caso, é possível observar que a arquitetura da etapa de *Predição* gasta 1000 nanosegundos (ns) para realizar a sua execução da mesma (20 nanosegundos $\times$ 50 ciclos de relógio). Por outro lado, a arquitetura da etapa de *Estimação* gasta 2080ns e a arquitetura do algoritmo EKF com enfoque MHU 3080ns. É importante mencionar que, enquanto a tarefa está sendo processada, o sinal *ready* está em nível alto. Quando o algoritmo é finalizado, o mesmo sinal vai para o nível baixo.

Tabela 7.3. Ciclos de relógio das arquiteturas propostas

Abordagem	**Arquitetura**	**Ciclo de Relógio**
1^a	Módulo de *Predição*	50
1^a	Módulo de *Estimação**	104
2^a	MHU para EKF	154

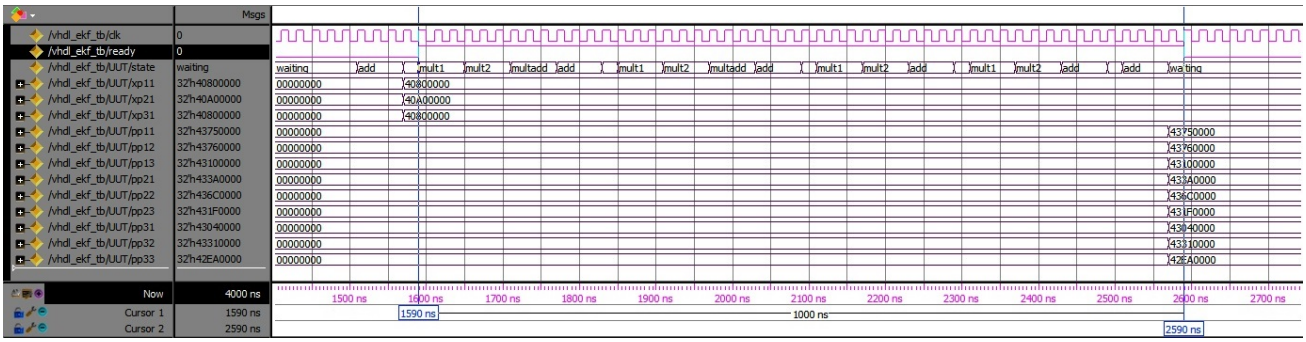
* desenvolvido em [5]

7.3 Tempo de execução das arquiteturas para o EKF

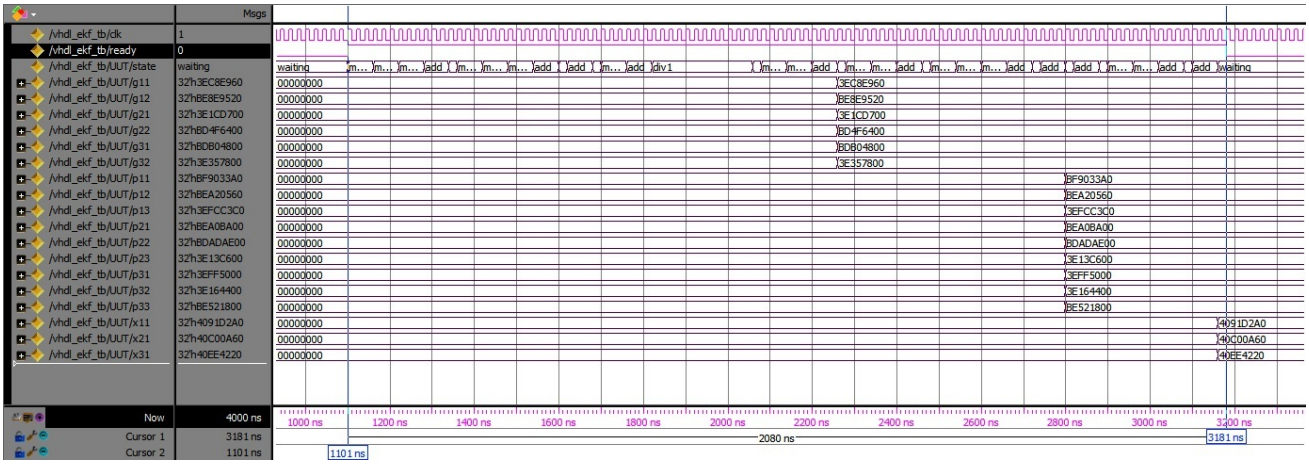
Com o objetivo de analisar o desempenho das arquiteturas propostas foi feita uma comparação do tempo de execução para diferentes implementações do EKF. Os resultados globais de tempo de execução são mostrados na Tabela 7.4.

Neste caso, algoritmo EKF foi implementado completamente nos processadores Nios II (50 MHz) e Intel® Core™ i5-2410M (2.90 GHz, 35W) para calcular o desempenho de implementações em *software*, estimando um tempo necessário de 1114 μs e 195 μs respectivamente.

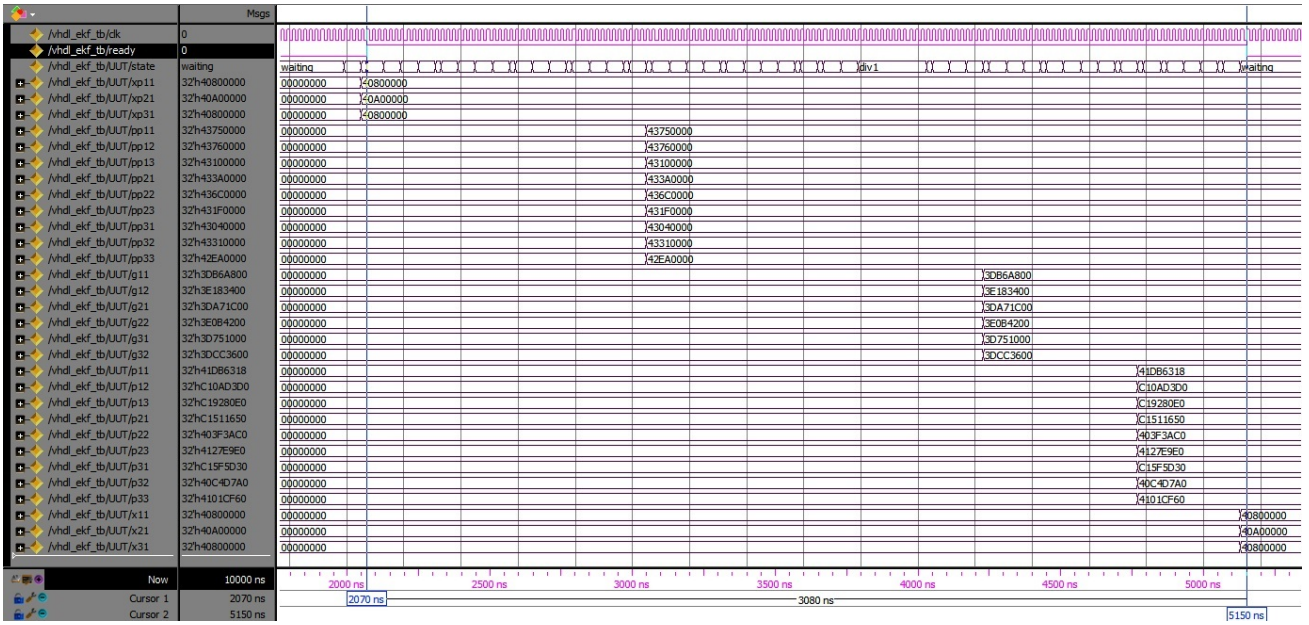
Adicionalmente, o mesmo algoritmo foi implementado em *hardware* considerando a primeira abordagem tendo um tempo de execução de 3,08 μs, resultado que foi obtido de somar os tempos de execução das etapas de *predição* 1 μs com *estimação* 2,08 μs, mencionados na Seção 7.2. Tanto a primeira quanto a segunda abordagem implementadas em *hardware* obtiveram o mesmo tempo de execução de 3,08 μs, por tanto um fator de aceleração de 361 e 63 em comparação com as



(a)



(b)



(c)

Figura 7.1. Resultado da simulação comportamental para a arquitetura (a) da etapa de *Predição* (b) da etapa de *Estimação* e (c) do Algoritmo EKF usando o enfoque MHU

implementações puras em *software* usando os procesadores Nios II e Intel ®, respectivamente. Em todos os casos, foi medido o calculo do tempo de execução de uma iteração do algoritmo EKF sem considerar os cálculos prévios para computar as matrizes Jacobianas.

Tabela 7.4. Comparação do tempo de execução para diferentes implementações do EKF

Implementação	**Tempo de Execução (μs)**
Puramente em *Software*:	
Processador Nios II (50 MHz)	1114
Processador Intel® (2.90 GHz)	195
Abordagem 1 (módulos individuais):	
Em *Hardware* (50 MHz)	3,08
Em *Hardware* + Atraso (50 MHz)	64
Abordagem 2 (enfoque MHU):	
Em *Hardware* (50 MHz)	3,08
Em *Hardware* + Atraso (50 MHz)	46

No entanto, tendo em conta o atraso de comunicação de dados entre Nios II e as arquiteturas de *hardware* (escrita e leitura via o barramento Avalon), estes fatores velocidade decaem em ambos casos (ver Implementação em *Hardware* mais Atraso na Tabela 7.4). Considerando esse atraso, pode-se observar na Tabela 7.4 que a segunda abordagem usando o enfoque MHU presenta um melhor desempenho em comparação às outras.

É importante lembrar que todas as implementações (exceto a correspondente em Intel®) rodaram com um relógio de 50MHz.

7.4 Validação das arquiteturas para o EKF

A validação experimental das arquiteturas propostas foram realizadas implementando o algoritmo EKF no *kit* de desenvolvimento Altera DE2-115, adaptando a mesma à plataforma móvel Pioneer 3-AT (P3-AT).

Para este caso, foi necessário a identificação dos parâmetros cinemáticos do modelo do robô descrito no Capítulo 3 (vide Equação 3.16) para ser usados no algoritmo EKF em tempo real. A partir de varias trajetórias de prova para o robô foi realizada a identificação dos parâmetros, onde as medidas dos *encoders* foram registradas a cada 1,428 segundos ($\Delta\ T$=1,428 s). Os parâmetros obtidos considerando o modelamento do robô móvel *Pioneer 3-AT* são mostrados na Tabela 7.5.

Em relação à implementação do algoritmo EKF para à localização do SSMR, considerou-se que a representação das matrizes usadas no EKF foram os seguintes:

Tabela 7.5. Parâmetros do modelo do SSMR *Pioneer 3-AT*

Parâmetros	Valor	Unidade
α	1,08	-
D	798	mm
x_{CIR}	-0,86	mm

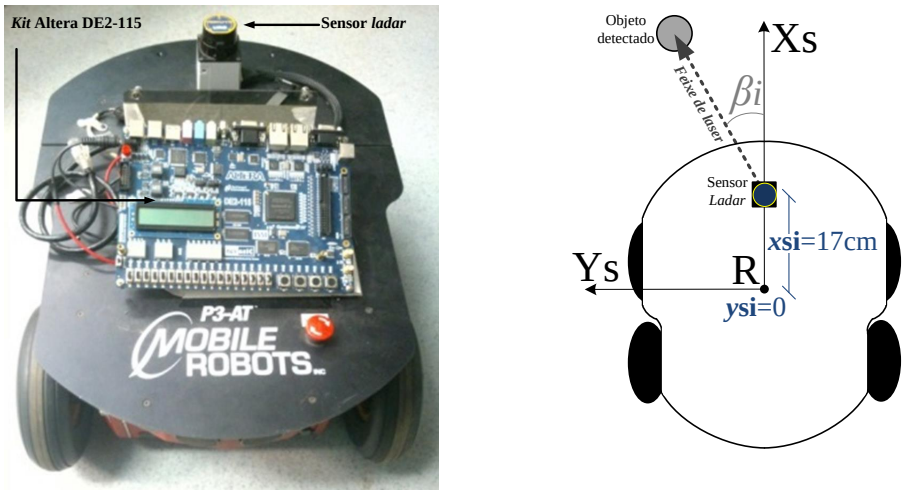
$$X_0^+ = [5mm\ ,\ 5mm\ ,\ 0^o]^T$$

$$P_0^+ = diag([1\ ,\ 1\ ,\ 1]),$$

$$Q = diag([(0,0352v_x(t))^2 + (7w(t))^2\ ,\ (0,0352v_x(t))^2 + (7w(t))^2]),$$

$$R = diag([20,25\ ,\ 20,25]),$$

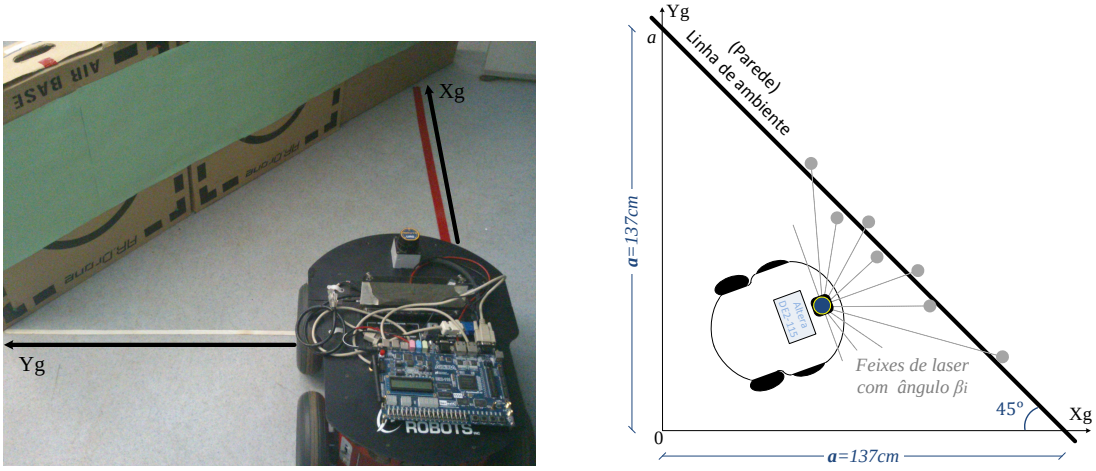
onde X_0^+ e P_0^+ são as matrizes iniciais de estado e covariância da pose do robô, respetivamente. Q e R são as matrizes de covariância de ruído associadas ao movimento e à medição, respectivamente. No caso da matriz Q, a presencia de seus elementos de entrada na diagonal da matriz é devida aos escorregamentos ou superfícies não planas. A derrapagem das rodas é uma função da velocidade ou aceleração das rodas. Por tanto, quando os ruídos foram somente devidos à existência de derrapagem (assumindo uma superfície plana), então os valores de entrada da diagonal da matriz de covariância são definidos como funções crescentes das velocidades v_x e w.



(a) (b)

Figura 7.2. (a) *Kit* Altera DE2-115, Pioneer 3-AT e sensor *ladar* usados nos testes. (b) Posicionamento do sensor *ladar* na plataforma P3-AT.

Em seguida um cenário foi preparado (no laboratório LEIA-UnB) para a implementação da tarefa de validação das arquiteturas propostas. Como foi mencionado no inicio desta seção, o



(a) (b)

Figura 7.3. (a) Cenário para a validação das arquiteturas EKF. (b) Dimensões do cenário.

algoritmo EKF rodou em um *kit* de desenvolvimento Altera DE2-115. Um robô, baseado na plataforma P3-AT, também foi usado, bem como os dados dos seus *encoders*. Um sensor *ladar* (LRF) também foi usado como medidor de distância, ajudando assim, a diminuir a matriz de covariâncias (vide Figura 7.2 (a)). Este sensor LRF foi colocado no robô com posicionamento $x_{si} = 17cm$ e $y_{si} = 0$ (vide Figura 7.2 (b)), com o fim que o sensor fique orientado na frente do robô para uma ter uma melhor leitura das medições. Os dados vindos dos sensores foram usados para estimar a pose do robô (X_k^+). Pode-se observar na Figura 7.3 (a) que o ambiente de teste, o qual foi projetado com $a = 137cm$ como é indicado na Figura 7.3 (b).

7.4.1 Implementação em *hardware* vs Implementação em *software*

No intuito de verificar o correto funcionamento das implementações em *hardware*, os resultados das implementações em *hardware* foram comparados passo a passo com os resultados da implementação puramente em *software*.

Durante o teste, viu-se que os valores resultantes do algoritmo EKF usando tanto a primeira quanto à segunda abordagem foram as mesmas, por essa razão nesta subseção, serão mencionados somente os resultados da última abordagem. Neste caso, o robô usou um feixe de laser com ângulo de orientação $\beta_i = 0^o$ (referir-se à Figura 7.2(b)). O teste consistiu em programar uma tarefa ao robô para realizar uma trajetória especifica, comparando passo a passo os resultados de estimação de localização em *hardware* e *software*.

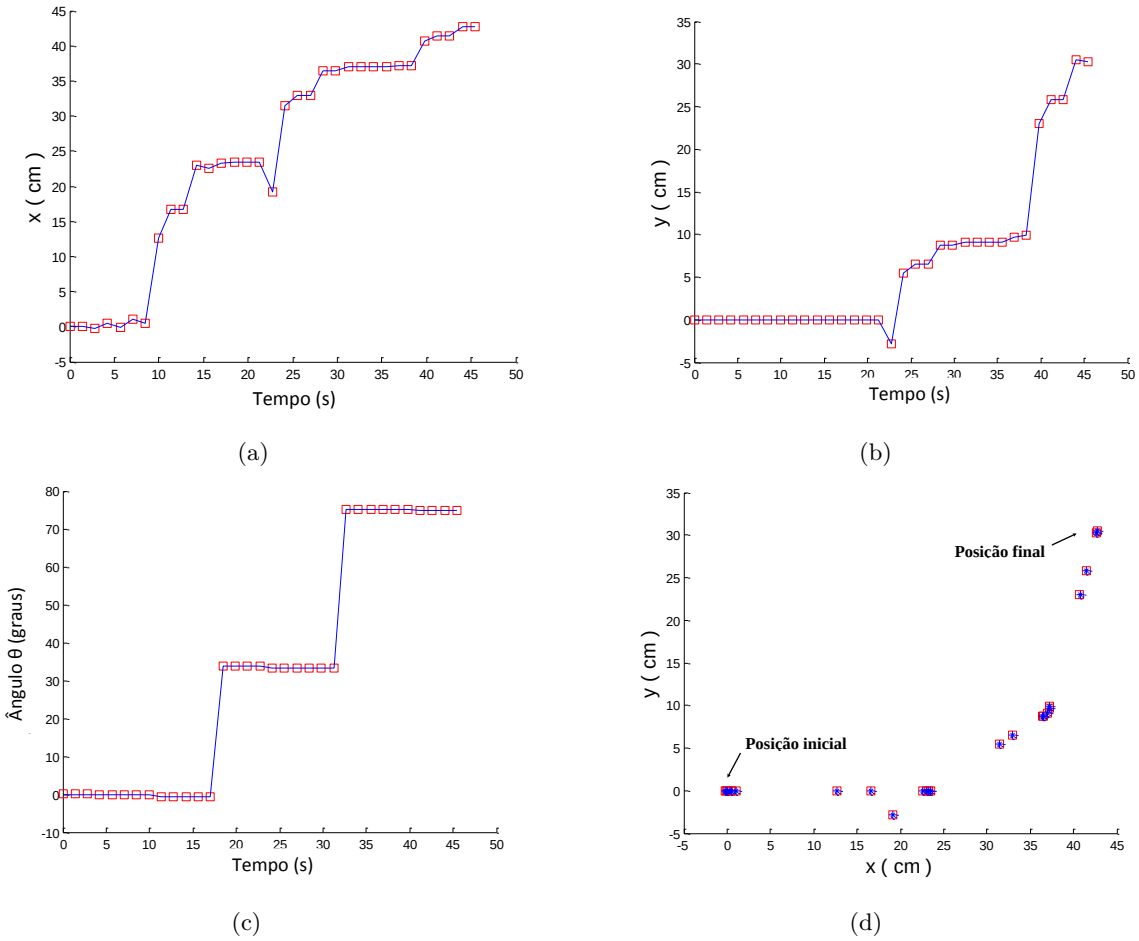


Figura 7.4. Resultados de estimação da posição do robô (a) x, (b) y e (c) θ. Os quadrados vermelhos e as linhas azuis representam a solução em *hardware* e *software* respectivamente, para a mesma aquisição de dados. (d) Estimativa da pose no plano $Xg - Yg$ onde as estrelas azuis representam a solução em *software* e os quadrados vermelhos correspondem à solução em *hardware*.

A implementação em *hardware* usou a arquitetura de MHU proposta no presente trabalho, a qual foi escrita em linguagem VHDL usando o *software* Quartus II 12.1; por outro lado a implementação em *software* foi desenvolvida totalmente no processador Nios II, sendo escrita em linguagem C. Para este caso, os resultados da estimação de cada variável de estado (x, y, θ) são mostrados nas Figuras 7.4 (a), (b), (c). Pode-se observar que tanto as estimativas para a implementação em *hardware* como em *software* são muito similares durante a sessão de teste. Por outro lado, a Figura 7.4 (d) fornece outro ponto de vista, representado os resultados de estimação da pose do robô no plano $Xg - Yg$, onde é possível também observar que ambos resultados, *hardware* e *software*, são muitos próximos durante toda a execução da tarefa programada.

Os dados mostrados na Figura 7.4, são apresentados nas Tabelas 7.6, 7.7 e 7.8; onde se pode corroborar que os resultados do vetor de estado (x, y, θ) da implementação proposta em *hardware* ficam muito próximos de solução totalmente em *software*, tendo uma diferença na ordem dos

milésimos.

Adicionalmente, foi implementado o algoritmo EKF em Matlab®, rodando em um processador Intel® Core™ i5. Neste caso o Matlab® foi considerado como estimador estadístico, para efeito de comparar o erro de cálculo de todos os elementos dos vetores de estado (X_k^- e X_k^+), matrizes de covariança (P_k^- e P_k^+) e matriz de ganho (G_k) do EKF usando a implementação proposta em *hardware*, como mostra Tabela 7.9. Os resultados demonstraram novamente o correto funcionamento da implementação em *hardware* proposta neste manuscrito.

7.4.2 Implementação em *hardware* vs Sistema de localização próprio do P3-AT

Logo de conferir o correto funcionamento das implementações em *hardware*, procedeu-se com a validação experimental da arquitetura proposta (segunda abordagem) para robótica móvel. Para isso, os resultados das implementações em *hardware* foram comparados com os resultados do sistema de localização próprio do P3-AT. A partir disso, foram executados três tipos de testes, cada um deles usando diferentes números de feixes do sensor *ladar*. No primeiro teste o sensor LRF utilizou 1 feixe de laser, no segundo teste 2 feixes de laser, e no ultimo teste foram utilizados 3 feixes de laser, isto a fim de observar o impacto do numero de feixes de laser no comportamento da estimação da implementação proposta em *hardware*.

7.4.2.1 Usando um (1) feixe de *ladar* no sistema de medição

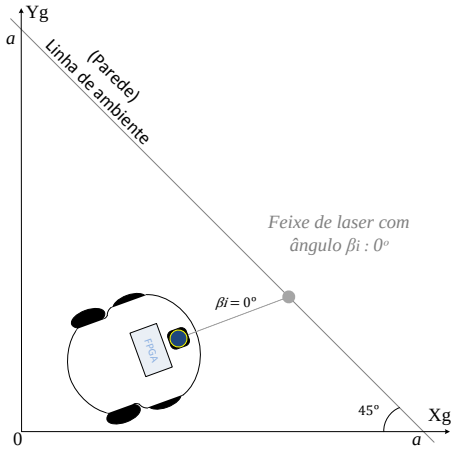


Figura 7.5. Test usando um (1) feixe de *ladar*.

Para este teste foi usado um feixe de laser no sistema de medição orientado com ângulo $\beta_i = 0^o$ para a frente do robô como é descrito na Figura 7.5). Os resultados da estimação de

Tabela 7.6. Comparação dos resultados de estimação da variável de estado x para diferentes implementações

Número de iteração	Em *hardware* - MHU (cm)	Em *software* - Nios II (cm)	Diferença (cm)
1	0	0	0,000000
2	0	0	0,000000
3	-0,19526	-0,19519	0,000071
4	0,440595	0,442885	0,002290
5	-0,05448	-0,05518	0,000700
6	1,008768	1,00464	0,004128
7	0,45307	0,449917	0,003153
8	12,63676	12,63361	0,003148
9	16,64656	16,64341	0,003147
10	16,64656	16,64341	0,003147
11	23,06897	23,06891	0,000057
12	22,56483	22,56696	0,002138
13	23,25517	23,25964	0,004467
14	23,48649	23,49096	0,004469
15	23,48649	23,49096	0,004469
16	23,48649	23,49096	0,004469
17	19,13244	19,13471	0,002275
18	31,43735	31,43978	0,002426
19	32,98327	32,98576	0,002483
20	32,98327	32,98576	0,002483
21	36,43449	36,43733	0,002842
22	36,51046	36,51318	0,002716
23	36,97678	36,97924	0,002460
24	36,97678	36,97924	0,002460
25	36,97678	36,97924	0,002460
26	36,97678	36,97924	0,002460
27	37,1932	37,19538	0,002182
28	37,24513	37,24755	0,002426
29	40,73634	40,73862	0,002281
30	41,47028	41,47253	0,002250
31	41,47028	41,47253	0,002250
32	42,71449	42,71682	0,002331
33	42,67043	42,67297	0,002537

Tabela 7.7. Comparação dos resultados de estimação da variável de estado y para diferentes implementações

Número de iteração	Em *hardware* - MHU (cm)	Em *software* - Nios II (cm)	Diferença (cm)
1	0,000000	0,000000	0,000000
2	0,000000	0,000000	0,000000
3	0,000003	0,000000	0,000003
4	0,000038	0,000000	0,000038
5	0,000037	0,000000	0,000037
6	-0,000030	0,000000	0,000030
7	-0,000008	0,000000	0,000008
8	0,000206	0,000000	0,000206
9	-0,015143	-0,015364	0,000221
10	-0,015143	-0,015364	0,000221
11	-0,071521	-0,071777	0,000256
12	-0,065724	-0,066010	0,000286
13	-0,073701	-0,074063	0,000362
14	-0,076394	-0,076755	0,000361
15	-0,076394	-0,076755	0,000361
16	-0,076394	-0,076755	0,000361
17	-2,786194	-2,787994	0,001800
18	5,447393	5,445367	0,002026
19	6,464826	6,462720	0,002106
20	6,464826	6,462720	0,002106
21	8,736275	8,734182	0,002093
22	8,786253	8,784097	0,002156
23	9,093118	9,090798	0,002320
24	9,093118	9,090798	0,002320
25	9,093118	9,090798	0,002320
26	9,093118	9,090798	0,002320
27	9,731165	9,728045	0,003120
28	9,922033	9,919843	0,002190
29	23,037014	23,034866	0,002148
30	25,794140	25,792002	0,002138
31	25,794140	25,792002	0,002138
32	30,444550	30,442816	0,001734
33	30,280426	30,279373	0,001053

Tabela 7.8. Comparação dos resultados de estimação da variável de estado θ para diferentes implementações

Número de iteração	Em *hardware* - MHU (graus)	Em *software* - Nios II (graus)	Diferença (graus)
1	0,000000	0,000002	0,000002
2	0,000000	0,000004	0,000004
3	0,000000	0,001014	0,001014
4	0,000000	-0,000455	0,000455
5	0,000000	-0,001903	0,001903
6	0,000000	-0,001306	0,001306
7	0,000000	-0,000126	0,000126
8	-0,219537	-0,219663	0,000126
9	-0,658612	-0,658738	0,000126
10	-0,658612	-0,658738	0,000126
11	-0,666787	-0,665790	0,000997
12	-0,666751	-0,664626	0,002125
13	-0,666755	-0,665812	0,000943
14	33,910397	33,911346	0,000949
15	33,910397	33,911346	0,000949
16	33,910397	33,911346	0,000949
17	33,786606	33,788887	0,002281
18	33,347530	33,349815	0,002285
19	33,347530	33,349815	0,002285
20	33,347530	33,349815	0,002285
21	33,347511	33,349117	0,001606
22	33,347511	33,347782	0,000271
23	33,347507	33,348495	0,000988
24	75,059624	75,060616	0,000992
25	75,059624	75,060616	0,000992
26	75,059624	75,060616	0,000992
27	75,093346	75,092972	0,000374
28	75,094185	75,093277	0,000908
29	75,094185	75,093277	0,000908
30	74,984413	74,983505	0,000908
31	74,984413	74,983505	0,000908
32	74,981308	74,978897	0,002411
33	74,981316	74,979614	0,001702

Tabela 7.9. Comparação dos resultados de implementações em *hardware* e em Matlab® para uma iteração do EKF

Variável	Em *hardware*	Em Matlab®	Erro
X_k^-:			
x (cm)	35,727600	35,727600	0,0000001
y (cm)	7,081845	7,081850	0,0000049
θ (graus)	27,414942	27,416030	0,0010886
P_k^-:			
$P^-[1][1]$	1,559276	1,559280	0,0000041
$P^-[1][2]$	1,135520	0,651990	0,4835304
$P^-[1][3]$	0,000226	0,000230	0,0000043
$P^-[2][1]$	1,135520	0,651990	0,4835304
$P^-[2][2]$	0,957242	0,448640	0,5086019
$P^-[2][3]$	0,000433	0,000030	0,0004027
$P^-[3][1]$	0,000226	0,000230	0,0000043
$P^-[3][2]$	0,000433	0,000030	0,0004027
$P^-[3][3]$	0,000008	0,000010	0,0000023
G_k:			
$G[1][1]$	-0,042870	-0,042950	0,0000803
$G[1][2]$	-0,005970	-0,003230	0,0027399
$G[2][1]$	-0,031822	-0,020390	0,0114324
$G[2][2]$	-0,009240	-0,003560	0,0056800
$G[3][1]$	0,000011	0,000000	0,0000105
$G[3][2]$	-0,000091	-0,000090	0,0000007
P_k^+ :			
$P^+[1][1]$	0,723968	0,843800	0,1198320
$P^+[1][2]$	0,494085	0,312020	0,1820652
$P^+[1][3]$	0,000021	0,000260	0,0002389
$P^+[2][1]$	0,494085	0,312020	0,1820651
$P^+[2][2]$	0,451305	0,284820	0,1664854
$P^+[2][3]$	0,000019	-0,000060	0,0000792
$P^+[3][1]$	0,000021	0,000260	0,0002389
$P^+[3][2]$	0,000019	-0,000060	0,0000792
$P^+[3][3]$	0,000003	0,000000	0,0000027
X_k^+:			
x (cm)	35,235385	35,239700	0,0043149
y (cm)	6,707285	6,846350	0,1390655
θ (graus)	27,383372	27,335816	0,0475555

cada variável de estado (x, y, θ) são mostrados na Figura 7.6 (a), (b), (c) e a Figura 7.6. (d) apresenta os resultados da estimação de pose no plano $Xg - Yg$.

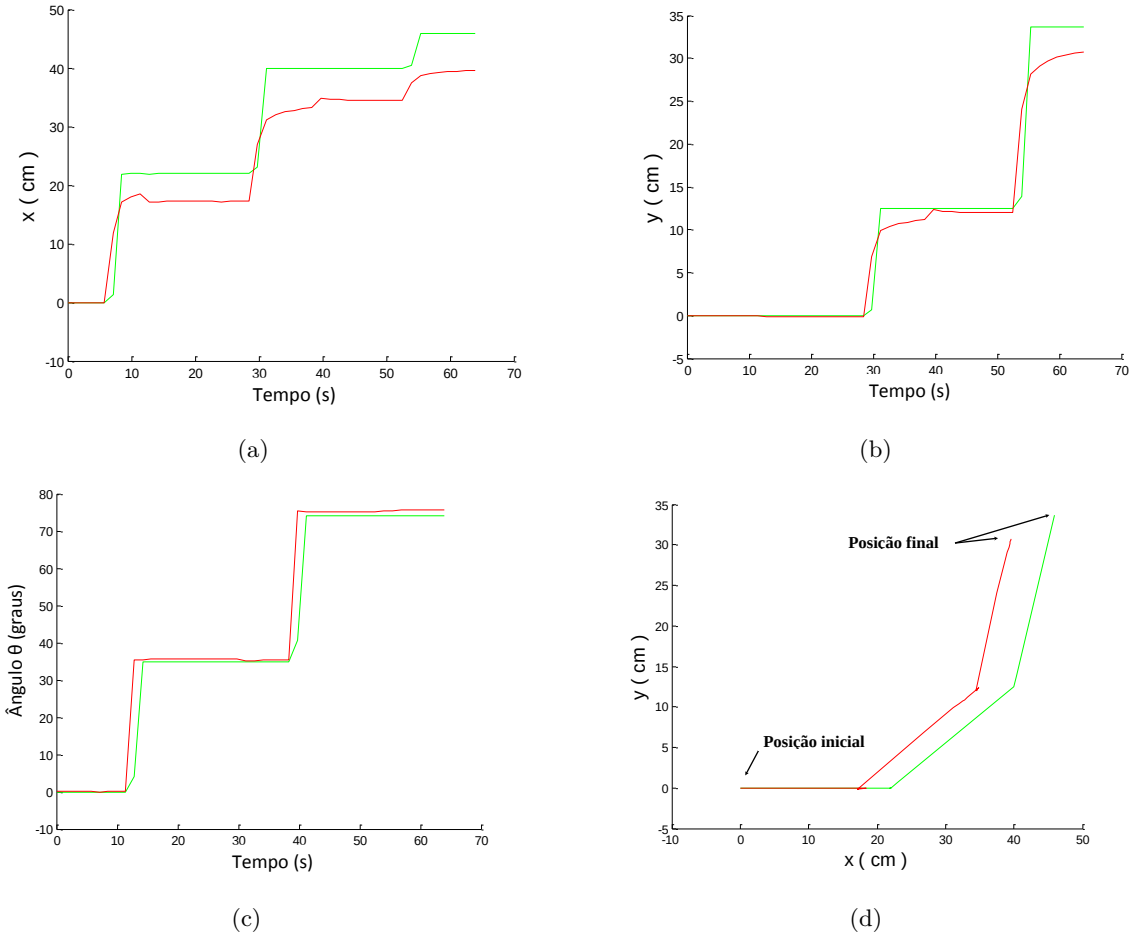


Figura 7.6. Resultados de estimação da posição do robô (a) x, (b) y e (c) θ usando 1 feixe de laser. (d) Estimativa de pose no plano $Xg - Yg$. A linha vermelha representa a implementação em *hardware* proposta e a linha verde corresponde ao sistema de localização interno do P3-AT.

7.4.2.2 Usando dois (2) feixes de *ladar* no sistema de medição

Neste teste se utilizou dois feixes de laser no sistema de medição, o primeiro feixe orientado com ângulo $\beta_i = 0^o$ e o segundo com um ângulo de $\beta_i = 10^o$ para a frente do robô como é mostrado na Figura 7.7). Os resultados da estimação de cada variável de estado (x, y, θ) e pose no plano $Xg - Yg$ são mostrados nas Figuras 7.8 (a), (b), (c) e (d).

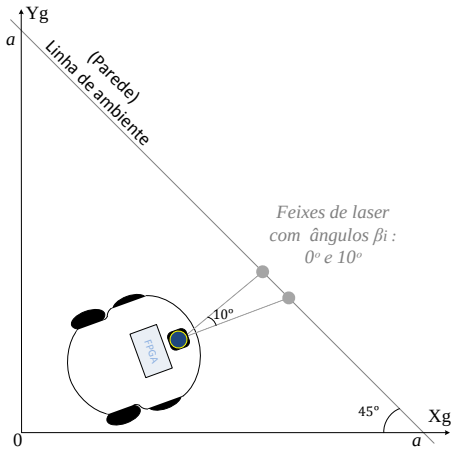


Figura 7.7. Test usando dois (2) feixes de *ladar*.

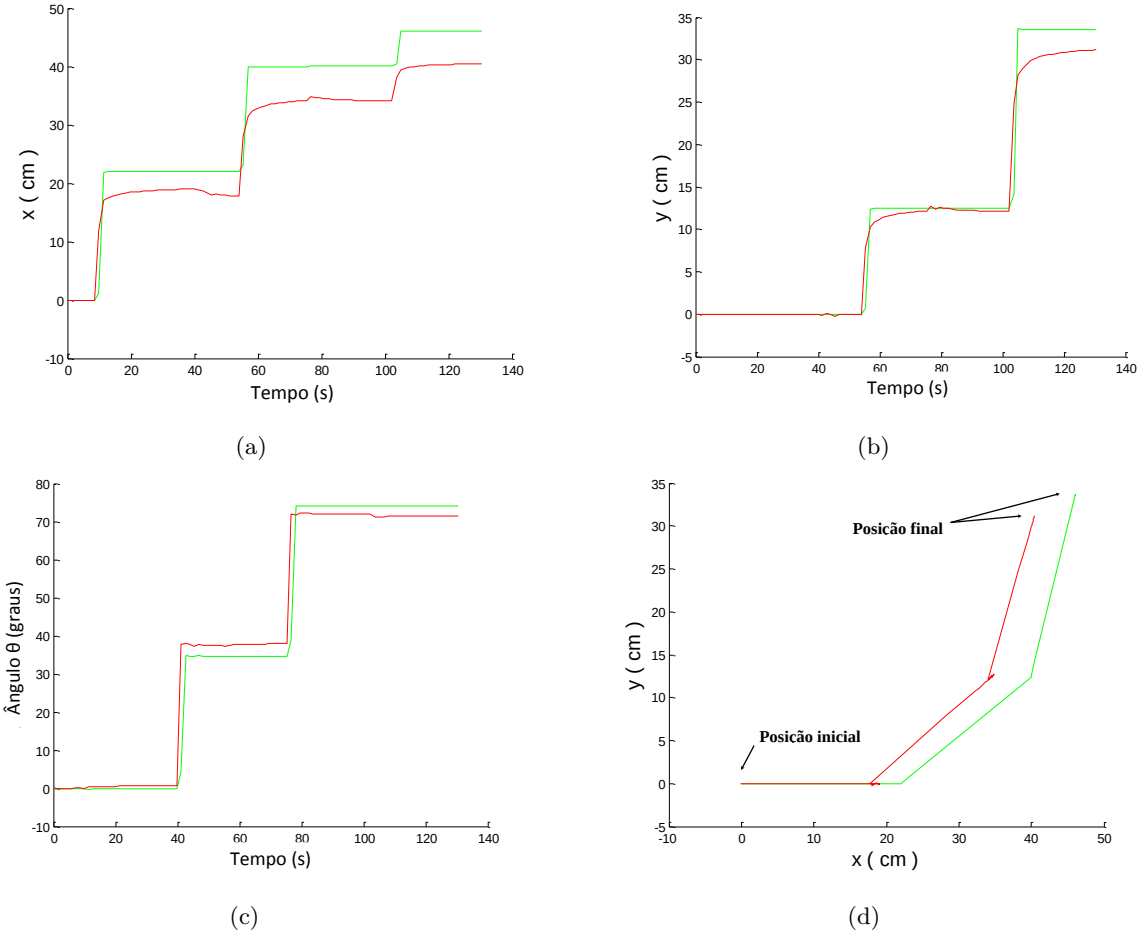


Figura 7.8. Resultados de estimação da posição do robô (a) x, (b) y e (c) θ usando 2 feixes de laser. (d) Estimativa de pose no plano $Xg - Yg$. A linha vermelha representa a implementação em *hardware* proposta e a linha verde corresponde ao sistema de localização interno do P3-AT.

7.4.2.3 Usando três (3) feixes de *ladar* no sistema de medição

O terceiro teste consistiu em utilizar três feixes de laser para o sistema de medição, onde o primeiro feixe foi orientado com ângulo $\beta_i = 0^o$ para a frente do robô, o segundo feixe com um ângulo de $\beta_i = 10^o$ e o terceiro feixe com um ângulo de $\beta_i = 30^o$ como é apresentado na Figura 7.9). Para este caso, os resultados da estimação de cada variável de estado (x, y, θ) são indicados nas Figuras 7.10 (a), (b), (c) e a Figura 7.10. (d) apresenta os resultados da estimação de pose no plano $Xg - Yg$.

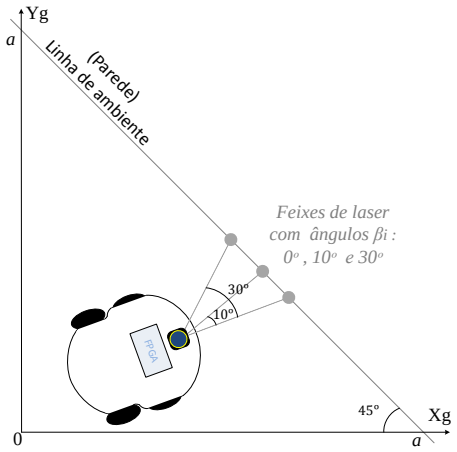


Figura 7.9. Test usando três (3) feixes de *ladar*.

A partir das Figuras 7.6, 7.8 e 7.10, pode-se observar que as estimativas de pose do robô por meio da implementação em *hardware* vai se aproximando à estimativa do sistema interno do Pioneer a medida que o numero de feixes de laser aumenta. É possível também perceber que na implementação proposta, que quando é usado um numero maior de feixes, nos resultados tendem a aparecer variações tipo *serra* em cada uma das variáveis de posição (x, y e θ) durante o tempo, como é o caso da Figura 7.10 (a) para a variável x ao-redor do tempo $110s$, isto é devido a que em cada iteração a implementação proposta utiliza o enfoque do EKF sequencial, o qual em uma primeira etapa *prediz* a pose do robô a partir do modelo odométrico proposto, e em uma segunda etapa *estima* ou atualiza a pose do robô a partir do modelo de medição baseado em LRF. Enquanto que os resultados do sistema próprio do P3-AT não apresentam essas variações mencionadas anteriormente, tendo em vista que seu sistema de localização é baseado especificamente em odometria (*encoders*) e no uso de um giroscópio interno para a correção da orientação angular do robô móvel.

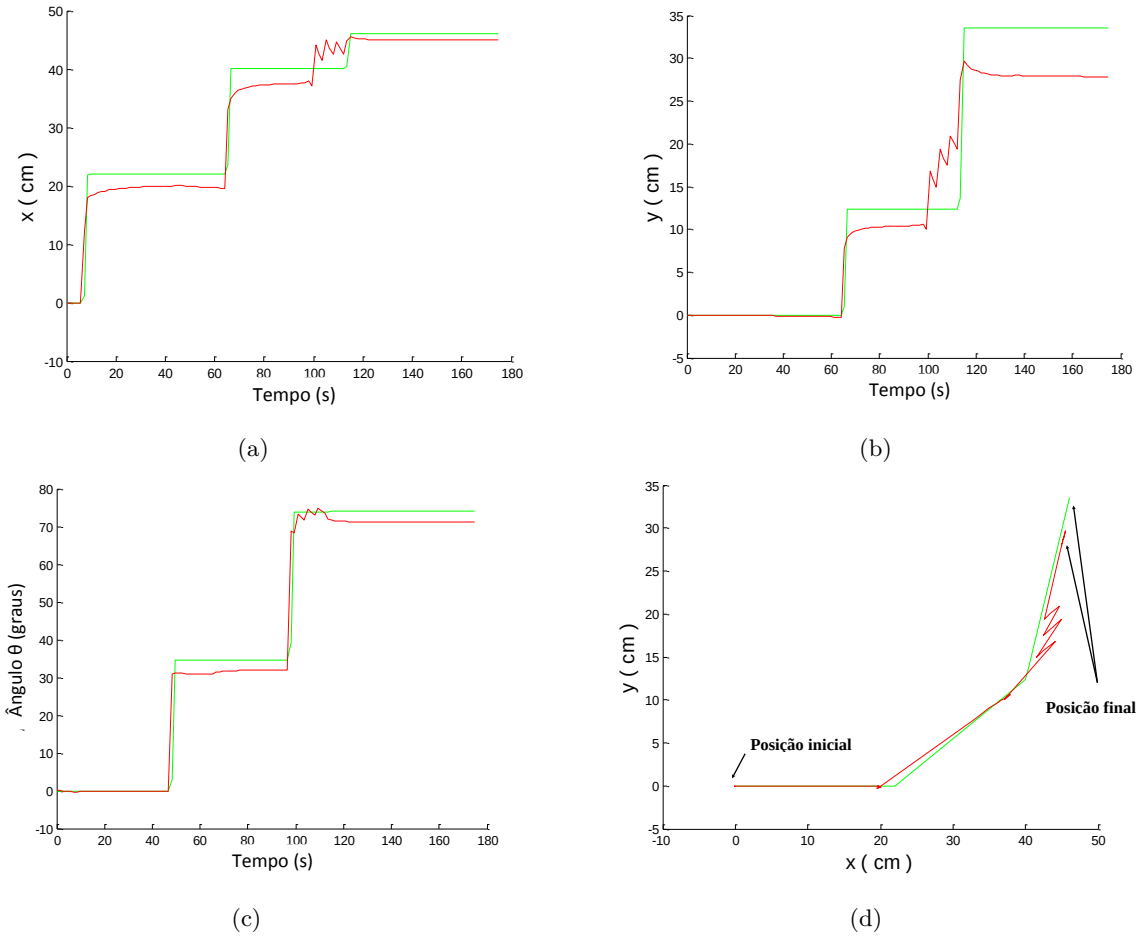


Figura 7.10. Resultados de estimação da posição do robô (a) x, (b) y e (c) θ usando 3 feixes de laser. (d) Estimativa de pose no plano $Xg - Yg$. A linha vermelha representa a implementação em *hardware* proposta e a linha verde corresponde ao sistema de localização interno do P3-AT.

7.5 Comentários gerais sobre os resultados

Os resultados de síntese mostraram que tanto a primeira quanto a segunda abordagem permitiam sem problemas a implementação completa do EKF (ambas etapas) no FPGA EP4CE115F29C7 - Cyclone IV E, já que o porcentagem de uso de elementos lógicos (LEs) para esse dispositivo foram de aproximadamente 33% e 24% para cada abordagem respetivamente. Adicionalmente, o funcionamento correto das arquiteturas propostas também foi conferido a partir das suas simulações feitas usando a ferramental ModelSim-Altera Start Edition.

Este funcionamento também foi validado rodando as implementações das abordagens no mesmo FPGA EP4CE115F29C7 - Cyclone IV E. Nesses casos, o desempenho das implementações propostas foram comparados com implementações puramente em *software*, em termos de tempo de execução e de resultados de estimação de posição do robô. A implementações propostas

mostraram sua vantagem, principalmente por obter menores tempos de execução se comparados com as outras soluções em *software*. Além disso, as mesmas demostraram que os seus resultados de estimação da pose do robô são tão bons quanto os resultados obtidos mediante só *software* (procesador Nios II).

Adicionalmente, com o objetivo de validar o sistema proposto para aplicações de robótica móvel, foram feitos testes de comparação entre os resultados do sistema de localização baseado na arquitetura de *hardware* com os resultados do sistema de localização próprio do P3-AT. Nestes testes, nosso sistema obteve resultados muito parecidos aos do sistema próprio do P3-AT (durante as seções de provas), mostrando uma aproximação maior entre os resultados comparados, a medida que se decidiu usar mais feixes de laser para o sistema de medição da arquitetura em *hardware*.

De acordo a todos estes testes realizados, é importante indicar que o modelamento do robô móvel joga um papel decisivo no funcionamento de algoritmo EKF. O modelo recebe um forte impacto de acordo aos parâmetros programados, relacionando aspectos como nível de pressão nos pneus, derrapagem, entre outros. De todos estes, o modelo é influenciado basicamente pelo efeito de derrapagem existente entre as rodas do SSMR e o chão. Nesse sentido, é possível observar que para cada tipo de chão, o modelo do sistema será afetado na mudança de alguns do seus parâmetros dentro dele, o qual é uma importante consideração no momento de realizar os testes e comparações com outra soluções.

Por outro lado, cabe lembrar que o propósito deste trabalho de dissertação de mestrado foi implementar o algoritmo do Filtro de Kalman Estendido versão sequencial em um FPGA Cyclone IV - Altera, para montar a mesma em uma plataforma móvel Pioneer 3-AT só para fins de validação experimental, visando ter como foco futuro o desenvolvimento de projetos na área de micro-robótica.

Nesse sentido, é importante mencionar, que a partir do analise dos resultados de síntese para outros FPGAs, estes mostraram algumas restrições. Este foi o caso do FPGA Cyclone IV EP4CE22F17C6N, disponível no kit de desenvolvimento DE0_Nano, o qual se tinha pensado usar para aplicações de micro-robôs, devido a seu pequeno porte a fácil portabilidade. Os resultados de síntese para este tipo de FPGA indicaram que implementação completa do EKF (ambas etapas) não seria viável usando nenhuma das abordagens propostas, já que o consumo de recursos de *hardware* excede notavelmente os recursos disponíveis para esse dispositivo (vide Tabela 7.2).

Nesse caso, uma abordagem recomendável seria desenvolver um co-projeto HW-SW para esse tipo de FPGA, onde se use a arquitetura em *hardware* da etapa de *Predição* (proposta na primeira metodologia), e que a etapa de *Estimação* seja programada totalmente no processador

Nios II. Esta solução seria fatível, devido a que o módulo de *Predição* apresenta um porcentagem de recursos de 71% para esse tipo de FPGA, como pode ser visto Tabela 7.2.

7.6 Conclusões do Capítulo

Neste capítulo foram mostrados e discutidos os diferentes resultados para as arquiteturas propostas do algoritmo do Filtro de Kalman Estendido. Primeiramente, viu-se os resultados de recurso de *hardware* e consumo de potência. Para este caso, o resultado os recursos consumidos foram exibidos em termos de elementos lógicos, DSPs, potência e frequência máxima de operação das arquiteturas. Aqui, a primeira abordagem mostrou que a sua vantagem principal está na modularização das etapas EKF, devido a que ao trabalhar com FPGAs de baixa densidade, pode-se implementar uma das etapas em *hardware* enquanto que a outra implementa-se em *software*.

Adicionalmente, foram mostrados os resultados de simulação comportamental das arquiteturas propostas. Verificou-se que a arquitetura da etapa de *Predição* tem um menor tempo de execução, enquanto que a arquitetura da *Estimação* tem o maior tempo. Esse resultado indica que a arquitetura do *Estimação* é uma forte candidata para se implementar em *hardware*, devido a que apresenta um tempo de simulação maior.

A seguir, foi visto uma comparação do desempenho (tempo de execução) das implementações do algoritmo EKF com relação à primeira e segunda abordagem em *hardware*, e com outras soluções totalmente em *software*. Nesse sentido, apreciam-se fatores de aceleração superiores a 3 das implementações em *hardware* em relação às soluções executadas totalmente em *software*.

Finalmente, mostraram-se os resultados de validação experimental das arquiteturas propostas para um caso real, verificando-se que os resultados das arquiteturas são similares aos resultados obtidos com uma solução em *software*. Também foram comparados os resultados de estimação de pose das arquiteturas propostas (com diferentes números de feixes de LRF) com os resultados do sistema de localização interno do P3-AT, demostrando assim a adaptabilidade da arquitetura proposta em aplicações de robótica móvel.

8 Conclusões e Trabalhos Futuros

8.1 Aspectos gerais

Neste trabalho foi apresentada uma de arquitetura em *hardware* (baseada em FPGA) do EKF sequencial para resolver o problema de localização em robótica móvel. No inicio, foi realizada uma revisão bibliográfica da literatura relacionada com projetos de implementação em FPGAs de algoritmos de localização robótica, assim como outras implementações focadas no Filtro de Kalman e no tema fusão sensorial. Seguidamente, foram apresentadas as arquiteturas das Unidades em Ponto Flutuante (UPFs), as quais foram usadas no algoritmo EKF para realizar operações aritméticas e trigonométricas em ponto flutuante.

8.2 Conclusões e comentários finais

A partir dos objetivos específicos definidos neste trabalho, observou-se o seguinte:

- Foi obtido o modelo do sistema correspondente ao robô móvel P3-AT de tipo SSMR.
- Foi adaptado o modelo de medição desenvolvido em [5] para um ambiente especifico, tendo em conta o uso de um sensor *ladar*.
- Foram desenvolvidas as equações do algoritmo EKF a partir dos modelos de processo e medição obtidos previamente.
- Foi implementado em *software* o algoritmo de EKF aplicado para localização robótica, para fins de validação dos modelos.
- Foi desenvolvido um módulo de *hardware* da etapa de predição do algoritmo EKF.
- Foi desenvolvido um módulo de *hardware* unificado (MHU) da versão sequencial de algoritmo EKF, considerando as cinco equações do filtro.
- Foi implementado em *hardware* o algoritmo EKF rodando em um FPGA, usando diferentes abordagens para solucionar o problema de localização.

- Foram validados as implementações em *hardware* do algoritmo EKF, comparando seus resultados com os resultados de implementações totalmente em *software*.
- Foi criado um ambiente de teste para a plataforma Pioneer 3-AT, com fins de validar a aplicabilidade da arquitetura proposta em robótica móvel.

Com respeito ao desenvolvimento da arquitetura do algoritmo EKF, o mesmo foi conseguido a partir de duas abordagens ou metodologias. A primeira abordagem, consistiu em usar módulos de *hardware* individuais para cada etapa do EKF. Para isto, projetou-se uma arquitetura para a etapa de *predição*, a qual foi conectada a outra arquitetura correspondente à etapa de *estimação* previamente desenvolvida [5], permitindo a implementação completa do algoritmo EKF em *hardware*. Em enquanto para a segunda abordagem, implementou-se uma única arquitetura do EKF (unificando as duas etapas) recebendo o nome de MHU.

Adicionalmente, foi visto neste trabalho o consumo de recursos de *hardware* e potência, bem como a *performance* (desempenho) das implementações do EKF. Os resultados apontaram para um desempenho satisfatório da implementação do algoritmo EKF (usando as duas abordagens) em um FPGA EP4CE115F29C7 - Cyclone IV E, mostrando pouco consumo de *hardware* que é consequência direta da versão sequencial do algoritmo.

O tempo de execução das implementações também foi analisado, demostrando que as implementações em *hardware* conseguem acelerar o processo do algoritmo EKF comparado com uma implementação totalmente em *software*. Neste analise, aprecia-se um decaimento do desempenho, devido ao tempo de escrita/leitura na arquitetura em *hardware* via o barramento Avalon.

Em relação à validação da arquitetura proposta, com o objetivo de provar o funcionamento apropriado da mesma, foi realizada uma comparação entre resultados usando implementações em *hardware* com resultados mediante soluções puramente em *software*, onde a comparação mostrou valores de estimação de localização próximos durante a sessão de teste para ambas soluções. Além disso, foi demostrada a aplicabilidade da arquitetura proposta para localização de robôs moveis, já que os resultados de estimação da arquitetura MHU (usando diferentes números de feixes na medição) foram comparados com os resultados do sistema de localização interno do P3-AT.

Para o nosso conhecimento, não há outros trabalhos que propõem este tipo de abordagem de *hardware* para execução total do algoritmo EKF aplicado à robótica móvel, analisando seus impactos em diferentes aspectos e tendo em mente aplicar este enfoque na área de micro-robótica, para pequenos robôs com restrições no consumo de recursos e energia.

8.3 Propostas de Trabalhos Futuros

Como trabalhos futuros, sugere-se:

- A implementação e adaptação das arquiteturas propostas (MHU, módulo de Predição e/ou Estimação) para plataformas robóticas com pequena dimensão e consumo, usados na área da micro-robótica;
- A implementação de um periférico DMA (do inglês ***D****irect* ***M****emory* ***A****ccess*) para transferência de dados entre *hardware* e *software*, com a ideia de diminuir o tempo de execução da implementação HW/SW;
- A implementação do algoritmo EKF em *hardware* adicionando sensores como o caso de um giroscópio, com objetivo de melhorar a etapa de estimação do algoritmo;
- A implementação de um módulo dedicado à tarefa de mapeamento, para ser usado com o módulo de localização proposto no presente trabalho, tendo em mente abordar a questão do SLAM (localização e mapeamento simultâneo);
- A implementação do algoritmo UKF em *hardware* orientado para problemas de robótica móvel.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SIEGWART, R.; NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots*. Cambridge, MA, USA: The MIT Press, 2004.

[2] KOZLOWSKI, K.; PAZDERSKI, D. Modeling and control of a 4-wheel skid-steering mobile robot. *International Journal of Applied Mathematics and Computer Science*, University of Zielona Gora Press, v. 14, n. 4, p. 477–496, 2004.

[3] MUNOZ, D. M. *Otimização por inteligência de exames usando arquiteturas paralelas para aplicações embarcadas*. Tese (Doutorado) — Universidade de Brasília, Brazil, 2012.

[4] MUNOZ, D. M. *et al.* FPGA-based floating-point library for CORDIC algorithms. In: *Programmable Logic Conference (SPL), 2010 VI Southern*. Porto de Galinhas, Brazil: [s.n.], 2010. p. 55–60.

[5] CRUZ, S. M. *Implementação de um Filtro de Kalman Estendido em Arquiteturas Reconfiguráveis ao problema de localização em Robótica Móvel*. Dissertação (Mestrado) — Departamento da Engenharia Mecânica - Universidade de Brasília, Brasília, DF, Brasil, Abril 2013.

[6] VENTURA, J. M. V. *Estudo de Coordenação de Robôs Móveis com Obstáculos*. Dissertação (Mestrado) — Escola de Engenharia de São Carlos - Universidade de São Paulo, São Carlos, SP, Brasil, Setembro 2011.

[7] BATURONE, I.; GERSNOVIEZ, A.; BARRIGA, A. Neuro-fuzzy techniques to optimize an fpgaembedded controller for robot navigation. *AppliedSoftComputing*, Elsevier, v. 21, p. 95–1062, 2014.

[8] RODRIGUES, J.; FERREIRA, J. FPGA-based rectification of stereo images. In: *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*. [S.l.: s.n.], 2010. p. 199–206.

[9] CRUZ, S. *et al.* 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*. [S.l.: s.n.], 2013. p. 306–313.

[10] CONDE, M. *et al.* An efficient data fusion architecture for infrared and ultrasonic sensors, using FPGA. In: *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*. [S.l.: s.n.], 2013. p. 1–4.

[11] WOF, W. *High-Performance Embedded Computing, Architectures, Applications, and Methodologies*. [S.l.]: Elsevier Science, 2007.

[12] PTC. Systems engineering: Development of mechatronics and software need to be integrated closely. In: . [S.l.]: Parametric Technology Corporation (PTC), 2012.

[13] MATLAB. *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[14] BARROS, E. *et al. Hardware/software co-design: projetando hardware e software concorrentemente*. [S.l.]: IME-USP, 2000.

[15] HARTENSTEIN, R. Why we need reconfigurable computing education. In: *1st International Workshop on Reconfigurable Computing Education*. Karlsruhe, Germany: [s.n.], 2006. Disponível em: <http://fpl.org/RCeducation/>.

[16] SASS, R.; SCHMIDT, A. *Embedded Systems Design with Platform FPGAs: Principles and Practices*. Elsevier Science, 2010. (Titolo collana). ISBN 9780123743336. Disponível em: <http://books.google.com.br/books?id=Ki7zs-Ex2d0C>.

[17] GAJSKI, D. *et al. Embedded System Design: Modeling, Synthesis and Verification*. New York, NY, USA: Springer, 2009. ISBN 9781441905031.

[18] SAMSUNG TELECOMMUNICATIONS AMERICA. *"Samsung Galaxy S5 - User Manual"*. 2014. Disponível em: <http://www.samsung.com/us/mobile/cell-phones/SM-G900TRKATMB>.

[19] PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN 0124077269, 9780124077263.

[20] BONATO, V.; MARQUES, E.; CONSTANTINIDES, G. A floating-point extended kalman filter implementation for autonomous mobile robots. In: *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. [S.l.: s.n.], 2007. p. 576 –579.

[21] BONATO, V. *Proposal of an FPGA hardware architecture for SLAM using multi-cameras and applied to mobile robotics*. Tese (Doutorado) — Institute of Computer Science and Computational Mathematics - University of São Paulo, São Carlos, SP, Brazil, January 2008.

[22] TERASIC. *"Altera DE-115 Development and Education Board"*. 2012. Disponível em: <http://www.terasic.com.tw>.

[23] ADEPT MOBILE ROBOTS. *"P3-AT Datasheet"*. 2012. Disponível em: <http://www.mobilerobots.com/Libraries/Downloads>.

[24] CONDE, M. *Implementação da Técnica de Zona Virtual Deformável em um Sistema Embarcado Baseado em FPGA*. Dissertação (Mestrado) — Departamento da Engenharia Mecânica - Universidade de Brasília, Brasília, DF, Brasil, Dezembro 2013.

[25] KAM, M.; ZHU, X.; KALATA, P. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, v. 85, n. 1, p. 108 – 119, jan. 1997. ISSN 0018-9219.

[26] THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics*. Cambridge, MA, USA: The MIT Press, 2005.

[27] BONATO, V. *et al.* An FPGA implementation for a kalman filter with application to mobile robotics. In: *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*. [S.l.: s.n.], 2007. p. 148 –155.

[28] CRUZ, S. *et al.* FPGA implementation of a sequential Extended Kalman Filter algorithm applied to mobile robotics localization problem. In: *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*. [S.l.: s.n.], 2013. p. 1–4.

[29] DELLAERT, F. *et al.* Monte Carlo localization for mobile robots. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. [S.l.: s.n.], 1999. v. 2, p. 1322 –1328 vol.2. ISSN 1050-4729.

[30] COUTO, L. N. *Sistema para localização robótica de veículos autônomos baseado em visão computacional por pontos de referência*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, São Carlos, SP, Brasil, Julho 2012.

[31] KALMAN, R. E. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, v. 82 (Series D), p. 35–45, 1960.

[32] MINGAS, G.; TSARDOULIAS, E.; PETROU, L. An FPGA implementation of the SMG-SLAM algorithm. *Microprocessors and Microsystems*, v. 36, n. 3, p. 190 – 204, 2012. ISSN 0141-9331. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0141933111001244>.

[33] GHORBEL, A. *et al.* A HW/SW implementation on FPGA of a robot localization algorithm. In: *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*. [S.l.: s.n.], 2012. p. 1 –7.

[34] LIU, Y.; BOUGANIS, C.-S.; CHEUNG, P. Efficient mapping of a Kalman filter into an FPGA using Taylor Expansion. In: *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on.* [S.l.: s.n.], 2007. p. 345 –350.

[35] BIERMAN, G. J. *Factorization methods for discrete sequential estimation.* [S.l.]: Academic Press, 1977.

[36] THORNTON, C. L. *Triangular covariance factorizations for kalman filtering.* Tese (Doutorado) — School of Engineering - University of California, 1976.

[37] MANDOW, A. *et al.* Experimental kinematics for wheeled skid-steer mobile robots. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on.* [S.l.: s.n.], 2007. p. 1222–1227.

[38] DIXON, W. *et al. Nonlinear Control of Wheeled Mobile Robots.* [S.l.]: Springer-Verlag, 2001.

[39] DIXON, W. *et al. Nonlinear Control of Engineering Systems, A Lyapunov-Based Approach.* Boston, USA: Birkhauser, 2003.

[40] PEDRAZA, S. *et al.* A motion control approach for a tracked mobile robot. In: *Proc. of the 4th IFAC International Symposium on Intelligent Components and Instruments for Control Applications.* Buenos Aires, Argentina: [s.n.], 2000. p. 147–152.

[41] PAZDERSKI, D.; KOZLOWSKI, K.; LAWNICZAK, M. Practical stabilization of 4wd skidsteering mobile robot. In: *Proc. of the Fourth International Workshop on Robot Motion and Control.* Puszczykowo: [s.n.], 2004. p. 175–180.

[42] PAZDERSKI, D.; KOZLOWSKI, K.; DIXON, W. Tracking and regulation control of a skid steering vehicle. In: *American Nuclear Society Tenth International Topical Meeting on Robotics and Remote Systems.* Gainesville, Florida: [s.n.], March 28–April 1 2004. p. 369–376.

[43] AGUIRRE, L. A. *Introdução à Identificação de Sistemas: Técnicas Lineares e Não-Lineares Aplicadas a Sistemas Reais.* 3rd. ed. Belo Horizonte, Brasil: Editora UFMG, 2007.

[44] ARIAS-GARCIA, J. *et al.* A suitable FPGA implementation of floating-point matrix inversion based on Gauss-Jordan elimination. In: *Programmable Logic (SPL), 2011 VII Southern Conference on.* [S.l.: s.n.], 2011. p. 263 –268.

[45] WANG, X. *Variable Precision Floating-point Divide and Square Root for Efficient FPGA Implementation of Image and Signal Processing Algorithms.* Tese (Doutorado) — Northeastern University, USA, 2007.

[46] ALTERA. *"What is an FPGA?"*. 2012. Disponível em: <http://www.altera.com>.

[47] VALLEJO, M. L. L.; RODRIGO, J. L. A. *FPGA: Nociones básicas y Implementación*. [S.l.], April 2004. Department of Electronic Engineer, Polytechnic University of Madrid.

[48] SANCHEZ, D. F. *et al.* Parameterizable floating-point library for arithmetic operations in FPGAs. In: *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*. New York, NY, USA: ACM, 2009. (SBCCI '09), p. 40:1–40:6. ISBN 978-1-60558-705-9. Disponível em: <http://doi.acm.org/10.1145/1601896.1601948>.

[49] MUNOZ, D. M. *et al.* Tradeoff of FPGA design of a floating-point library for arithmetic operators. *Journal of Integrated Circuits and Systems*, v. 5, n. 1, p. 42–52, 2010.

[50] MARKSTEIN, P. Software division and square root using goldschmidt's algorithms. In: *Proc. Conference on Real Nambers ans Computers*. Dagstuhl, Germany: RNC, 2004. p. 146–157.

[51] ALTERA. *"Quartus II Handbook"*. 2012. Disponível em: <http://www.altera.com/literature/>.

[52] ALTERA. *"ModelSim-Altera Start Edition"*. 2012. Disponível em: <www.altera.com/literature/>.

Anexos

A. SENSORES

A.1 *Encoder*

Um *Encoder* é um sensor eletro-opto-mecânico que unido a um eixo, proporciona informação da posição ou velocidade de um motor ou roda (dependendo da aplicação). Tem como objetivo, atuar como um dispositivo de realimentação em sistemas de controle integrado.

Os *encoders* são muito utilizados como instrumentos auxiliares na área de robótica, lentes fotográficas, aplicações industriais que requerem medição angular, militares, etc.

Já na área da robótica móvel, estes sensores se utilizam diretamente acoplados aos eixos das rodas dos veículos basicamente para tarefas de auto-localização a partir de procedimentos de *odometria* (ou em inglês *dead-reckoning*).

Quanto a sua classificação podem ser de dois tipos: incremental ou absoluto. O *encoder* absoluto mede a posição angular e infere a velocidade. Já, o *encoder* incremental (como mostra a Figura A.1) mede a velocidade rotacional e pode inferir a posição relativa à última medida.

Figura A.1. Exemplo de um *encoder* incremental

A.1.1 Princípio de Funcionamento

O princípio básico de funcionamento é o seguinte: o sistema de leitura é baseado em um disco, formado por janelas radiais transparentes e opacas, alternadas. Este é iluminado perpendicularmente por uma fonte de luz infravermelha, quando então, as imagens das janelas transparentes são projetadas no receptor. O receptor converte essas janelas de luz em pulsos

elétricos (ver Figura A.2)

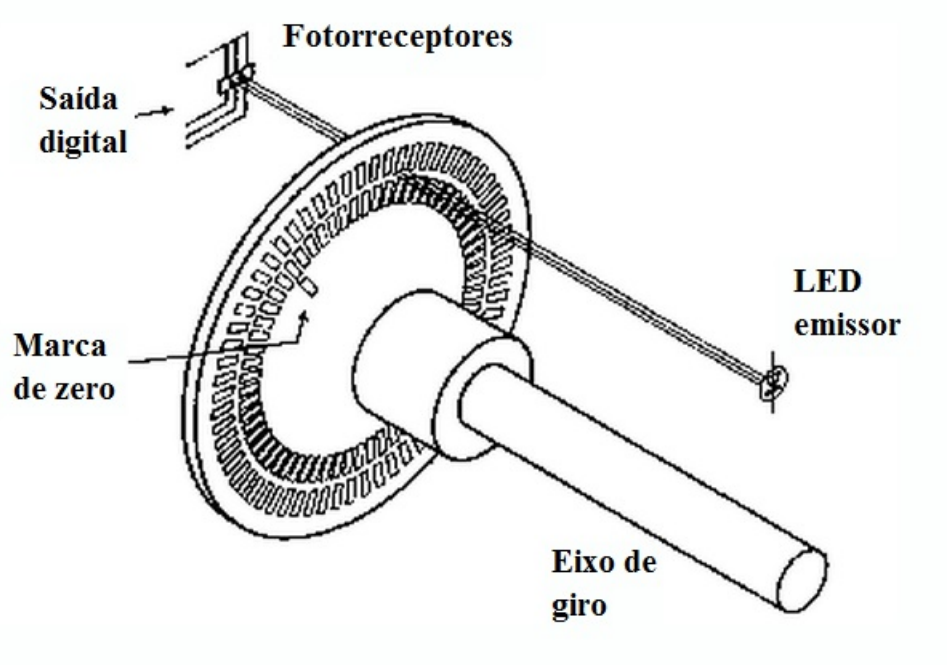


Figura A.2. Princípio de funcionamento de um *encoder*

Cálculo das velocidades de deslocamento linear das rodas

Neste trabalho, são usados *encoders* incrementais nos eixos das rodas do robô Pioneer. Na maioria destes dispositivos, a saída é um número inteiro proporcional à rotação angular do eixo.

As rotações angulares das rodas são contínuas, no entanto, os *encoders* proveem saídas de tipo discretas. Considere-se que os sinais contínuos e_L e e_R (correspondente ao *encoder* esquerdo e direito respectivamente) são quantizados e serão os sinais de saída $\overline{e_L}$ e $\overline{e_R}$ dos *encoders*, as quais serão utilizados na implementação das equações de localização robótica.

A Figura A.3 mostra um exemplo da função de quantização para um *encoder* de resolução igual a 16 (medida em ciclos por revolução). Esta resolução do *encoder* será nomeada de CR. Adicionalmente, chamaremos de variáveis θ_L e θ_R à rotação angular (em radianos) das rodas esquerda e direita, respectivamente.

Logo, a saída dos *encoders* é governada pelas Equações (A.1) e (A.2):

$$\overline{e_L} = h(e_L) \tag{A.1}$$

$$\overline{e_R} = h(e_R) \tag{A.2}$$

onde os sinais e_L e e_R são valores reais e estão expressados como:

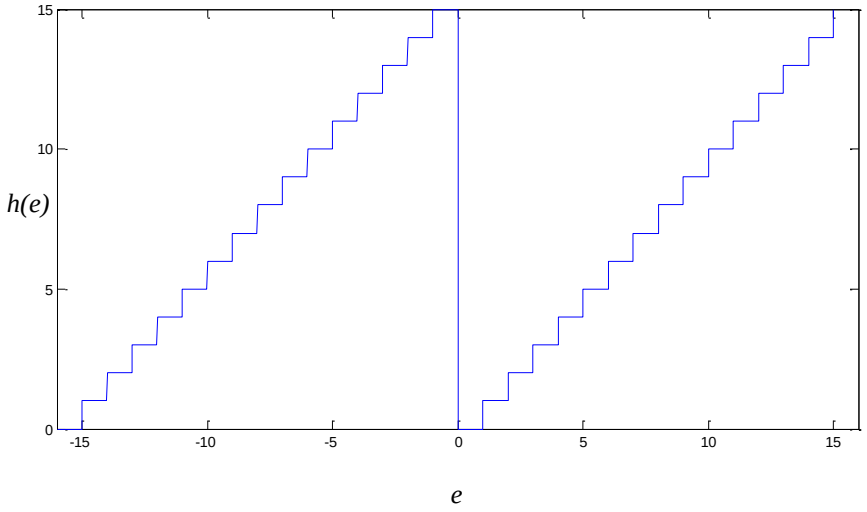


Figura A.3. Quantização da sinal de um *encoder*, adaptada de [6]

$$e_L = \frac{CR}{2\pi}\theta_L(t) \tag{A.3}$$

$$e_R = \frac{CR}{2\pi}\theta_R(t) \tag{A.4}$$

Derivando as Equações (A.3) e (A.4), e resolvendo para $\dot{\theta_L}$ e $\dot{\theta_R}$ se obtêm:

$$\dot{\theta_L} = \frac{2\pi}{CR}\dot{e_L} \tag{A.5}$$

$$\dot{\theta_R} = \frac{2\pi}{CR}\dot{e_R} \tag{A.6}$$

Finalmente, a partir das Equações (A.5) e (A.6) pode-se obter as velocidades de deslocamento linear V_L e V_R das rodas necessárias para o modelamento da plataforma móvel:

$$V_L = r\dot{\theta_L} \tag{A.7}$$

$$V_R = r\dot{\theta_R} \tag{A.8}$$

onde r é o valor do raio das rodas.

A.2 Sensor Laser - *Ladar*

Um *Ladar* (do inglês ***LA****ser* ***D****etection* ***A****nd* ***R****anging*), também conhecido como *Laser Rangefinder*, é uma tecnologia óptica de detecção remota que mede propriedades da luz refletida de modo a obter a distância a um determinado objeto e/ou também em alguns casos a velocidade dele. A sua resolução é pelo ordem dos milímetros e seu consumo de energia varia, mas tende a ser um pouco mais do que nos sensores menores e menos sofisticados, como o caso do *Sonar* (do inglês ***SO****und* ***NA****vigation and* ***R****anging*).

As aplicações do *ladar* estão presentes nas áreas da geodesia, arqueologia, geografia, geologia, geomorfologia, sismologia, engenharia florestal, oceanografia costeira, detecção remota, física da atmosfera e para fins militares. Também é usado como sensores na área de robótica móvel, devido à sua resolução, precisão e rápida geração de dados (a luz permite medir de uma forma extremamente rápida e com um campo de visão muito estreito). A Figura A.4 mostra o sensor *ladar* usado neste trabalho.

Figura A.4. *Ladar* Hokuyo URG-04LX

A.2.1 Princípio de Funcionamento do *Ladar*

O *ladar* emite e recebe dados em feixe de luz para calcular a distância do obstáculo. Para ter uma ideia mais detalhada sobre o funcionamento do sensor, o *ladar* emite um feixe de luz infravermelha e um espelho girante muda a direção desse feixe, como se apresenta na Figura A.5. O laser choca contra a superfície do objeto para logo ser refletido. A direção da luz refletida é novamente mudada pelo espelho e capturada por um fotodiodo, que fica dentro do sensor.

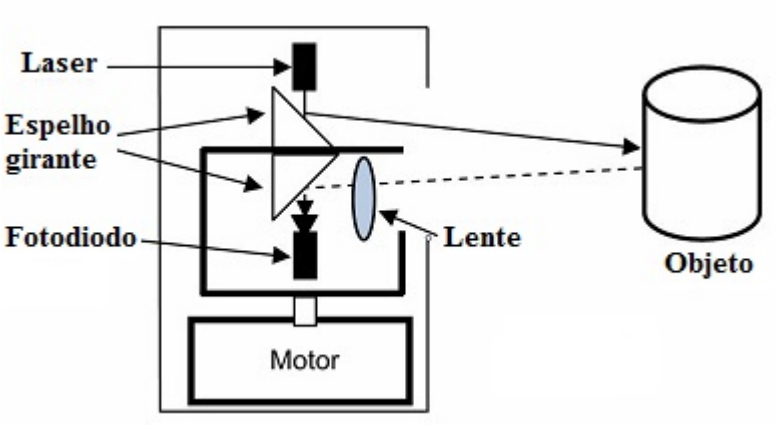


Figura A.5. Princípio de funcionamento de um *ladar*

É usado para criar mapas 2D do ambiente ao redor a partir da detecção de objetos próximos. Ele faz uma varredura do ambiente em um plano (vide Figura A.6), tendo uma leitura em um determinado angulo, passa uma fração da resolução angular total e depois toma uma outra leitura. Isso é feito de forma contínua até que se complete o campo de visão total dele, repetindo o processo varias vezes.

É importante sinalar que devido às dimensões e restrições mecânicas do dispositivo, fica uma região angular 'cega' como visto na Figura A.6.

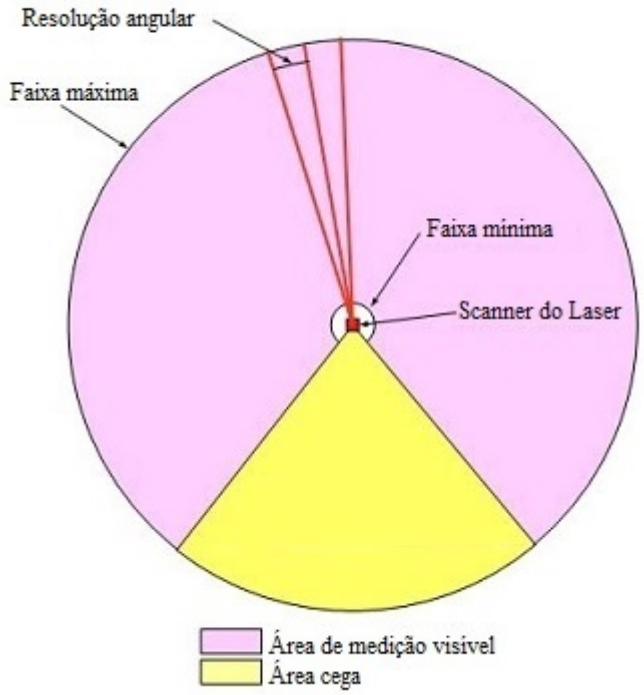


Figura A.6. Campo de visão de um *ladar*

Especificações

Com respeito às especificações dos *ladares*, eles são muitas vezes indicados pela sua resolução da medição, frequência e resolução angular. A resolução da medição esta relacionada como a medida exata da distância em uma determinada direção (usualmente entre 1 a 3 mm). A frequência se refere quantas vezes o arco de 360 graus é varrido pelo sensor em um segundo (comummente entre 10Hz ou até mais). A resolução angular refere-se para quantas amostras são tomadas a cada varredura completa de 360 graus do sensor. Para o *ladar* usado neste trabalho, a resolução angular é de 0.36 (vide Tabela A.1), valor que pode ser obtido de dividir 360 graus entre 10 que é o numero de bits do conversor A/D incluído no dispositivo. Finalmente na Tabela A.1 apresentamos as demais características do sensor *ladar* URG 04LX da empresa Hokuyo utilizado neste trabalho.

Tabela A.1. Especificações do *Ladar* URG 04LX

Especificações	
Voltage	5.0V +/- 5%
Corrente	0.5A nominal
Raio De Detecção	20mm a 5,6m
Comprimento de Onda	785nm, Classe 1
Angulo de Detecção	240°
Frequência	10Hz
Resolução	1mm
Precisão	De 20mm a 1m: +/- 30mm
	De 1m a 4m: +/- 3% da medição
Resolução Angular	0.36°
Interface	RS232 e USB
Peso	160g