USO DE SISTEMAS RECONFIGURÁVEIS PARA ACELERAÇÃO DO FILTRO DE KALMAN EM LOCALIZAÇÃO DE ROBÔS MÓVEIS

Sérgio M. Cruz*, Carlos H. Llanos*, Daniel M. Munhoz*, Milton Conde*, Geovany A. Borges*

* *Universidade de Brasília*
Campus Universitário Darci Ribeiro
Brasília, DF, Brasil

Emails: sergiocruz@unb.br, llanos@unb.br, damuz@unb.br, mecondec@unb.br, geaborges@unb.br

Abstract— This work describes a hardware architecture for implementing a sequential approach of the Extended Kalman Filter (EKF) that is suitable for mobile robotics tasks, such as self-localization, mapping, and navigation problems, especially when FPGAs (*Field Programmable Gate Arrays*) are used to execute this algorithm. In order to allow the development of small robotic platforms (for instance those required in microrobotics area) specific requirements such as small size, low-power, and floating-point arithmetic capability are demanded, as well as the design of specific and suitable hardware architectures. The system has been adapted for achieving a reconfigurable platform, suitable for research tasks, and the same has been tested in a Pioneer 3AT mobile robot platform (from Mobile Robots Inc.) for evaluating its functionality by using its local sensing system. In order to compare the performance of the system, the same localization technique has been implemented in a PC, as well as using an FPGA-embedded microprocessor (the Nios II from Altera Inc.).

Keywords— FPGA, Kalman Filter, Robot Localization

Resumo— Este trabalho descreve uma arquitetura de *hardware* para a implementação de uma versão sequencial do Filtro de Kalman Estendido (EKF, do inglês *Extended Kalman Filter*). Para permitir o desenvolvimento de plataformas robóticas pequenas (por exemplo, aquelas requisitadas em robótica móvel) condições específicas tais como tamanho pequeno, consumo baixo de potência e capacidade de aritmética em ponto flutuante são exigidos, assim como projetos de arquiteturas de *hardware* específicas e adequadas. O sistema foi adaptado para ser aplicado em uma plataforma reconfigurável, apropriada para tarefas de pesquisa, e a mesma foi testada em uma plataforma robótica Pioneer 3AT (da Mobile Robots Inc.) a fim de avaliar sua funcionalidade, usando seu sistema de sensoriamento. Para comparar o desempenho do sistema, o mesmo foi implementado em um PC, assim como pela utilização de um microprocessador embarcado na FPGA (o Nios II, da Altera).

Keywords— FPGA, Filto de Kalman, Localização de Robôs

1 Introdução

Nos últimos anos, o uso de FPGA (do inglês ***F**ield **P**rogrammable **G**ate **A**rray*) vem obtendo uma atenção especial da comunidade científica para soluções de problemas computacionais envolvendo cálculos algébricos intensos, aplicados à supercomputação, processamento de imagens, visão computacional, assim como na área da mecatrônica Arias-Garcia et al. (2011). Uma prova disso é que em uma simples busca pelas palavras *FPGA* e *algorithms* no *site* do Google (realizada em setembro de 2011) quase sete milhões de resultados foram encontrados. Nesse contexto, os FPGAs têm sido usados para a realização de soluções eficientes e adequadas em diversas áreas, a maioria delas relatadas ao desenvolvimento de *sistemas embarcados*.

Sistemas computacionais embarcados representam um campo amplo de desenvolvimento, responsável por um mercado de trabalho crescente Hartenstein (2006). Esses tipos de sistemas podem ser entendidos como sistemas computacionais especializados, desenvolvidos para realizar uma tarefa específica. Comumente, são projetados em unidades de processamento com capacidade limitada (geralmente microcontroladores) e um conjunto de sensores que lhes permitem coletar dados, viabilizando a tomada de decisões e a realização de ações de forma automática Sass and Schmidt (2010).

É importante destacar que os sistemas embarcados comumente são projetados para um propósito definido e, portanto, pode-se usar em benefício a informação disponível sobre os requisitos que o sistema deve cumprir, assim como as restrições às quais está sujeito Sass and Schmidt (2010). Em geral todos os sistemas embarcados são desenhados para operar sobre restrições de portabilidade (tamanho e peso), consumo de recursos, baixa frequência do *clock*, desempenho adequado, baixo consumo de energia e dissipação de potência Gajski et al. (2009). Esta última restrição penaliza implementações com alta frequência de operação, portanto novas soluções devem ser analisadas visando cumprir os requerimentos do produto desejado. Uma resposta a este quesito é o uso de soluções de *hardware* que explorem o paralelismo intrínseco dos algoritmos a serem embarcados, permitindo assim gerar implementações com bom desempenho, mesmo operando com baixas frequências de *clock*.

O uso de FPGAs em sistemas embarcados é explicado devido ao fato de que estes últimos estão sujeito à várias restrições em projeto, tais como desempenho, área, custo, flexibilidade e consumo de potência. Nesse contexto, várias aplicações de FPGAs (como parte dos módulos embarcados) direcionadas à área da robótica surgiram nos últimos anos. Esses desenvolvimentos foram focados para acelerar a execução de algoritmos, usando o FPGA como um acelerador de *hardware*, em parte ou no todo, de alguns algoritmos específicos aplicados à robótica móvel, como por exemplo, cadeias de Markov, filtros de Kalman, métodos de Monte Carlo, algoritmo de SLAM (do inglês ***S**imultaneous **L**ocalization **A**nd **M**apping*), entre outros. Esses algoritmos probabilísticos têm sido historicamente usados em robótica móvel, a fim de tornar as tarefas de mapeamento, localização e navegação mais robusto em relação às incertezas e/ou ruído gerados pelos sensores e também às incertezas intrínsecas do ambiente Bonato (2008).

Robôs móveis são fornecidos com um grande número de tipos de sensores e plataformas de sensoriamento, proporcionando arquiteturas com informações complementares ou às vezes de aspectos redundantes. Em vários 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. Neste caso, a utilização de técnicas para administrar uma grande quantidade de informações provenientes de vários sensores, cada um com parâmetros diferentes de exatidão e precisão, são essenciais. Estas técnicas, que visam a estimação de uma grandeza física (o mensurando) e levam em conta diversas medições, são conhecidas como *Fusão Sensorial*.

Na área de robótica móvel, a fusão sensorial é amplamente utilizada, principalmente focada para resolver problemas de localização, navegação e mapeamento. A fusão de sensores é o processo de integração de dados provenientes de diferentes sensores, mesmo usando diferentes princípios físicos, para a detecção de objetos, ou a estimação de parâmetros, ou a definição de estados, que são necessários para a auto-localização, cartografia, *path computing*, planejamento e controle de trajetória, bem como sua execução. Métodos de fusão sensorial, na área de robótica, são necessários a fim de traduzir as diferentes entradas sensoriais em estimativas confiáveis, visando a obtenção de modelos de ambiente confiáveis para as tarefas de navegação.

1.1 Definição do problema e motivações

A fusão de sensores precisa lidar com o comportamento estatístico de cada sensor, que pode ser conhecido ou não. Se o comportamento estatístico for conhecido, a tarefa de fusão depende de técnicas bem desenvolvidas como a estimativa *a posteriori* e *a máxima verossimilhança*, adaptando os resultados da filtragem de Kalman, assim como outras teorias Bayesianas. Neste caso, a maioria dos usos do filtro de Kalman em fusão sensorial (por exemplo, para a navegação) são dirigidas para a construção e manutenção de um modelo de ambiente para robôs móveis e a monitorização da posição desse robô no ambiente. Para isso, as equações do filtro de Kalman são implementadas em *software* sobre uma plataforma de sistema embarcado baseado em microprocessador. A fusão de sensores também pode ser alcançada usando arquiteturas descentralizadas para aumentar a capacidade de cálculo e comunicação, assim como para melhorar a performance. Neste caso, a estimativa local dos parâmetros a partir dos dados disponíveis é feita seguida da fusão global das estimativas locais. As estimativas locais podem ser executadas por *hardwares* específicos em que as equações do filtro de Kalman foram distribuídas. A maioria das aplicações do filtro de Kalman utilizam a versão não-linear (EKF), que também inclui o cálculo Jacobiano sobre o processo Kam et al. (1997).

Na aplicação do EKF utilizando FPGAs (focado na fusão sensorial para resolver o problema de localização) trabalhos anteriores têm lidado com o armazenamento de grandes volumes de dados provenientes de diferentes sensores na memória. Este fato representa um problema grave, uma vez que os tamanhos das matrizes aumentam em função do número de sensores. Neste contexto, sabe-se bem que a complexidade computacional para o algoritmo EKF no SLAM é $O(n^2)$, onde n representa o número de características (*features*) Thrun et al. (2005). Adicionalmente, cada característica precisa ser detectada por um sistema de medição, geralmente baseado em vários sensores. O armazenamento prévio de dados dos sensores para a etapa da estimação (ou predição) leva as operações com matrizes (por exemplo, adição/subtração, multiplicação, transposta e inversa) a dimensões maiores. Esse fato, por sua vez, requer a necessidade de projetos de arquiteturas de *hardware* que levem em consideração as necessidades de largura de banda de memória para os EKFs baseados em FPGAs **?**, e isto pode representar um caso específico do problema de *memory-wall* Hartenstein (2006).

O problema das dimensões elevadas das matrizes pode ser superado usando uma derivação sequencial do algoritmo EKF, em que a operação 'inversa da matriz' pode ser reduzida (mesmo a uma inversão escalar), sendo as observações dos sensores processadas uma por vez. A redução da dimensão da matriz também tem impacto no desempenho das outras operações, como adição/subtração, multiplicação e trans-

posta. Este fato é interessante para implementações em FPGA, dadas os seguintes aspectos: (a) FPGAs têm recursos de *hardwares* limitados, que podem ser essenciais para operações em ponto flutuante, (b) requisitos de entrada e saída dos FPGAs podem ser drasticamente reduzidos utilizando a abordagem sequencial 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 dos FPGAs pode equilibrar o processamento dos dados em série, e (f) a utilização de pequenos dispositivos com um desempenho apropriado permite aplicar as soluções para a área da microrobótica.

2 O Filtro de Kalman

O Filtro de Kalman é uma das ferramentas de estimativa mais úteis disponíveis hoje em dia, proporcionando um método repetitivo de estimar o estado de um sistema dinâmico na presença de ruído. O filtro de Kalman pode produzir estimativas dos valores reais das medições e os seus valores calculados associados ao prever um valor, estimar a incerteza do valor previsto, bem como o cálculo de uma média ponderada do valor previsto e o valor medido. O maior peso é dado para o valor com o menor grau de incerteza Thrun et al. (2005).

A estimativa produzida pelo método tende a estar mais perto dos valores reais do que as medições iniciais, dado que a média ponderada tem uma incerteza de estimativa melhor do que qualquer um dos valores individuais que realizam essa mesma média Thrun et al. (2005). A Figura 1 mostra a estrutura geral do Filtro de Kalman e suas duas fases: (a) predição e (b) correção. A filtragem de Kalman tem muitas aplicações na tecnologia e as respectivas extensões e generalizações do método também foram desenvolvidas, por exemplo *estendida* (EKF) e *unscented* (UKF, do inglês ***U**nscented **K**alman **F**ilter*).

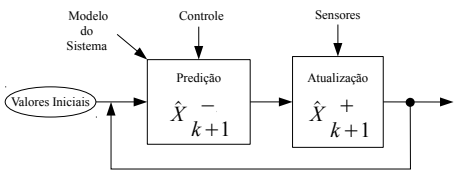

Figura 1: Estrutura Geral do Filtro de Kalman

Como mencionado anteriormente, o Filtro de Kalman é um excelente estimador de estados para sistemas lineares, porém pode ser difícil a sua aplicação na prática. Por exemplo, um robô que se move por um ambiente com velocidades rotacionais e translacionais constantes, tipicamente descreve uma trajetória circular, cujo o próximo estado não pode ser estimado pelo KF. O filtro de Kalman estendido supera essa limitação: a presunção de linearidade do sistema.

Considere o sistema não linear representado pelas equações (1) e (2):

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

$$y_{k+1} = h(x_{k+1}) + v_{k+1} \tag{2}$$

Aqui, assume-se que a probabilidade do próximo estado e as probabilidades da medição são dirigidas pelas funções não lineares $f()$ e $h()$, sendo $f()$ uma função não linear do *sistema do processo* e $h()$ uma função não linear do *sistema de medição*. Neste caso, essas funções podem ser usadas para propagar ambos o vetor de estados x_{k+1} e o vetor de saída y_{k+1}. E por fim, u_k e w_k representam o controle e o ruído do processo respectivamente, enquanto que v_{k+1} está associado ao ruído da medição.

O filtro de Kalman estendido (EKF) calcula uma aproximação do valor verdadeiro. Ele representa essa aproximação por uma Gaussiana. Em particular, a crença $bel(x_t)$ no tempo t é representada por uma média μ_t e covariância $\sum_t$ Thrun et al. (2005).

3 Metodologia

A solução em *hardware* (módulo integrado) é mostrada na Figura (2), em que a comunicação entre o processador Nios II e arquitetura em *hardware* é simplificada. Essa abordagem também reduz os recursos de *hardwares* requeridos, em termos de blocos em ponto flutuante como multiplicadores, divisores e somadores/subtratores. Isso ocorre porque é possível compartilhar o mesmo caminho de dados para a implementação em *hardware* das três equações (*ganho*, *covariâncias* e *estados*), o que resulta na redução do número de operações em ponto flutuante e, consequentemente, na área do FPGA. Vale ressaltar que, tanto a matriz Jacobiana de medição quanto a função não linear 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 CORDIC para acelerar o computo de H e $h()$.

A implementação do algoritmo de correção do EKF foi dividida em onze etapas, como mostradas nas Equações (3) a (13).

$$1^a etapa : O_{3\times2}^1 = P_k^- \cdot H^T \tag{3}$$

$$2^a etapa : O_{2\times2}^2 = H \cdot O_{3\times2}^1 \tag{4}$$

$$3^a etapa : O_{2\times2}^3 = O_{2\times2}^2 + R_k \tag{5}$$

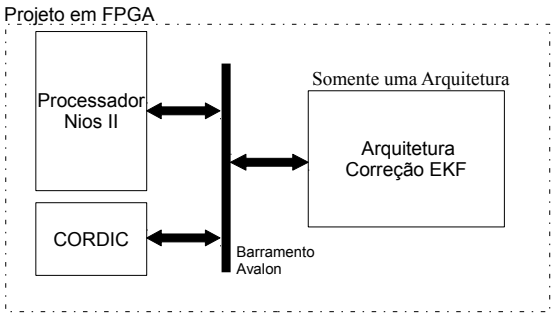

Figura 2: Implementação em FPGA com somente uma arquitetura

$$4^a etapa : O^4_{2\times2} = inv(O^3_{2\times2}) \quad (6)$$

$$5^a etapa : K_k = O^1_{3\times2} \cdot O^4_{2\times2} \quad (7)$$

$$6^a etapa : O^5_{2\times1} = Y_k - h(\hat{X}^-_k) \quad (8)$$

$$7^a etapa : O^6_{3\times1} = K_k \cdot O^5_{2\times1} \quad (9)$$

$$8^a etapa : \hat{X}^+_k = \hat{X}^-_k + O^6_{3\times1} \quad (10)$$

$$9^a etapa : O^7_{3\times3} = K_k \cdot H \quad (11)$$

$$10^a etapa : O^8_{3\times3} = O^7_{3\times3} \cdot P^-_k \quad (12)$$

$$11^a etapa : P^+_k = P^-_k - O^8_{3\times3} \quad (13)$$

A estrutura da FSM (Máquina de Estados Finitos) usada na arquitetura da equação de correção do EKF é exibida na Figura (3) e pode-se observar que ela é composta de seis estados, nomeados de *waiting*, *mult1*, *mult2*, *multadd*, *add* e *div1*. Já a Figura (4) exibe o caminho de dados para computar a FSM. Nesse caso, funções especiais no Nios II estão disponíveis para executar esses processos de leitura e escrita na matrizes. Depois de preencher o Banco das Matrizes de Entrada (BME), um comando de arranque (*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 ser lido. A arquitetura também inclui registros intermediados, que são representados pelo Banco das Matrizes Intermediária (BMI). Na arquitetura EKF, o BME é composto por P^-_k, H, R_k, Y_k, $h()$ e $\hat{X}^-_k$; o BMS é composto por K_k, $\hat{X}^-_k$ e P^+_k; e o BMI é composto por $O^1_{3\times2}$, $O^2_{2\times2}$, $O^3_{2\times2}$, $O^4_{2\times2}$, $O^5_{2\times1}$, $O^6_{3\times1}$, $O^7_{3\times3}$ e $O^8_{3\times3}$.

Pode-se observar que a Figura (5) mostra toda a arquitetura do algoritmo de correção do EKF. Nesse caso, há nove blocos de caminhos de dados, cada um deles representando o caminho de dados para calcular um elemento da matriz. Isso se deve ao fato de a maior dimensão matricial usada na computação do algoritmo de correção do EKF ser 3×3. Vale lembrar que os caminhos de dados 5 a 11 não contem blocos de divisão porque o número de elementos na inversão matricial do processo é quatro, logo, somente os caminhos de dados 1 a 4 possuem blocos de divisão.

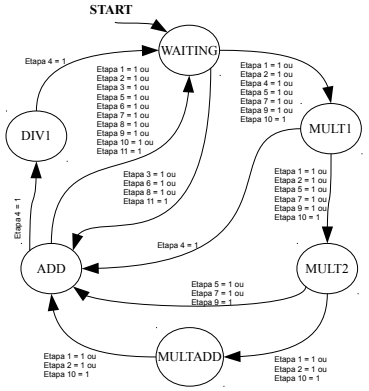

Figura 3: FSM usada na arquitetura para realizar o algoritmo de correção do EKF

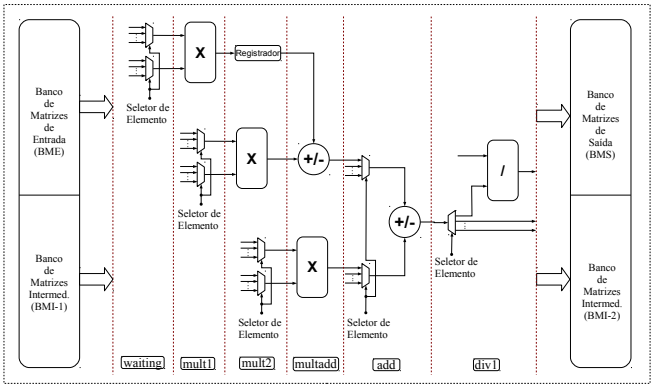

Figura 4: Escalonamento que gera o caminho de dados para computar o algoritmo de correção do EKF

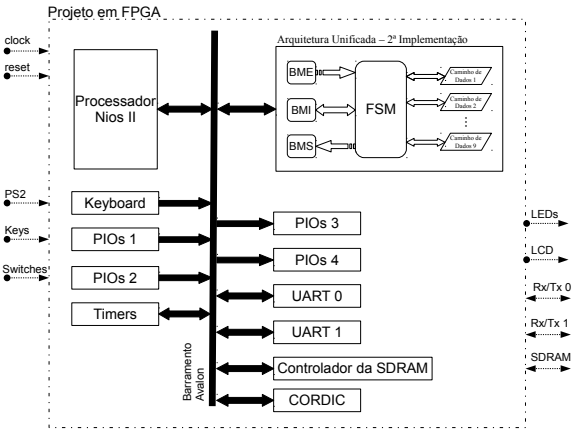

Figura 5: Projeto do FPGA com a arquitetura do algoritmo de correção do EKF

Por fim, a Tabela (1) apresenta o número de blocos em ponto flutuante requerido pela implementação do módulo integrado.

4 Resultados

Este capítulo apresenta os resultados experimentais obtidos através da primeira e segunda abor-

Tabela 1: Recursos de blocos em ponto flutuante para a implementação

Arquitetura	Somador/Subtrator	Multiplicador	Divisor
Módulo Integrado	9	9	4

dagens para a solução em *hardware* do problema de localização, levando em consideração recursos de *hardware* e desempenho. As arquiteturas propostas foram desenvolvidas em linguagem VHDL (do inglês ***VHSIC Hardware Description Language***) e compiladas usando o *software* Quartus II (Altera, 2012b). As mesmas foram validadas no *kit* de desenvolvimento DE-115, que possui um FPGA de baixo custo Cyclone IV EP4CE115F29C7N (Terasic, 2012).

4.1 Recursos de Hardware e Consumo de Potência

Os resultados de recursos de *hardware* para a arquitetura proposta são descritas na Tabela 2 bem como o consumo de potência, na qual foi estimada pelo módulo PowerPlay Power Analyzer do *software* Quartus II.

Tabela 2: Recursos de *hardware* para a arquitetura EKF

Arquit	LEs (*)	DSPs (*)	Fmáx (MHz)	Pot (mW)
EKF **	22089 (19%)	175 (33%)	49,08	201,13

* Ocupação do *hardware*

** Somente fase de correção

5 Desempenho da Arquitetura EKF

O algoritmo de correção do EKF (implementação do módulo integrado) foi executada em cinco plataformas diferentes e seus resultados foram comparados entre si. Primeiramente, o algoritmo foi rodado no processador embarcado Nios II (Altera, 2012a), posteriormente, o mesmo algoritmo foi executado tanto nos ambientes Matlab® e Dev-C++(Bloodshed, 2012) (usando um processador Intel Core 2 Duo 2.66GHz, 65W). Adicionalmente, o desempenho dos dados também foram avaliados rodando o mesmo algoritmo no PC embarcado do robô Pioneer 3AT, cuja a CPU (do inglês ***Central Processing Unit***) é um processador Intel Pentium M 1.80GHz, 21W.

O desempenho dos dados obtidos nas plataformas anteriores foram comparados com o desempenho da arquitetura implementada em *hardware* do módulo integrado. Os resultados desses desempenhos podem ser vistos na Tabela 3, cujo o tempo de execução está expresso em microssegundos (us). Nesse caso, tanto a implementação em Nios II quanto em *hardware* rodaram com um relógio de 50MHz. Vale ressaltar que o algoritmo das plataformas Dev-C++ e PC embarcado do P3-AT fez o uso da biblioteca para C/C++ chamada GMATRIX (Borges, 2005).

Tabela 3: Comparação do tempo de execução do algoritmo de correção entre as plataformas

Plataformas	Tempo de Execução (us)
Processador Nios II	577
Hardware	2,08
Nios II + *Hardware*	68
Matlab®	149
Dev-C++	22
PC P3AT	21

6 Validação da Arquitetura EKF

Um cenário foi preparado para a implementação da tarefa de validação da arquitetura do módulo integrado. Um robô, baseado na plataforma Pioneer 3AT, também foi usado, bem como os dados dos seus sensores de ultrassom. Um sensor ladar também foi usado como medidor de distância, ajudando assim, a diminuir a matriz de covariâncias. Os dados vindos desses sensores foram usados para estimar a pose do robô ($\hat{X}_k^+$). Nesse caso, a posição atual do robô é conhecida bem como as medidas do cenário. Pode-se observar na Figura 6 o ambiente de teste e a posição do robô.

Figura 6: Cenário para a validação das arquiteturas EKF

Após a realização dos testes (vale lembrar que a estimação foi *offline*), os resultados foram plotados em um gráfico juntamente com os resultados obtidos da simulação feita no programa Matlab®. A estimação de cada variável de estado está mostrada na Figura 7. Pode ser observado que ambas as estimações (feitas pelo Matlab® e arquitetura em *hardware*) são muito parecidas. Ambas convergem para a posição real do robõ na *coordenada global*, que é zero para x, y e θ.

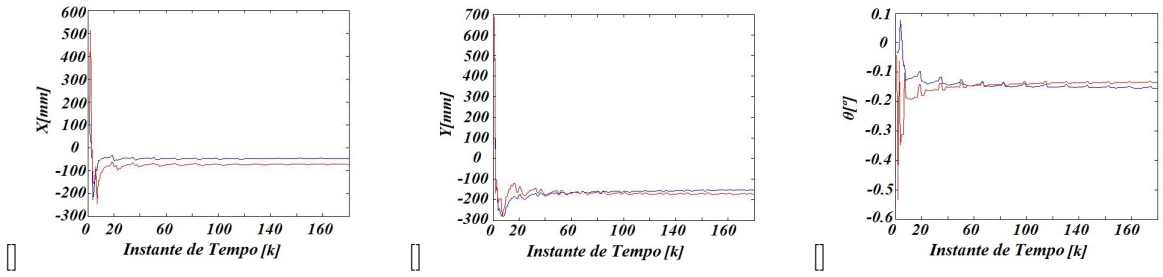

Figura 7: Estimação da pose do robô pelo Matlab® (linha azul) e arquitetura EKF (linha vermelha) para a mesma aquisição de dados em (a) X, (b) Y e (c) θ

7 Conclusões

Neste trabalho foi apresentada uma proposta de arquitetura em *hardware* (baseada em FPGA) para resolver o problema de localização de robôs móveis. Foi visto também o consumo de recursos de *hardware* e potência, bem como a performance (desempenho) dessa proposta.

Para provar o funcionamento apropriado da arquitetura, uma tarefa de validação foi realizada. Os resultados apontaram para um desempenho satisfatório do módulo EKF (referente a etapa de correção), com pouco consumo de *hardware* que é consequência direta da versão sequencial do algoritmo. A comunicação de dados entre o processador Nios II e a arquitetura EKF é a responsável direta pelo decaimento da performance como um todo.

Verificou-se que as arquiteturas trabalham tão bem quanto a versão em *software*; entretanto, se levarmos em consideração que estamos trabalhando com uma tecnologia de baixíssimo consumo de potência e pequena área de silicone, concluímos que a arquitetura proposta é perfeitamente aceita para aplicações reais.

Agradecimentos

Os autores agradecem à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo suporte financeiro a este trabalho. Como também à Altera Inc. e DHW Engenharia pela doação de *kits* de desenvolvimento e suporte técnico.

Referências

Altera (2012a). "nios ii processor reference handbook".

Altera (2012b). "quartus ii handbook".

Arias-Garcia, J., Pezzuol Jacobi, R., Llanos, C. and Ayala-Rincon, M. (2011). A suitable fpga implementation of floating-point matrix inversion based on gauss-jordan elimination, *Programmable Logic (SPL), 2011 VII Southern Conference on*, pp. 263 –268.

Bloodshed (2012). Cpp.

Bonato, V. (2008). *Proposal of an FPGA hardware architecture for SLAM using multicameras and applied to mobile robotics*, PhD thesis, Institute of Computer Science and Computational Mathematics - University of São Paulo, São Carlos, SP, Brazil.

Bonato, V., Marques, E. and Constantinides, G. (2007). A floating-point extended kalman filter implementation for autonomous mobile robots, *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pp. 576 –579.

Borges, G. A. (2005). *GMATRIX: Uma bilioteca matricial para C/C++*, ENE-UnB, Brasilia-DF.

Gajski, D., Abdi, S., Gerstlauer, A. and Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*, Springer, New York, NY, USA.

Hartenstein, R. (2006). Why we need reconfigurable computing education, *1st International Workshop on Reconfigurable Computing Education*, Karlsruhe, Germany.

Kam, M., Zhu, X. and Kalata, P. (1997). Sensor fusion for mobile robot navigation, *Proceedings of the IEEE* **85**(1): 108 – 119.

Sass, R. and Schmidt, A. (2010). *Embedded Systems Design with Platform FPGAs: Principles and Practices*, Titolo collana, Elsevier Science.

Terasic (2012). "altera de-115 development and education board".

Thrun, S., Burgard, W. and Fox, D. (2005). *Probabilistic Robotics*, The MIT Press, Cambridge, MA, USA.