

UNIVERSIDADE FEDERAL DE SÃO CARLOS

PRÓ-REITORIA DE PESQUISA

COORDENADORIA DE INICIAÇÃO CIENTÍFICA E TECNOLÓGICA

Título do Projeto de Pesquisa:

Implementação de Algoritmos Aplicados à
Robótica Móvel em Plataforma Embarcada
de Alto Desempenho

Relatório Final

Aluno: Daniel Noguchi
Projeto de Iniciação Científica
Departamento de Computação
Universidade Federal de São Carlos

Professor Orientador: Prof. Dr. Marcio Merino Fernandes

INCT-SEC, CNPQ

31 de julho de 2010

1. Introdução

O campo da robótica móvel tem evoluído consideravelmente nos últimos 20 anos. Normalmente, encontramos robôs em ambientes industriais, realizando tarefas repetitivas, entediadas e/ou perigosas. Em um ambiente industrial, os padrões de operação são bem conhecidos e assim, um robô pode ser programado seguramente para operar em tal ambiente. Porém, em um ambiente mais hostil, onde temos mudanças na iluminação, obstáculos, interação com humanos, entre outros, estes robôs não tem capacidade de atuar sem que sejam reconfigurados. Tal ambiente é o foco de atuação dos robôs móveis. Atualmente, há uma tendência destes robôs saírem do chão de fábrica para atuarem em diversas áreas como por exemplo na indústria do entretenimento (brinquedos), realizando serviços pessoais, auxiliando na automação industrial e atuando na área de segurança eletrônica.

Seguindo os avanços tecnológicos nas correntes teóricas da pesquisa robótica, a partir da década de 90 temos o paradigma da robótica probabilística, que representa a incerteza dos dados (leitura dos sensores, posição do robô, modelos, entre outros) através de ferramentas estatísticas como o cálculo probabilístico. O uso de tais ferramentas é de grande benefício na área da robótica móvel, pois nos permite trabalhar com a incerteza inerente dos novos ambientes em que esses robôs se localizam. Como é dito no livro *Probabilistic Robotics* (THRUN; BURGARD; FOX, 2006):

“Um robô que carrega a noção de sua própria incerteza e que age de acordo com ela é superior a um que não o faz”

De fato, já temos no mercado diversas empresas atuando com robôs móveis, como a In Touch Technologies, que fabrica robôs de telemedicina para tratamento remoto de paciente em hospitais, a iRobot Corporation, famosa fabricante do robô aspirador de pó Roomba, a KYVA Systems, que fabrica robôs para o gerenciamento automatizado de estoque e despacho de materiais, a MobileRobots Inc, empresa que desenvolve e produz sistemas robóticos autônomos (conhecida por fornecer robôs para pesquisa acadêmica, como o Pioneer 3-DX) e o Departamento de Defesa dos Estados Unidos, um dos maiores compradores de robôs móveis do mundo.

Todos estes produtos precisam navegar pelo ambiente dinâmico de uma casa/prédio, de um chão de fábrica ou de um ambiente externo. É neste cenário que as ferramentas probabilísticas fornecem métodos para navegação e localização de robôs autônomos. Tratamos mais exatamente de filtros probabilísticos para a estimação da pose de um robô.

Assim, neste projeto, avaliamos o uso de um algoritmo específico chamado filtro de partículas para localização de um robô móvel em um determinado ambiente, bem como a implementação deste em uma plataforma embarcada de alto desempenho. Este relatório está organizado da seguinte maneira: na seção 2 apresentamos os objetivos do projeto e sua importância. Na seção 3, descreveremos as ferramentas utilizadas, sendo estas a Teoria do Filtro de Partículas, a plataforma Player/Stage (GERKEY; VAUGHAN; HOWARD, 2003) para pesquisa em robótica e sistemas de sensores e o emulador QEMU (BELLARD, 2010). Na seção 4 apresentamos as atividades previstas e realizadas e os resultados obtidos. Por fim, a seção 5 trata da conclusão.

2. Objetivos

O projeto tem dois objetivos principais. O primeiro é avaliar uma plataforma para a implementação de sistemas embarcados para o controle de robôs móveis. A segunda é

investigar o processamento de algoritmos comumente utilizados na área de robótica móvel nesta mesma plataforma, bem como todo o desenvolvimento necessário para se implementar tal sistema de controle em um hardware embarcado.

A plataforma escolhida foi o kit de desenvolvimento BeagleBoard (BEAGLEBOARD, 2010), baseada na família de dispositivos OMAP3 da Texas Instruments (TEXAS, 2009), com capacidade comparável dos PC's atuais. O algoritmo explorado foi o filtro de partículas para localização robótica e a plataforma de desenvolvimento Player/Stage (GERKEY; VAUGHAN; HOWARD, 2003) foi utilizada como uma camada de abstração de hardware (Hardware Abstraction Layer - HAL) para a transferência de dados entre o programa desenvolvido e os sensores/atuadores do robô. O algoritmo e a plataforma Player/Stage foram selecionados em conjunto com o laboratório LCR, do ICMC-USP. Também foi utilizado o emulador de Hardware QEMU (BELLARD, 2010).

Estes objetivos foram desenvolvidos com base no sistema operacional Ubuntu GNU/Linux e foi utilizado um ambiente emulado no QEMU com uma arquitetura ARMv7 e processador Cortex-A8.

2.1 Importância

O desenvolvimento deste projeto permitiu o estudo de um hardware embarcado de baixo custo e com toda a expansibilidade das máquinas Desktop atuais, além de seu caráter silencioso. Jason Kridner, um dos principais arquitetos do projeto da BeagleBoard, afirmou que ao retirarem quase todos os periféricos on-board, provendo em seu lugar barramentos padrão de expansão como USB 2.0 e DVI-D, os desenvolvedores foram capazes de acoplar seus próprios periféricos e fazerem o que quiserem com estes. Assim, “este é o tipo de plataforma que pode ser utilizada para desenvolver soluções da área de ciência da computação que você pode colocar em qualquer lugar”, diz ele no site da plataforma beagleboard.¹

Desta forma, podemos ter as mais variadas aplicações, mas visando a área de sistemas embarcados críticos, podemos ter aplicações no setor de segurança eletrônica, monitoramento remoto, telemática veicular, automação e robótica. Podemos ainda ter a combinação dos itens descritos, com o uso da placa no controle de robôs móveis para monitoramento e segurança em ambientes internos ou um veículo autônomo em um ambiente externo, onde a placa por ser de pequeno tamanho e de baixo consumo de energia traria diversas vantagens neste tipo de situação. Assim, o estudo da integração da plataforma player/stage e o algoritmo de filtro de partículas este tipo de hardware pode fornecer uma possível implementação dos casos descritos acima. Portanto, justifica – se a investigação da implementação de tal algoritmo de robótica móvel em uma plataforma embarcada de alto desempenho.

3. Ferramentas Utilizadas

As seguintes ferramentas foram utilizadas:

- Algoritmo de filtro de partículas
- Plataforma de Software Robótico Player/Stage
- Emulador de Hardware QEMU

A seguir descreveremos cada uma delas e como foram utilizadas.

¹ <http://beagleboard.org/brief> (Visitado em: Julho de 2010)

3.1 Algoritmo Filtro de Partículas

O Filtro de Partículas, também conhecido como Métodos Sequenciais de Monte Carlo (Sequential Monte Carlo Methods – SMC) é uma técnica estatística para estimação de modelos. É uma implementação de um filtro Bayesiano, cujo principal objetivo é acompanhar uma variável de interesse enquanto esta evolui com o tempo (REKLEITIS, 2004). Este tipo de algoritmo geralmente representa funções de densidade de probabilidade (probability distribution functions – pdf's) no tempo. Para isto, constrói – se um modelo baseado em amostras do pdf. São utilizadas múltiplas cópias (as amostras, chamadas de “partículas”) da variável de interesse, e estas partículas são formadas por um estado (x) e um peso (w). Assim, as partículas são os estados possíveis da variável. No fim, a estimativa da variável de interesse é obtida realizando – se a soma ponderada de todas as partículas. O algoritmo é recursivo por natureza e pode ser subdividido nas seguintes etapas:

Inicialização: É criado um número inicial de partículas, com posições aleatórias e peso uniforme no momento $t=0$. Esta etapa ocorre somente uma vez.

Atualização: Uma série de ações (cálculos que determinarão novos pesos para as partículas) é tomada, modificando o estado das partículas. A partir destes valores, será criado um novo conjunto de pontos que irá substituir o anterior.

Reamostragem: Um novo conjunto de partículas é criado, utilizando as partículas do conjunto anterior, e as partículas com menor peso são descartadas e substituídas por novas partículas mais próximas às regiões de maior probabilidade. Se isto não fosse feito, após algumas iterações teríamos uma grande quantidade de partículas com peso quase nulo, o que não nos é de interesse (SANTOS, 2009).

Predição: As partículas são modificadas de acordo com o modelo existente, incluindo a adição de ruído aleatório nas variáveis para simular o efeito do ruído sobre a variável de interesse. Isto é feito para estimar o estado da variável no estado seguinte.

Passada a fase de inicialização, o algoritmo funciona em um loop, voltando à etapa de atualização, e as partículas vão convergindo para uma região de maior probabilidade de localização do robô.

Um algoritmo de filtro de partículas escrito em C obtido em conjunto com o laboratório LCR do ICMC-USP (LCR, 2010) foi adaptado para utilização em conjunto com o programa Player/Stage e o emulador Qemu.

3.2 Plataforma de Software Robótico Player/Stage

O Player é uma camada de abstração de hardware baseada no protocolo TCP-IP para dispositivos robóticos. O Stage é um simulador de robôs que provê um ambiente para simulação de robôs móveis e sensores, juntamente com vários objetos para estes robôs interagirem. Ambos são Parte do Projeto Player/Stage (GERKEY; VAUGHAN; HOWARD, 2003).

Quando executado em um robô, o player provê uma interface de simples manipulação para configuração dos sensores e atuadores do robô através do Protocolo TCP-IP. Dados podem ser lidos dos sensores e escritos nos atuadores, permitindo o controle do robô enquanto este está interagindo com o ambiente (SOMBY, 2008). Além disso, o player pode funcionar como um repositório de códigos, onde desenvolvedores independentes podem escrever

“drivers abstratos”, que não são nada mais que maneiras de encapsular algoritmos úteis de modo que possam ser facilmente reutilizados. Uma série destes módulos já foi desenvolvida, incluindo algoritmos de visão computacional e módulos para navegação/localização robótica.

O player e o Stage são ambos software livre, licenciados sob os termos da GNU General Public License.

Este programa foi instalado tanto no host quanto no ambiente emulado, sendo desenvolvidos arquivos de configuração do ambiente de simulação e controle do robô, e um mapa foi construído para navegação do robô.

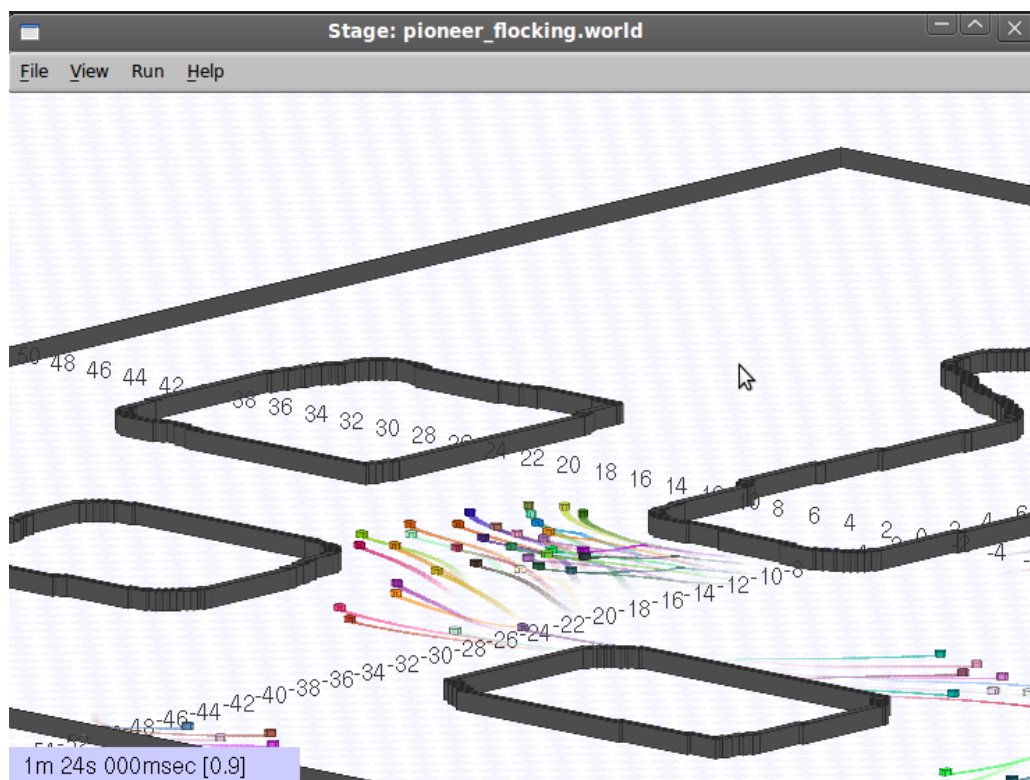


Figura 1 - Exemplo de um grupo de robôs pioneer explorando um mapa no ambiente simulado pelo Stage

3.3 Emulador de Hardware QEMU

O Qemu é um emulador de processadores e virtualizador (BELLARD, 2010). Quando executado como emulador de máquinas, ele permite a execução de sistemas operacionais e programas compilados para uma arquitetura específica (e.g uma placa com arquitetura ARM) em uma outra máquina (e.g um computador comum, com arquitetura x86).

Juntamente com o Qemu, foi utilizada a ferramenta rootstock (GRAWERT, 2010), para criação do sistema de arquivos a ser utilizado juntamente com o kernel do Linux portado para a arquitetura ARM, constituindo assim um sistema operacional Ubuntu GNU/Linux para utilização no ambiente emulado pelo Qemu.

4. Atividades Previstas

Um cronograma de atividades foi proposto para a execução das atividades. O período de duração do projeto foi de um ano. As atividades podem ser divididas em duas etapas: primeiramente temos o aprendizado das ferramentas, tecnologias e fundamentos teóricos para o desenvolvimento de projetos envolvendo sistemas embarcados e robótica e em seguida a aplicação destes conhecimentos para implementação do sistema de controle de um robô móvel. A seguir temos a descrição do cronograma proposto inicialmente:

1. Estudo da Arquitetura da Placa ARM C rtex A8.
2. Estudo de Sistemas Operacionais de Tempo Real (em especial QNX).
3. Implanta  o do S.O. de Tempo Real de refer ncia e da placa de refer ncia com eventual lote de testes iniciais de desempenho (consumo de energia, tempo se gasta, prioridades)
4. Estudo do Algoritmo de Campos Potenciais (usado para planejamento de movimentos em rob tica)
5. Estudo do Algoritmo de Filtro de Part culas (usado para localiza  o em rob tica m vel)
6. Implementa  o dos algoritmos definidos nos itens 4 e 5.
7. Estudo de viabilidade para efetuar testes de campo.
8. Implementa  o de framework para simula  o dos algoritmos
9. Efetuar testes de campo e/ou simula  o
10. An lise de desempenho quantitativa e qualitativa
11. Elabora  o de relat rio final e artigo cient fico

Estas atividades estavam inicialmente planejadas para serem realizadas de acordo com o mapeamento a seguir:

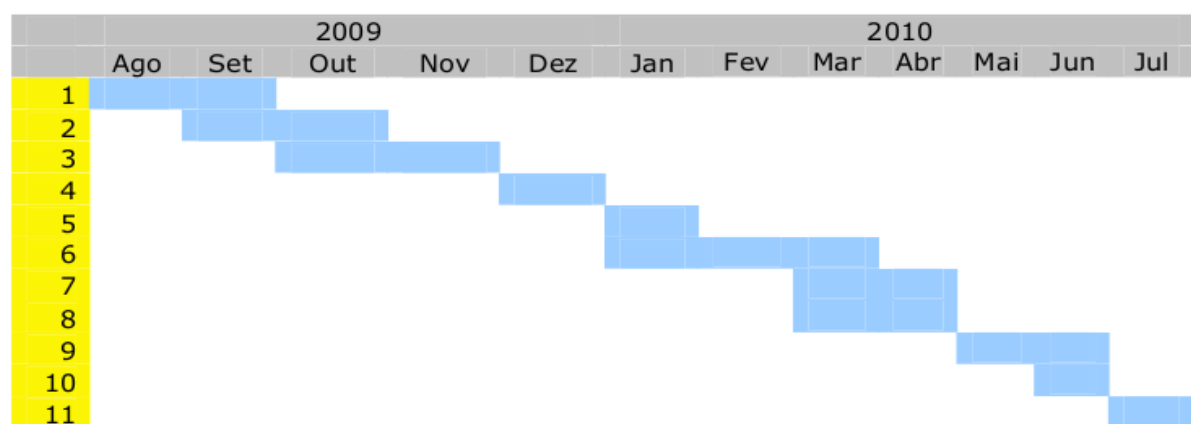


Figura 2 – Cronograma de Atividades Previsto

Assim, o ponto de partida seria o estudo da arquitetura de hardware de refer ncia, e um estudo dos Sistemas Operacionais dispon veis para Implementa  o na placa. Uma vez adquirida a placa, passar amos   etapa de instala  o do S.O mais adequado na mesma, resolvendo quaisquer dificuldades encontradas. Ap s esta fase, daria – se o in cio ao estudo dos algoritmos de rob tica selecionados para implementa  o, verificando – se a viabilidade de instala  o dos mesmos na Beagleboard (BEAGLEBOARD, 2010). Um estudo mais avan ado da placa seria necess rio nesta etapa, para verificar quest es relacionadas com   interface com sensores para aquisi  o de dados necess rios para testes de campo, e as portas e pinagens dispon veis na placa para tal interfaceamento. Caso necess rio, os testes de campo poderiam ser substituídos por ambientes de simula  o. Por fim, uma an lise funcional, quantitativa e qualitativa das atividades.

4.1 Atividades Realizadas

Inicialmente tivemos o estudo da arquitetura ARM (ARM, 2009) e a fam lia de processadores Cortex A8 (TEXAS, 2009). Nos primeiros meses, uma extensa e rica

documentação foi encontrada no site da placa beagleboard (BEAGLEBOARD, 2009) e correlatos. Deu – se o início a uma atividade de mineração de tais informações, para separar o que era relevante para o projeto e excluir informações redundantes ou conflitantes. Juntamente, tivemos o estudo do manual de referência da placa. Paralelamente, tivemos o estudo do desenvolvimento em ambiente Unix, principalmente a programação neste ambiente e também a utilização de ferramentas de cross compiling. Descobrimos que podemos construir (“build”) nossa própria “toolchain”, que é um conjunto de ferramentas de programação encapsuladas para criar um produto final, normalmente um outro programa. Uma toolchain indica que o resultado de uma ferramenta pode ser utilizado como dado de entrada de outra, assim temos que as ferramentas estão interconectadas. Assim, poderíamos gerar nossa Toolchain do zero, ou utilizar pacotes já construídos que estão disponíveis online, que foi a opção adotada. Algumas ferramentas de cross compiling disponíveis foram utilizadas, mas a mais promissora foi a da Codesourcery (CODESOURCERY, 2009), recomendada pelo site da ARM. Obtivemos bons resultados na compilação de programas para a família de processadores Cortex-A, arquitetura ARM v7.

Em seguida demos início ao uso do simulador de hardware Qemu (BELLARD, 2010), explorando o uso da ferramenta e do ambiente emulado. É importante salientar que o estudo e uso destas ferramentas envolveu o estudo de ambientes Unix e GNU/Linux, incluindo suas ferramentas e aplicações específicas, arquivos de configuração e desenvolvimento de programas em ambiente Unix. Assim, uma série de atividade paralelas foi desenvolvida, exploramos até o desenvolvimento em ambiente microsoft, porém após uma revisão das atividades juntamente ao Professor Orientador, verificou – se que o uso das ferramentas até o momento era o mais apropriado, uma vez que as aplicações estudadas (para serem utilizadas na arquitetura ARM) são em sua maioria desenvolvidas para ambientes Unix.

Neste ponto do projeto, já esperávamos que a placa estivesse a caminho, porém por problemas na importação da mesma, o cronograma de atividades teve de ser alterado, fazendo com que as atividade de estudo se prolongassem um pouco mais. Assim, fizemos também uma investigação da arquitetura, datapath e conjunto de instruções ARM.

Para a próxima etapa, tivemos uma primeira reunião com o grupo do ICMC-USP sobre os requisitos para testar os algoritmos. Foi realizada uma reunião com o Prof. Vanderlei Bonato, do ICMC USP. Na reunião, foram cobertos os principais algoritmos utilizados na navegação e localização de robôs, bem como as ferramentas utilizadas para o uso destes algoritmos, como a ferramenta Player/Stage (GERKEY; VAUGHAN; HOWARD, 2003). Foram esclarecidos alguns pontos sobre os algoritmos de Filtro de Partículas e Campos Potenciais, bem como a utilização destes na placa beagleboard. Ficou claro que será necessário o uso de um sistema operacional que sustente a ferramenta Player/Stage e será através dela que os algoritmos serão utilizados. Passamos então ao estudo desta ferramenta, para utilização da mesma. Como os próprios manuais indicaram, sua instalação não é algo trivial. Seguimos os passos indicados para instalação em ambiente Linux, usando particularmente o sistema operacional Ubuntu GNU/Linux.

Seguindo os manuais disponíveis no site do projeto², conseguimos instalar a ferramenta. Demos continuidade ao seu estudo, vendo os principais tipos de arquivos envolvidos no sistema (.cfg, .world e .inc), como construir um ambiente (mapa) para interagir com o robô, bem como a sintaxe básica para a construção deste ambiente.

O processo seguiu com a definição da distribuição Linux que seria embarcada na placa Beagleboard. Através de Pesquisa Realizada em projetos envolvendo a beagleboard, encontramos um grupo que trabalhava na construção de robôs utilizando a placa BeagleBoard. Nele, encontramos a descrição de alguns sistemas operacionais mais utilizados, que são: Angstrom, Windows Embedded, Google Android, Maemo e Ubuntu. Nos interessamos pelas Distribuições Ubuntu - por ser bem conhecida por nós - e pela Angstrom, por estar

2 <http://playerstage.sourceforge.net/> (Visitado em: Novembro de 2009)

sendo amplamente utilizada pela comunidade beagleboard. Como a plataforma Player/Stage requer uma série de pacotes para funcionar corretamente, escolhemos a distribuição Ubuntu GNU/Linux.

Como neste ponto ainda não estávamos de posse da placa, voltamos nossa atenção para o desenvolvimento do trabalho (implementação dos algoritmos de controle robótico em uma arquitetura ARM) todo em um ambiente emulado. Para isso, buscamos alternativas de simular o hardware da beagleboard no Qemu, e acabamos por encontrar um port específico do Qemu para a beagleboard, porém após a execução deste, vimos que este não tinha suporte suficiente para as nossas necessidades (emulação do Ubuntu e conexão com a rede no ambiente emulado). Assim, utilizamos o qemu-system-arm, que nos permite emular uma arquitetura ARM, especificando a imagem do Ubuntu compatível com a arquitetura ARM-v7 e processador cortex-A8 (sendo estes os encontrados na beagleboard). Um estudo de tópicos relacionados com o port do Ubuntu para a beagleboard também foi realizado, este envolvendo bootloaders, imagens de kernel, busybox e periféricos.

Abordamos então o desenvolvimento da seguinte forma: Primeiramente seria desenvolvido um ambiente virtual onde um robô iria navegar e ser controlado por um programa cliente através de uma conexão TCP com o Player. Este programa receberia dados do robô (posição e leitura dos sensores) e os armazenaria em dois arquivos separados, laser.dat e position.dat. Estes arquivos então alimentariam o algoritmo do filtro de partículas que foi obtido em conjunto com o laboratório LCR do ICMC-USP (LCR, 2010). Este algoritmo geraria uma série de partículas, que seriam plotadas no mapa original em que o robô navegou para verificação do funcionamento do mesmo. Após estas etapas, este desenvolvimento seria passado para o ambiente virtual no Qemu.

Continuamos os testes do Player/Stage, executando programas de exemplo que vieram junto com o arquivo do programa, bem como algoritmos simples compilados utilizando as API's fornecidas pelo player para interação com a ferramenta. O programa foi executado com alguns erros menores (na hora de se terminar o programa que estava interagindo com o player). O estudo dos arquivos de configuração (.cfg), simulação de ambiente (.world) e inclusão de sensores e outros (.inc) para criação de um ambiente simulado foi bem sucedido. Foram comparadas implementações dos arquivos de configuração e de simulação de ambiente das versões antigas do player (que possuem documentação) com as atuais, e foram encontradas diferenças significantes. Compilamos diversos exemplos de programas - que atuam como clientes de uma conexão com o servidor no player - que vem com a ferramenta. Criamos um programa que utiliza as API's do player para se conectar com o servidor e ler dados dele. Fomos capazes de criar nossos próprios arquivos aproveitando os exemplos já existentes, carregando nosso próprio mapa. Executamos também o exemplo "laserobstacleavoid" que controla o robô no servidor, e a tomando este como base criamos nosso próprio programa cliente para controle do robô.

Por questões de implementação do algoritmo do filtro de partículas, acabamos por utilizar o mesmo mapa que foi disponibilizado pelo laboratório LCR (LCR, 2010), que estava em um arquivo .map, gerado através de um algoritmo chamado Occupancy Grid Mapping, que designa para cada coordenada x-y um valor binário que especifica se aquela coordenada está ou não ocupada por um objeto (THRUN; BURGARD; FOX, 2006).

Utilizando o seguinte algoritmo em python elaborado por João Victor Duarte Martins licenciado sob os termos da GNU General Public License e intitulado map2png.py, fomos capazes de mapear o arquivo rth.map para uma imagem png. Esta imagem foi utilizada pelo programa Player/Stage para como o mapa no qual o robô iria navegar. O algoritmo e a imagem são descritos a seguir:


```
#!/usr/bin/python
# -*- coding: utf-8 -*-

#####
# Copyright (C) 2010 João Victor Duarte Martins <jvictor@sdf.lonestar.org> #
# #
# This program is free software; you can redistribute it and/or modify #
# it under the terms of the GNU General Public License Version 2 as #
# published by the Free Software Foundation. #
# #
# This program is distributed in the hope that it will be useful, #
# but WITHOUT ANY WARRANTY; without even the implied warranty of #
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the #
# GNU General Public License for more details. #
# #
# You should have received a copy of the GNU General Public License #
# along with this program; if not, write to the Free Software #
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. #
#####

from PIL import Image
import sys

# Parse argument line: first arg is the path to the map file
if len(sys.argv) < 2:
    print "Usage: %s MAPFILE" % sys.argv[0]
    sys.exit(-1)

# Parse the name of the map file without .map extension
name = sys.argv[1].split('.')[0]

mf = file(sys.argv[1], 'r')

mapd = tuple([ int(d.rstrip('\n')) for d in mf.readline().split()[1:] ])
img = Image.new('RGB', mapd, 0x23)

pallette = { '1': 0, '-1': 150, '0': 255 }

data = []
for l in mf:
    l = l.rstrip('\n')
    data.append((pallette[l], pallette[l], pallette[l]))

img.putdata(data)
img.save('%s.png' % name, 'PNG')
```

Algoritmo 1 - Map2png.py: Algoritmo para gerar uma imagem a partir de um occupancy grid map armazenado em um arquivo.

Após a execução do programa, obtivemos a seguinte figura:



Figura 3 - Mapa rth.png gerado a partir do arquivo rth.map

Tendo o mapa gerado, os arquivos de configuração criados para o controle do robô e o

programa cliente laserobstacleavoid adaptado, fomos capazes de simulá-lo navegando neste ambiente. A figura a seguir demonstra o resultado obtido:

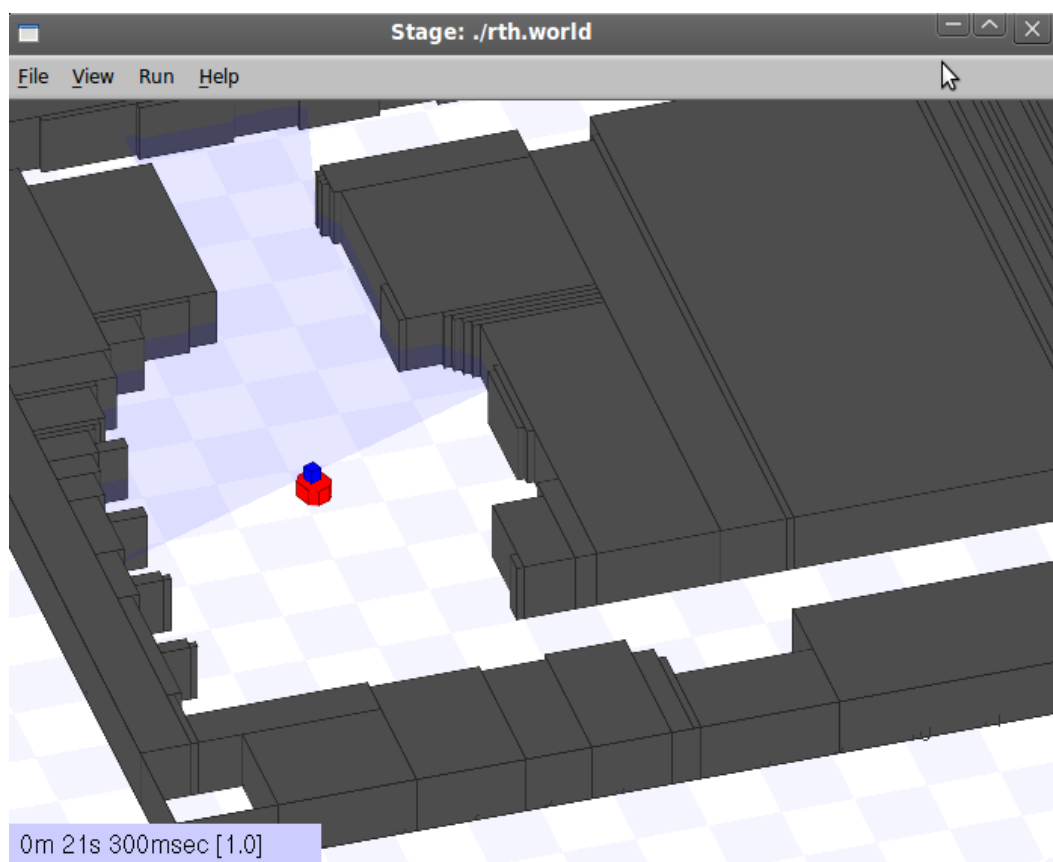


Figura 4 - Programa Player/Stage sendo executado com o mapa rth.png e sob o comando do programa cliente laserobstacleavoid.

Em seguida, de posse dos arquivos gerados pelo programa cliente, um com as coordenadas do robô e o outro com as leituras do laser, alimentamos o algoritmo do filtro de partículas com estes, sendo que o algoritmo foi adaptado para gerar arquivos com as partículas geradas a cada 10 iterações do algoritmo. Uma verificação do último arquivo gerado com as partículas mostrou que o algoritmo estava indicando a posição correta final do robô.

O passo seguinte seria executar todos estes processos no ambiente simulado pelo Qemu. Para isto, precisamos criar uma imagem do Sistema Operacional Ubuntu para ser utilizada no ambiente emulado. Utilizamos então a ferramenta rootstock (GRAWERT, 2010) para criação desta. Em seguida, precisaríamos instalar o player/stage no ambiente emulado, e para tal, necessitaríamos de uma série de pacotes que não vem instalados inicialmente quando criamos uma imagem mínima do Ubuntu. Assim, seria necessário acesso a rede no ambiente emulado. Configurando adequadamente as opções do Qemu (BELLARD, 2010) na linha de comando, foi possível redirecionar a porta 80 do host para o Qemu, permitindo o acesso a internet e consequentemente, aos repositórios de pacotes.

Por questões de capacidade de memória e processamento do host (pois todo o desenvolvimento foi realizado em um netbook), não foi possível carregar uma interface gráfica no ambiente emulado, uma vez que esta consome muitos recursos, se contarmos o fato que tudo isto seria emulado. Isto impossibilitou o uso completo do Player/Stage (GERKEY; VAUGHAN; HOWARD, 2003), que necessita da interface gráfica para simular o robô.

Portanto, somente a linha de comando estava acessível. Mesmo com esta restrição, ainda foi possível instalar o Player/Stage no ambiente emulado, pois a compilação do programa cliente depende de certas bibliotecas do Player. Assim, foi adotada a seguinte estratégia: Executaríamos o Player/Stage no host, fazendo o robô navegar no ambiente simulado, e na máquina virtual executaríamos o programa cliente, fazendo com o host uma conexão TCP, controlando o robô. O seguinte esquema ilustra a saída adotada:

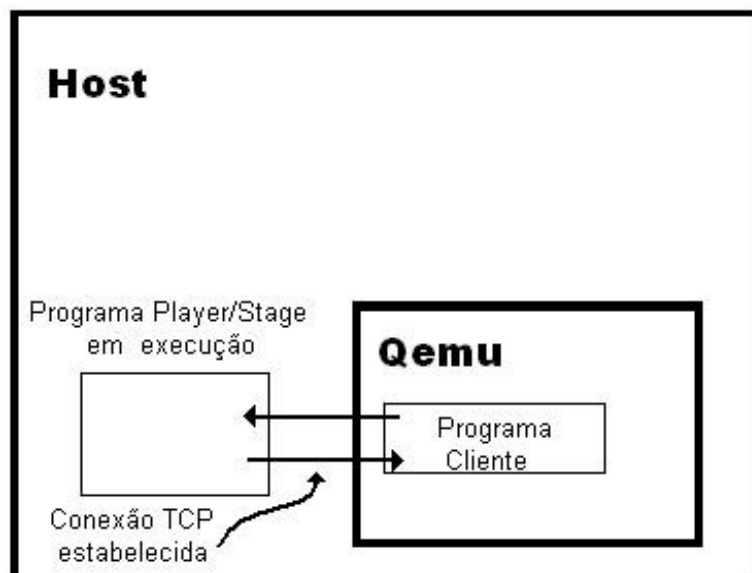
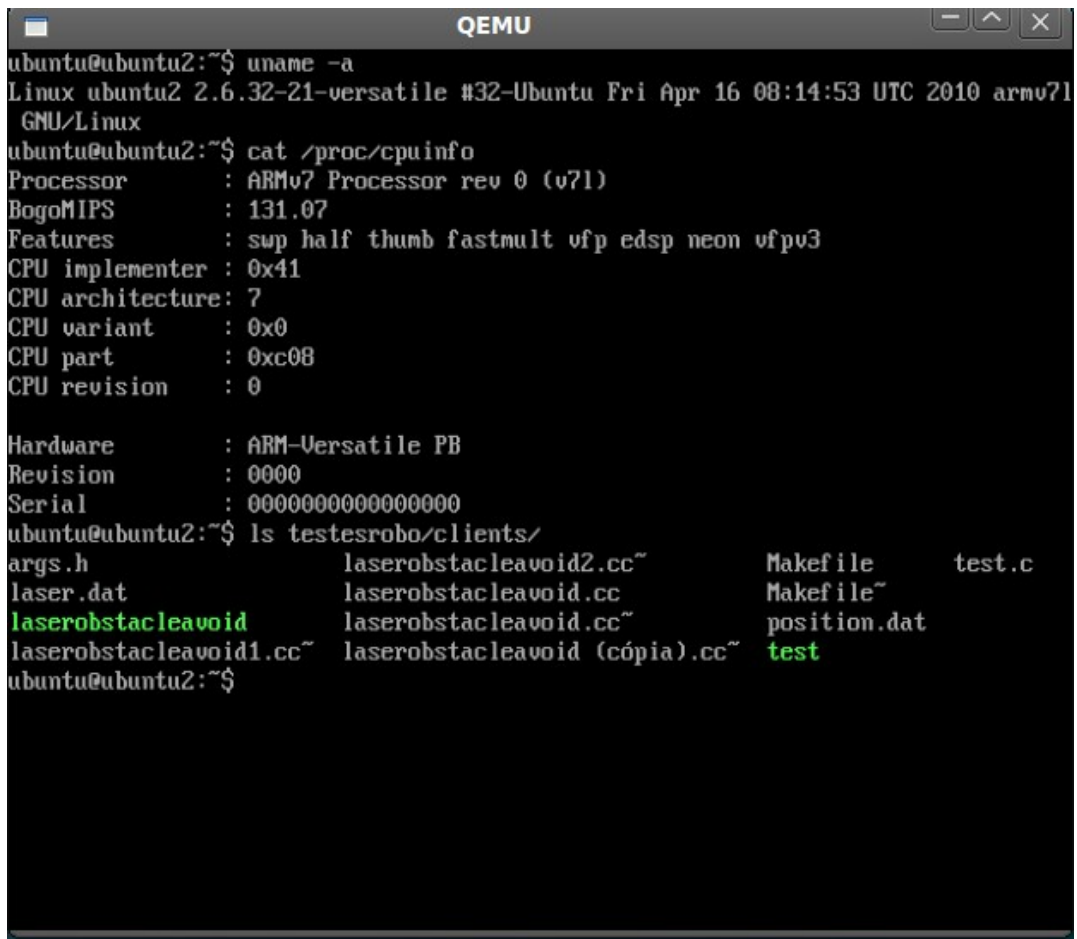


Figura 5 - Esquema da Funcionamento do controle do robô; O programa cliente executado no Qemu controla o robô no host através de uma conexão TCP.

Assim, conseguimos inicializar o Ubuntu no ambiente emulado, fazer o download dos pacotes e passar os arquivos criados até o momento no host para o sistema de arquivos da imagem gerada. Passamos então para a etapa de compilação dos programas. Nesta etapa, haviam duas opções: Poderíamos realizar o cross compiling de toda a ferramenta e do programa cliente, ou compilar tudo dentro do ambiente emulado. Como estávamos utilizando a emulação de um sistema completo, e obtivemos o acesso a rede neste ambiente, foi mais prático realizar a compilação dentro do ambiente emulado, mesmo que este fosse um pouco lento. O conjunto de ferramentas disponíveis dentro do ambiente (e as que fossem necessárias poderiam ser facilmente adquiridas dos repositórios) como o gcc, g++ e outras possibilitou esta abordagem. Porém, perto da metade da compilação do player, encontramos um erro da ferramenta “make”, indicando que uma função não estava definida no escopo. O problema apresentado se relacionava com a especificidade da arquitetura, pois a função em questão fazia parte de uma série de funções que permitia a realização de Input e Output de baixo nível em portas na arquitetura PC (x86 e 64 bits). Em vista deste empasse, procuramos nos repositórios por pacotes da ferramenta Player/Stage e encontramos um na versão 2.0.4, que difere da instalada no host, que era a 3.0.1. Porém, com pequenas modificações no programa cliente, conseguimos utilizar este pacote. Ao realizar o controle do robô com o programa compilado fazendo uso desta versão do Player/Stage, tivemos alguns erros de transmissão dos dados, devido as mudanças na API da ferramenta, da versão 2.0.4 para a 3.0.1, porém a conexão TCP foi bem sucedida. A figura abaixo mostra o Sistema Operacional em execução no ambiente emulado, e o programa compilado no ambiente virtual em destaque:



```
ubuntu@ubuntu2:~$ uname -a
Linux ubuntu2 2.6.32-21-versatile #32-Ubuntu Fri Apr 16 08:14:53 UTC 2010 armv7l
GNU/Linux
ubuntu@ubuntu2:~$ cat /proc/cpuinfo
Processor       : ARMv7 Processor rev 0 (v7l)
BogoMIPS       : 131.07
Features        : swp half thumb fastmult vfp edsp neon vfpv3
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc08
CPU revision    : 0

Hardware       : ARM-Versatile PB
Revision       : 0000
Serial         : 0000000000000000
ubuntu@ubuntu2:~$ ls testesrobo/clients/
args.h          laserobstacleavoid2.cc~  Makefile        test.c
laser.dat       laserobstacleavoid.cc~  Makefile~
laserobstacleavoid  laserobstacleavoid.cc~  position.dat
laserobstacleavoid1.cc~ laserobstacleavoid (cópia).cc~ test
```

Figura 6 - Ambiente virtual em execução no QEMU. É possível ver a identificação do processador ARMv7.

Por fim, era necessário visualizar o processo de geração de partículas e a convergência das mesmas a cada iteração do algoritmo. Para isso, utilizamos os arquivos gerados pelo filtro de partículas e plotamos cada coordenada de cada partícula de volta ao mapa original.

Seguindo a mesma lógica do programa `map2png.py`, desenvolvemos o seguinte algoritmo para plotar as partículas obtidas no mapa utilizado, intitulado `plotpart.py`, para verificação do funcionamento correto do algoritmo e do controle do robô. O algoritmo e o resultado (plotando as partículas iniciais) estão descritos a seguir:

```

from PIL import Image
import sys

if (len(sys.argv) < 3):
    print "erro"

data_filename = sys.argv[1]
img_filename = sys.argv[2]

im = Image.open(img_filename)
pix = im.load()
pp = file(data_filename, 'r')

for l in pp:
    l = l.rstrip('\n')
    part = l.split()
    posX = int(10*float(part[0]))
    posY = int(10*float(part[1]))
    posY = 207 - posY
    print "marcando %d %d" %(posX, posY)
    pix[posX, posY] = 0x0000FF

im.save(data_filename + ".png")
pp.close()

```

Algoritmo 2 – plotpart.py; Algoritmo para plotar as partículas na imagem do mapa em que o robô navegou.

Executando este programa, obtivemos a seguinte imagem:

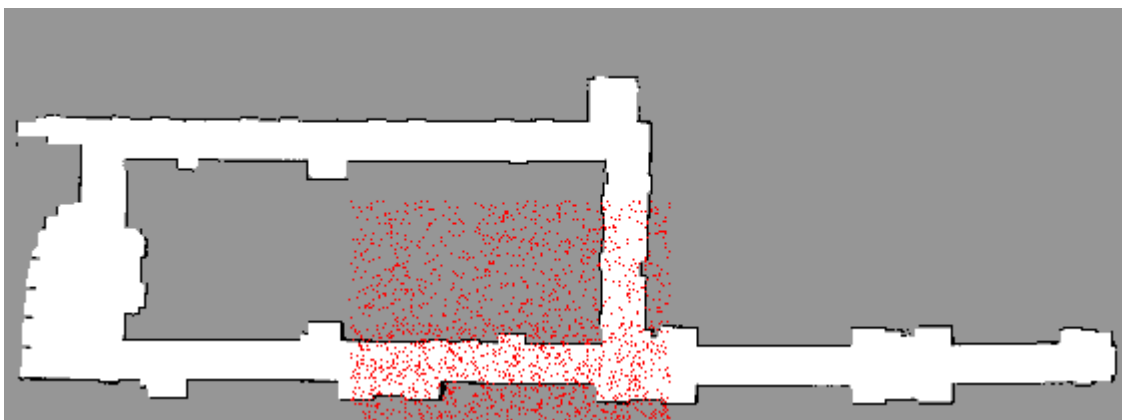


Figura 7 - Mapa rth.png com 2000 partículas plotadas

4.2 Resultados Obtidos

A partir desses dados obtidos, uma série de 20 imagens geradas pelo algoritmo foram plotadas no mapa em que o robô navegou, onde estas partículas indicaram corretamente a posição final do robô. Além disso, conseguimos com sucesso realizar a conexão TCP entre o ambiente emulado e o host (no caso o computador no qual o Qemu estava sendo executado). A sequência de imagens a seguir descreve a evolução do algoritmo que foi obtida:



Figura 8 - 30 iterações do algoritmo

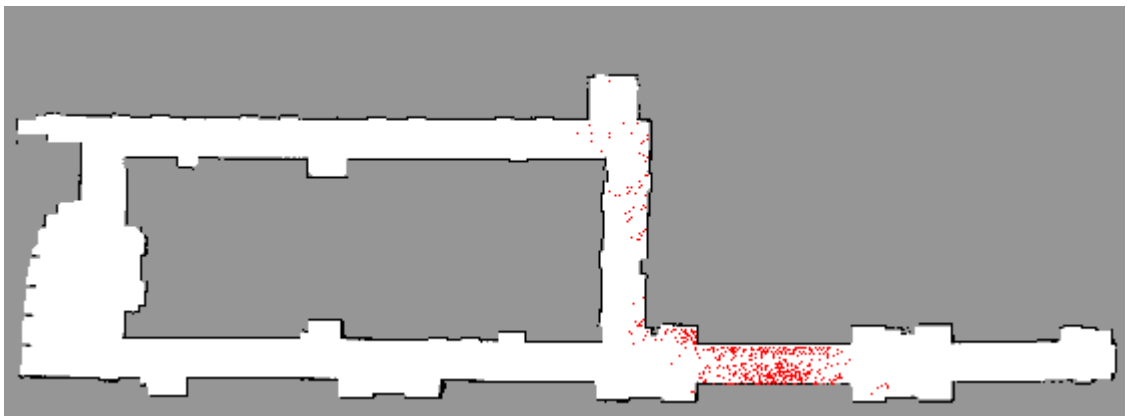


Figura 9 - 60 iterações do algoritmo



Figura 10 - 90 iterações do algoritmo



Figura 11 -150 iterações do algoritmo



Figura 12 - 200 iterações do algoritmo

Como podemos ver, o algoritmo foi capaz de localizar o robô corretamente. O porte das atividades para o ambiente emulado e a execução do programa cliente também foi bem sucedido, indicando um resultado positivo na utilização de um sistema embarcado para tal sistema de controle robótico.

4.3 Dificuldades Encontradas

Inicialmente, a plethora de informações encontradas sobre o desenvolvimento de sistemas embarcados e desenvolvimento para sistemas baseados em arquitetura ARM requereu a filtragem destas informações, que muitas vezes eram desconexas, incorretas ou bem específicas. Além disso, a curva de aprendizado de algumas ferramentas se mostrou alta, principalmente quando se trata de configuração de scripts e arquivos. Outra grande dificuldade foi a instalação da ferramenta Player/Stage, que necessitava de uma série de pacotes e configurações, que muitas vezes se não executadas corretamente causavam conflitos. Além disso, uma parte da documentação disponível estava desatualizada.

Por fim, o maior agravante foi a dificuldade na importação da Placa Beagleboard, que atrasou de modo que as atividades não puderam ser realizadas nela, forçando o cronograma de atividades a mudar e fazer com que nos concentrássemos nas atividades de simulação. Em vista disso, não se trabalhou com o algoritmo de campos potenciais devido à falta da placa. Portar o algoritmo apenas para o ambiente emulado não acrescentaria muito em termos do que já estava sendo feito para o algoritmo de filtro de partículas.

5 Conclusões

Contudo, os resultados obtidos foram promissores. A análise funcional indicou que a

implementação de tal sistema de controle no sistema embarcado investigado é possível e o algoritmo do filtro de partículas utilizado foi capaz de localizar o robô corretamente. Tal pesquisa pode servir de base para a utilização da placa no controle de robôs móveis para monitoramento e segurança em ambientes internos ou num veículo autônomo em um ambiente externo, onde a placa por ser de pequeno tamanho e de baixo consumo de energia traria diversas vantagens neste tipo de situação. Assim, o estudo da integração da plataforma player/stage e o algoritmo de filtro de partículas a este tipo de hardware pode fornecer uma implementação acessível, de baixo custo e baixo consumo de energia para os casos descritos acima.

6 Agradecimentos

Esta pesquisa foi desenvolvida com apoio financeiro dos órgãos CNPQ e INCT-SEC para iniciação científica. Também agradecemos a colaboração do Prof. Vanderlei Bonato, do ICMC-USP, que disponibilizou seu tempo para esclarecimentos do Algoritmo Filtro de Partículas e do programa Player/Stage. Agradeço também ao colega de departamento João Victor Duarte Martins, que me auxiliou no que concerne o uso das ferramentas do ambiente Unix e Programação em Python. Tais colaborações fizeram possível a realização bem sucedida desta pesquisa.

7. Referências Bibliográficas

ARM HOLDINGS. Processors [Online] [Visitado em: Outubro de 2009] <http://www.arm.com/products/processors/index.php> .

BEAGLEBOARD. A low cost, fan-less single-board computer [Online] [Visitado em: Julho de 2010] <http://beagleboard.org> .

BELLARD, Fabrice. Qemu. An Open Source Machine Emulator and Virtualizer [Online] [Visitado em: Julho de 200.] http://wiki.qemu.org/Main_Page.

CODESOURCERY. GNU Toolchain for ARM Processors [Online][Visitado em: Outubro de 2009] www.codesourcery.com/sgpp/lite/arm .

GERKEY, B. P., Vaughan, R. T. e Howard, A. 2003. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. IEEE International Conference on Advanced Robotics - ICAR. July, 2003.

GRAWERT, Oliver. Rootstock. Tool to build an Ubuntu root filesystem for a target device from scratch [Online] [Visitado em: Julho de 2010] <https://wiki.ubuntu.com/ARM/RootfsFromScratch> .

LCR. SSC 711 – Co-Projeto de Hardware/Software para Sistemas Embarcados [Online] [Visitado em: Fevereiro de 2010] <http://www.icmc.usp.br/~lcr/en/courses/hwswcodesign/> .

REKLEITIS, I. 2004. “A Particle Filter Tutorial for Mobile Robot Localization”. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, Montreal, Québec, Canada, 2004.

SANTOS, Francisco C. A. 2009. Introdução ao Filtro de Partículas [Online] [Visitado em: Julho de 2010] http://www.lti.pcs.usp.br/pcs5019/TermPapers/PCS5019_TermPaper_FranciscoSantos.pdf .

SOMBY, Michael. Linux for Devices. Updated Review of Robotics Software Platforms [Online] [Visitado em: Julho de 2010] <http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Updated-review-of-robotics-software-platforms/> .

TEXAS INSTRUMENTS INCORPORATED. Embedded Processors [Online] [Visitado em: Outubro de 2009] http://www.ti.com/home_p_processors .

THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. Probabilistic robotics. Cambridge, Mass. : MIT Press, c2006.

8 Parecer do Orientador

De maneira geral, o trabalho foi bem conduzido, atingindo a maior parte dos objetivos previstos. O aluno demonstrou bastante interesse e dedicação ao longo do projeto, que envolveu a assimilação de novos conceitos e habilidades práticas. Conforme já mencionado neste relatório, as principais dificuldades encontradas tem como causa o atraso na importação da placa BeagleBoard. Apesar de não inviabilizar o projeto, isso exigiu alguns ajustes no cronograma e metodologia inicialmente planejada. Mesmo assim, considero que o projeto tenha sido bastante proveitoso para a formação do aluno, tanto pela oportunidade de conduzir um projeto científico, como pelos conhecimentos adquiridos em uma área atual e relevante para a área de computação.

Prof. Dr. Marcio Merino Fernandes

9Avaliação do aluno:

Tendo em vista que o desenvolvimento deste projeto requereu amplos conhecimentos em desenvolvimento em ambiente Linux, não sendo somente o desenvolvimento de programas, mas o uso do sistema como um todo, envolvendo a manipulação de scripts, conhecimentos em cross-compiling e toolchains, modificação de características específicas de arquivos de configuração do Unix (library paths, program paths, edição de arquivos de configuração de rede, etc) para o funcionamento correto das ferramentas, ambientes virtuais, entre outros e levando em conta que meu conhecimento prévio de trabalho em ambiente Unix não passava de 6 meses de familiarização básica com este tipo de SO, creio que alcancei os objetivos esperados satisfatoriamente, tendo executado com sucesso as atividades que envolviam os tópicos descritos acima.

Um benefício direto foi a maturação nos conhecimentos da linguagem de programação C, bem como o aprendizado e utilização da linguagem python.

Como ressalva, tive certas dificuldades com a manipulação de imagens que foi necessária em uma etapa do projeto, pois algumas características intrínsecas do trabalho com imagens, como field of view, coordenadas cartesianas de uma imagem, razão pixel por metro, etc acabaram sendo difíceis de lidar. Contudo, creio ter conseguido integrar esta parte com o resto do projeto de forma satisfatória.

O maior benefício que obtive ao desenvolver este projeto foi a aquisição de conhecimentos da área de sistemas embarcados, pela qual demonstrei grande interesse. Como estudante de Engenharia de Computação, me interessei pela área de programação de baixo nível e desenvolvimento que envolva hardware. Assim, este projeto foi de grande benefício para a obtenção de conhecimento prático que muitas vezes pelo curto espaço de tempo não são vistos em grande extensão na graduação, com destaque ao desenvolvimento em sistemas unix e sistemas embarcados em geral. Outro benefício foi o trabalho em conjunto com outros pesquisadores e laboratórios de Pesquisa, como o Laboratório de Computação Reconfigurável do ICMC-USP e o Prof. Dr. Vanderlei Bonato. Esta troca de conhecimentos e trabalho em equipe teve para mim um valor inestimável na formação de um profissional capaz de interagir com e colaborar positivamente no desenvolvimento de projetos.

Outro ponto importante foi o contato com o campo da robótica móvel, que teve uma grande evolução nas últimas duas décadas. Ao estudar o sucesso que as mais variadas empresas já estão obtendo com produtos nesta área, e também ao identificar o potencial deste campo em futuras aplicações pude ver como esta área é crucial para o desenvolvimento de tecnologia de ponta para o nosso país. O investimento em pesquisa e desenvolvimento nesta área é de grande importância no que concerne o desenvolvimento de sistemas embarcados críticos, mais especificamente veículos autônomos.

Tive também a oportunidade de conhecer a área de sistemas embarcados, que necessita de profissionais altamente capacitados para poder lidar com a complexidade e requisitos estritos que o desenvolvimento de placas e hardware embarcados necessitam.

Por fim, agradeço ao Prof. Dr. Marcio Merino Fernandes pela orientação e dedicação ao longo deste projeto.

10. Destino do Bolsista

O bolsista Daniel Noguchi está no quarto ano do curso de Engenharia de Computação e concluiu a pesquisa satisfatoriamente, atingindo a maior parte dos objetivos previstos. O aluno pretende utilizar os resultados obtidos para dar continuidade ao projeto através de produções científicas.

11. Produção Técnico-Científica

Os resultados obtidos durante a pesquisa foram submetidos ao III Congresso de Iniciação em Desenvolvimento Tecnológico e Inovação (CIDTI - <http://www.cict.ufscar.br/>), a ser realizado na Universidade Federal de São Carlos, de 13 a 15 de outubro. A participação se dará na forma de apresentação de trabalho oral. Aguardando resultado da inscrição.