UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Tutorial para participação na Virtual Robots Competitions da RoboCup e implementação de um controlador para a categoria

Marcelo Ribeiro Xavier da Silva

Trabalho de conclusão de curso apresentada como parte dos requisitos para obtenção do grau Bacharel em Ciências da Computação

Florianópolis – SC

2008/1

Marcelo Ribeiro Xavier da Silva

Tutorial para participação na Virtual Robots Competitions da RoboCup e implementação de um controlador para a categoria

Trabalho de conclusão de curso apresentada como parte dos requisitos para obtenção do grau Bacharel em Ciências da Computação

Orientador: Professor Doutor Mauro Roisenberg

Banca Examinadora

_____

Professor Doutor Jovelino Falqueto

_____

Professor Doutor Ricardo Azambuja Silveira

_____

Professora Doutora Silvia Modesto Nassar

Sumário

Lista de abreviaturas e siglas

RoboCup	(Junta internacional em prol da robótica inteligente),	p. 9
USARSim	(Urban Search And Rescue Simulation),	p. 10
VR	(Virtual Robots),	p. 10
ITA	(Instituto Tecnológico da Aeronáutica),	p. 10
UFSC	(Universidade Federal de Santa Catarina),	p. 10
SLAM	(Simultaneos Localization And Mapping),	p. 10
IA	(Inteligência Artificial),	p. 11
EKF	(Extended Kalman Filter),	p. 39

Lista de Figuras

Resumo

A área de Inteligência Artificial ainda está em estado de evolução, em razão disso, foi criada uma junta internacional, conhecida como RoboCup, que tem como principal objetivo incentivar este crescimento. Isto é feito através da promoção de inúmeras competições relativas à robótica inteligente. Dentre essas competições existe a Virtual Robots Competition, porém, esta ainda não possui grande representatividade no Brasil. Com o intuito de se divulgar esta competição aqui, várias universidades se propuseram a criar uma ferramenta nacional para ser usada na Virtual Robots Competition, além de servir como instrumento didático para o aprendizado de IA. Parte desta ferramenta trata da localização e mapeamento simultâneos, ou SLAM. Este processo é feito por meio de agentes inseridos em diversos ambientes. E é o aprendizado prático/teórico do SLAM o tema abordado por este trabalho, sendo um dos objetos criar algoritmos para serem usados na nova ferramenta de controle, alcunhada de BrasilVR. Além disso este trabalho servirá de facilitador para ingresso na competição, através de um tutorial de instalação e configuração das ferramentas necessárias.

1 Introdução

A inteligência artificial é a área de pesquisa das ciências da computação dedicada a estudar os problemas "complexos", aqueles cuja computação convencional não consegue ou tem grande dificuldade em resolver.

A expressão robô vem do tcheco robota, e significa trabalhador, foi criada por Karel Capek, em 1917. Ou seja, um robô é uma máquina que exerce um trabalho específico, um trabalho para o qual fora justamente criada. Com a injeção de Inteligência Artificial, essas máquinas estão exercendo cada vez melhor suas funções e estão se tornando cada vez mais abrangentes, executando cada vez mais tarefas.

Com a finalidade de promover a Inteligência Artificial e a Robótica Inteligente foi criada uma junta Internacional, chamada de . Para cumprir essa função de disseminar e fazer evoluir a cultura da robótica, a RoboCup promove várias competições que objetivam a integração de inúmeras tecnologias, algoritmos e métodos. Dentre estas competições, existe a Virtual Robots Competition(VIRTUAL..., 2006) que, como induz o nome, é uma simulação de robôs.

A Virtual Robots Competition introduz um problema de busca e reconhecimento de padrões. Dentro desta temática, é criada uma equipe robótica que deve procurar e pré-diagnosticar feridos em um mapa que simula um local pós castástrofe. Esta competição usa o Urban Search And Rescue Simulation (USARSim)(USARSIM,), um simulador de alta fidelidade que usa o engine do jogo UnrealTournament (comprado separadamente). Dentro desse simulador são gerados os mapas, as vítimas,os obstáculos, os robôs, além de toda a interação entre esses objetos. Neste ambiente, os sensores dos robôs também são simulados de forma semelhante aos reais. Assim cabe aos usuários, criar a "inteligência"desta equipe, de tal forma que encontrem da maneira mais rápida e eficaz os feridos, para que possam aplicar então os algoritmos de reconhecimento de padrão. Todas as regras(RULES..., 2008), incluindo sensores permitidos, podem ser encontradas no site da competição.

Para fazer a inserção dos robôs no simulador existem ferramentas chamadas de controladores. Porém estas ferramentas não servem apenas para inserir os robôs no simulador. Como

o nome leva a pensar, elas servem para fazer o controle de toda a informação gerada e/ou lida pela equipe de robôs, ou seja, servem para fazer toda interação dos usuários com os sensores dos robôs. No manual de instalação do é citado o Mobility Open Architecture Simulation and Tools (MOAST)(MOAST,) que é um dos controladores suportados pelo simulador. O manual do USARSim cita ainda dois outros controladores, porém, a utilização destes controladores não é intuitiva, além do fato de ser necessário ter profundo conhecimento sobre como fazer a entrada dos dados e a leitura de informações devolvidas por eles, tornando-os não muito didáticos.

Considerando todos os fatores que dificultam a utilização dos controladores para a , pesquisadores do desenvolveram um controlador em C++, que deveria, além de servir para a competição, suprir a necessidade de uma ferramenta eficiente para ensino de robótica e inteligência artificial. Para que esta ferramenta fosse eficiente ela deveria ser tão intuitiva que não prenderia o aluno no funcionamento dela em si, mas sim no seu objetivo ao utilizá-la. O controlador criado pelo ITA já pode ser usado na competição, porém ainda não está completo para ser utilizado como ferramenta didática. Portanto, em parceria com eles, e com outras instituições, pretendemos finalizar este processo criativo.

Coube aos alunos da a tarefa de fazer a parte de localização e mapeamento simultâneos (SLAM - Simultaneous Localization And Mapping). O objetivo do algoritmo de é encontrar um mapa do local por onde os robôs estão se movimentando, sem perder as informações da localização de cada robô envolvido no processo de construção.

Como existem pouco tutoriais e manuais de como participar na categoria, faz parte da solução dos problemas criar um tutorial em português que possa vir ajudar no primeiro contato com as ferramentas e problemas da Virtual Robots Competition.

1.1 Justificativa

Para participar da categoria Virtual Robots Competition há uma grande dificuldade em saber o que precisa ser instalado, o que precisa ser aprendido e como fazê-lo na ordem correta. A escassez de tutoriais, principalmente em português, motivou a criação de algo deste gênero.

Existem inúmeros controladores para a Virtual Robots Competitions da RoboCup, porém nenhum deles foi considerado satisfatório por não terem em suas interfaces uma forma simples para entrada e saída de dados, tornando o uso complicado e nada intuitivo.

A idéia inicial dessa parceria entre universidades é criar um controlador que seja tão simples de ser usado que venha a ser uma ferramenta para o aprendizado de conceitos e técnicas de .

Acreditamos que ao criar um controlador simples em termos de uso e completo em termos de entrada e saída de dados, não apenas irá ajudar a aumentar o interesse das pessoas na área de Inteligência Artificial, como também irá aumentar a participação de brasileiros na Virtual Robots Competitions.

O ideal deste trabalho é ajudar a longo prazo a desenvolver a área de Inteligência Artificial, criando e/ou aumentando o interesse entre os estudantes brasileiros tanto na área quanto em competições que busquem o mesmo objetivo.

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo deste trabalho é auxiliar a criação de um controlador cuja implementação seja inovadora e cumpra as regras definidas na RoboCup para a Virtual Robots Competition. Este controlador será usado como ferramenta didática para estudo de técnicas de Inteligência Artificial, para isso deve ser bastante intuitivo e fácil de usar

É também objetivo deste trabalho, construir um passo a passo simples o bastante para que qualquer pessoa possa ingressar na Virtual Robots Competition sem encontrar grandes dificuldades, ao menos no que diz respeito à instalação e utilização de ferramentas.

1.2.2 Objetivos específicos

Para alcançar os objetivos gerais definidos para o trabalho, os seguintes objetivos específicos foram listados:

- Fazer um levantamento de informações para criar o tutorial de forma completa, abrangendo todo o processo preparatório para a participação na VR Competition
- Fazer com que o controlador seja adaptado às regras da Virtual Robots Competition
- Estudar sobre SLAM
- Adaptar o controlador, tornando-o:
  - Multiplataforma;
  - Extensível;
  - Escalável;
  - Fácil para inserção e testes de novos algoritmos.

1.3 Metodologia

O trabalho foi realizado e está organizado do seguinte modo:

No capítulo 2 é explicada a competição para o qual está tentando-se criar um controlador

No capítulo 3 é iniciado o tutorial, passando por pontos relevantes tanto da instalação de ferramentas quanto de algumas regras da competição.

No capítulo 4 é traduzido o problema que se tenta resolver com este estudo, além das explicações de relevância para o entedimento das soluções escolhidas.

No capítulo 5 é mostrado e explicado o algoritmo de SLAM que foi implementado, ressaltando a contribuição dos trabalhos estudados para que a implementação final fosse alcançada.

O trabalho termina com as conclusões ressaltando como foi se desenvolvendo o projeto.

2 *Regras e funcionamento da Virtual Robots Competitions*

Nesta seção será explicado o funcionamento da Virtual Robots Competition passando por pontos relevantes, desde testes nos quais as equipes devem passar até explicações sobre pontuação.

2.1 Escopo e Definição

A Virtual Robots Competition é uma das competições que a RoboCup(ROBOCUP, 1998) promove. A RoboCup é uma junta internacional que objetiva o desenvolvimento da Robótica Inteligente e da Inteligência Artificial. Virtual Robots Competition é basicamente um jogo onde se devem inserir robôs, aqui chamados de agentes, dentro de um ambiente simulado. Os mapas do ambiente caracterizam um local pós-catástrofe, como um prédio depois de um incêndio ou uma casa depois de um terremoto, por exemplo. A função dos agentes inseridos nesse simulador é encontrar possíveis feridos criando um mapa que possua a posição destes feridos além de representar a forma como se encontra local depois da catástrofe ter ocorrido. Quem provê o ambiente simulado é o Unified System for Automation and Robot Simulation (USARSIM,). O USARSim é um simulador de ambientes baseado no motor do jogo Unreal Tournament(UNREAL. . . ,), e pode conter tipos de objetos com características variadas, como a representação de seres vivos, objetos móveis (como outros agentes por exemplo), objetos estáticos (paredes, escadas, etc), dentre outras coisas.. Dentro deste simulador é possível interagir com os objetos através dos sensores dos agentes, ou seja, fazer leituras de distância, leitura de caminho percorrido (odometria), etc. Dentro da hierarquia da RoboCup existem as competições simuladas e não simuladas. A Virtual Robots Competition se situa dentro da RoboCupRescue Simulation league, que abriga outras competições simuladas. Um dos objetivos desta divisão é promover a troca de algoritmos entre competições de uma mesma categoria.

2.2 Configurações de Software e Hardware

Dentro da organização da competição, as equipes que participam da competição têm direito a dois conjuntos de três computadores (chamados de cluster). Como a competição se dá em vários ambientes simulados. Destes dois clusters providos pela organização, um deve ser usado pelo time durante uma simulação enquanto o outro deve ser usado para que o time se prepare para a próxima competição, para o segundo, estão disponíveis sensores de relevo (GroundTruth), para que as equipes possam preparar melhor suas estratégias. Os códigos gerados pelas equipes devem ser rodados em máquinas próprias, sendo que é fornecido um cabo para que essas máquinas sejam conectadas ao cluster. A organização do evento monta um servidor sem fio e as comunicações entre o membro da equipe que opera a máquina onde está o código e agente inserido no ambiente simulado devem usar esse serviço wireless. Na figura 2.1 está representado esse esquema de comunicação. Para acessar o vídeo (acessar visualmente o USARSim) a estação base operadora deverá usar o WCS. Dessa forma, os vídeos serão obtidos apenas quando o robô mantiver contato com o rádio.

2.3 Esquema da Competição

Para que haja uma filtragem de quem participará da competição, são aplicados testes de qualificação (ou testes elementares) e testes de cenário. Em outras palavras, as equipes terão de exibir competência e habilidades básicas como mapeamento e mobilidade dentro do USARSim.Existem três níveis de dificuldade na competição, mapas fáceis (também chamados de zonas amarelas), mapas intermediários (zonas laranja) e mapas vermelhos (zonas difíceis). As equipes devem mostrar que possuem capacidade de para operar nas zonas fáceis e intermediárias do ambiente de teste. Essas habilidades serão necessárias para ter sucesso nos cenários da competição. As áreas de maior dificuldade do ambiente de teste podem ser usadas para testar soluções avançadas e inovadoras, além de mostrarem o possível rumo que a competição tomará numa próxima edição

2.4 Testes Elementares

Os testes elementares são divididos em mobilidade e mapeamento, cada um deles será explicado melhor a seguir.

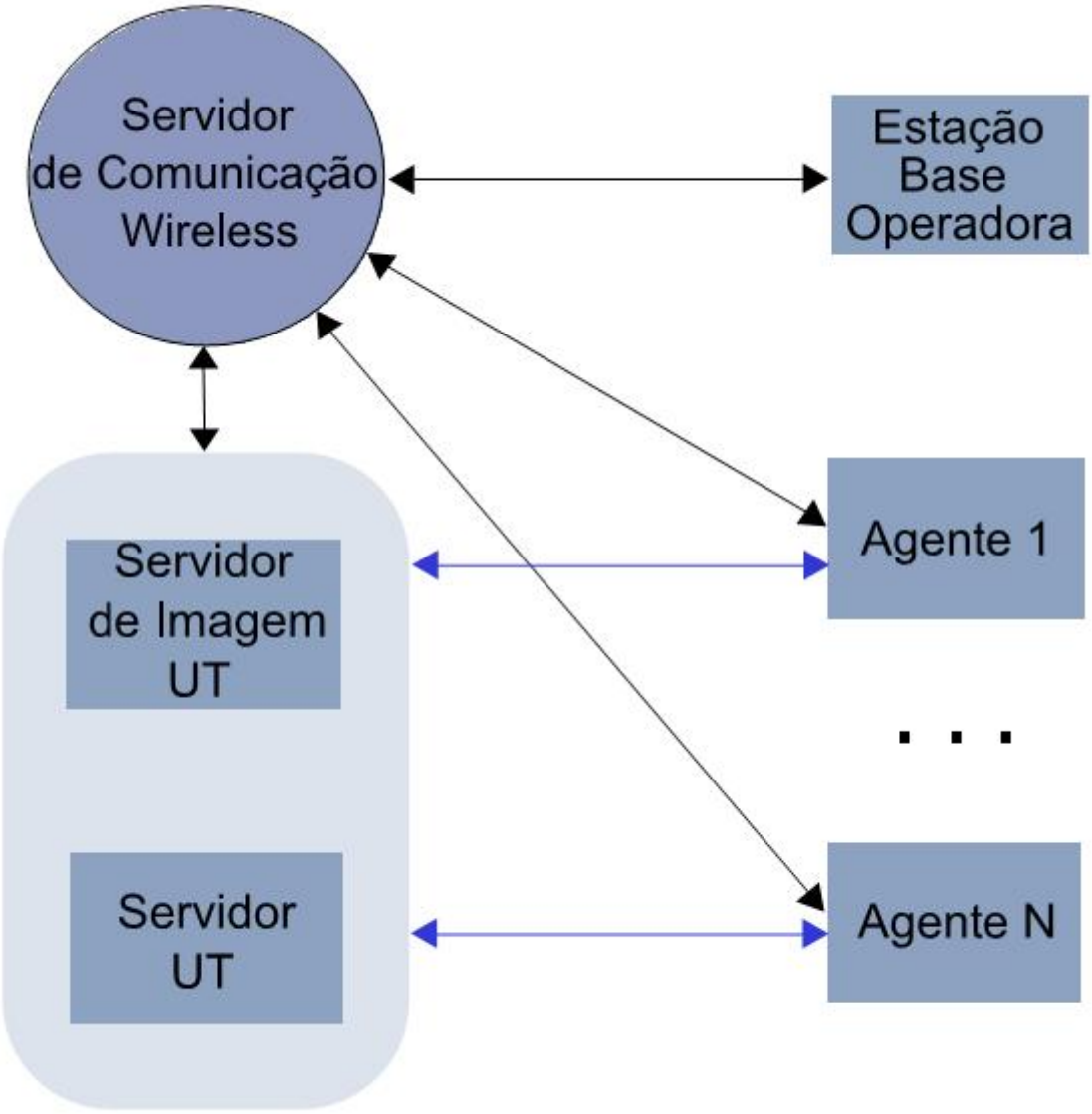

Figura 2.1: Servidor de Comunicação Wireless(VIRTUAL. . . ,)

2.4.1 Mobilidade

O teste de mobilidade consiste em iniciar um ou mais agentes no simulador em um ponto do mapa, definido pela organização, e a equipe deve fazer com que pelo menos um dos robôs chegue a um ponto do mapa definido pelos organizadores do evento. Porém, o tempo para que esse "alvo" seja atingido é limitado, portanto, não basta chegar ao alvo, é necessário fazê-lo antes de expirar o tempo. A princípio, o teste se dá na zona amarela (mapa fácil), mas metas adicionais podem ser aplicadas em zonas laranja ou vermelha. Durante esse teste as equipes podem usar sensores que capturem o relevo, e o mapa é comprimido, isso permite que as equipes tomem outras formas de navegação do mapa.

2.4.2 Mapeamento/Localização de Vítima

Para o teste de mapeamento os robôs precisam criar um mapa de grande percentagem do mundo. O mapa deve conter anotações de área explorada, localização das vítimas e todo o caminho traçado pelos robôs. A organização usada no teste é similar àquela que será usada durante a competição real, incorporando perfeita comunicação, apesar de promover ruídos nas leituras dos sensores. O sensor GroundTruth também é permitido nesta etapa. É obrigatório para conclusão deste teste que o time de robôs encontre e relate a localização de, pelo menos, uma vítima dentro do tempo previsto de duração do teste.

2.5 Cenários

Os cenários usados durante a competição são providos pelo e através do simulador, o USARSim. Com antecedência de um dia ao da competição, informações sobre o cenário são fornecidas às equipes. Um exemplo dessas informações são a dificuldade e o tipo de cenário. Existem também informações que devem ser divulgadas a priori, por exemplo, a localização de um evento, como o derramamento de um elemento químico em determinado ponto do mapa, ou simplesmente mapas do ambiente da execução. O formato do mapa é em GeoTiff. Deve-se considerar que os mapas podem não estar atualizados e podem, portanto, conter erros devido às renovações, passagens bloqueadas, etc.

2.6 Antes da execução

Antes das execuções, cada time recebe um documento de texto que contêm as coordenadas da posição de início (x, y, z) e orientação para cada robô. As coordenadas de posicionamento são expressas em metros, e as orientações são em radianos. O arquivo consiste em número de linhas iguais ao número de robôs, sendo que cada linha é composta pelo número dela seguido da coordenada do ponto de início e das orientações do robô a que ela se refere. Essas operações terão início trinta minutos antes da execução efetiva. Cada time irá se preparar no cluster que não está sendo utilizado. A execução começa no tempo programado. Independente da equipe não estar pronta o tempo começa a correr. Os robôs usam baterias que operam de 20 a 25 minutos. No tempo prescrito, os robôs devem ser inicializados dentro do simulador. Dentro do mapa os robôs devem aguardar por um comando de “start” que deve ser emitido antes que eles comecem a explorar a área. Quem monitora a condição da bateria, é o próprio robô. Informações não relatadas e colocadas no arquivo de “log” antes da bateria expirar não são contabilizadas

no total de pontos. Todos os robôs devem ser inseridos no mapa ao mesmo tempo, embora os times possam decidir ativá-los no momento que acharem mais conveniente.

2.7 Depois da execução

Para fins de pontuação, cada equipe deve prover os seguintes elementos:

- Obrigatório: Um arquivo de texto simples, chamado de arquivo de vítimas, contendo as vítmas encontradas durante a exploração. Este arquivo deve conter a localização (x,y,z) de cada uma das vítimas em um referencial coerente com o mapa e as localizações de início dadas. O formato do arquivo de vítimas é dividido em linhas. Cada linha contém, exatamente nesta ordem, o identificador da vítima (atribuído pela equipe), a posição em metros e a especificação de como a vítima foi encontrada. Como toda vítima possui um sensor, chamado de sensor da vítima, o último campo pode ter dois valores válidos. Se o sensor da vítima for usado para encontrá-la, então o valor atribuído ao campo é VSENSOR, porém, se não for usado esse sensor, o valor que o campo recebe é VISUAL. A última linha do arquivo de vítimas deve conter uma string com o valor END. O campo ID pode receber qualquer valor de string, pois serve, única e exclusivamente, para identificar a vítima descoberta pela equipe na posição relatada.

- Obrigatório: Um arquivo, chamado de arquivo bitmap, contendo o mapa e informações adicionais. Este arquivo deve conter múltiplos mapas sobrepostos. Todas as sobreposições devem ser georeferenciadas, isto é, devem possuir marcações que definam obstáculos, e precisam estar também no formato de arquivos GeoTiff ou Mif, além de cobrirem uma região por completo. A sugestão é que os mapas de sobreposições representem probabilidades ou agrupamentos de mapa, ou áreas cobertas por cada robô. As sobreposições devem representar a seguinte informação com as dadas colorações:

  - Não atravessável - Preto RGB(0, 0, 0)
  - Completo (atravessável, e livre de vítimas) - Verde RGB(0, 255, 0)
  - Incompleto (atravessável, porém não necessariamente livre de vítimas) - Branco RGB(255, 255, 255)
  - Inexplorado - Azul RGB (0, 0, 255)
  - Localização de Vítima - Vermelho (255, 0, 0). Locais podem ser indicados com uma cruz, um círculo ou um símbolo similar. O tamanho do símbolo deve ser de um tamanho que na escala do mapa não seja superior a um metro quadrado. Se uma

vítima for encontrada em movimento, deve ser indicado que a mesma estava a pé e deve ser apontado o seu caminho.

- Opcional: Qualquer informação recolhida durante a competição. Por exemplo, "screenshots" das vítimas, informações adicionais relativas às vítimas. Em caso de "screenshots" serem fornecidos, o arquivo deve ser nomeado consistentemente com os nomes utilizados no arquivo de vítimas, a fim de fazer uma associação é imediata.

- Todas as informações devem ser apresentadas ao árbitro em, no mais tardar, 15 minutos após o prazo de término da execução. Informações atrasadas não são aceitas para fins de pontuação.

É importante deixar claro que mesmo que uma equipe utilize dez agentes, ao final da etapa apenas um arquivo deve ser enviado ao árbitro. O operador que iniciou os robôs deve preparar este arquivo. Qualquer outra pessoa que manipule os arquivos gerados pelos robôs será contabilizada como um operador adicional. Cabe à equipe decidir qual dos arquivos deve ser entregue. Se uma fusão dos arquivos gerados for mais abrangente, é permitido que seja entregue este arquivo.

2.8 Pontuação

O sistema de pontuação baseia-se no critério de recompensar equipes que sejam capazes de explorar, de forma autônoma e segura, partes significativas do ambiente, proporcionando mapas de alta qualidade e demonstrando a capacidade de busca e localização de vítimas usando modelos robóticos consistentes. Ou seja, ganham pontos adicionais as equipes que mostrarem um desempenho superior ao esperado. Fatores de mérito:

- Um sensor especial de localização de vítima é disponibilizado para uso das equipes. Por este sensor ser utilizado, cinco pontos são atribuídos para equipe a cada vítima que for corretamente localizada. Uma boa localização significa encontrar a vítima no recinto correto e em um raio de até um metro do ponto exato em que esta se encontra. Para vítimas que estejam se movimentando, a localização deve estar a um metro da trajetória traçada pela vítima. Cinco pontos adicionais são dados para quem prover uma imagem da vítima. A imagem deve ter uma qualidade que seja no mínimo suficiente para avaliar estado da vítima. Se uma equipe opta pela utilização de processamento de imagem para detectar vítimas, então dez pontos são acrescidos por fornecimento de imagem e de informações

de localização. Como foi dito anteriormente, a foto deve ter uma qualidade que possibilite uma avaliação do estado da vítima.

- Até 50 pontos são concedidos por exploração. As equipes são obrigadas a fornecer mapas consistentes com formato formalmente descrito. Equipes que completam um bom percentual do ambiente recebem 50 pontos. A quantidade é definida pelos árbitros do evento. Para porcentagens mais baixas, o número de pontos sofre um decréscimo linear de zero a cinco pontos de pena (até a penalidade atingir o valor máximo dado pela exploração). Por cada vítima não reportada que esteja numa área tida como completada (zona marcada com a cor verde no arquivo de bitmap).

- Até 50 pontos são concedidos por mapeamento. O mapeamento é julgado pelos seguintes critérios:

  - Qualidade de métrica – A precisão do arquivo de mapa comparado com o terreno real.
  - Fusão de leituras dos veículos – As equipes devem transformar todos os arquivos de mapa em um só arquivo. As equipes que incluem as informações de saída de vários robôs numa só têm um bônus na pontuação.
  - Atribuição - Uma das razões para se gerar um mapa é transmitir informações. Esta informação é frequentemente representada como atributos no mapa. Pontos são acrescidos por inclusão de informações sobre a localização de vítimas, localização de obstáculos, e os caminhos que os robôs tomaram.
  - Agrupamento – A mais alta tarefa de mapeamento é reconhecer que elementos discretos no mapa constituem outros elementos. Por exemplo, o fato de várias paredes formarem um cômodo ou um conjunto particular de obstáculos serem um carro. Pontos de bônus são concedidos por notar tais grupos no mapa.
  - Qualidade no esqueleto do mapa – O esqueleto do mapa reduz um mapa complexo em um conjunto de locais conectados. Por exemplo, quando se representa um corredor com um número elevado de saídas, um esqueleto pode representar esse corredor como uma linha, e símbolos ao longo dessa linha podem representar as portas. Um mapa pode ser impreciso em termos de medidas (um corredor pode ser representado com 20 metros de comprimento, ao invés de 15 metros), mas pode ter uma estrutura, ou esqueleto, precisa, já que as informações contidas no esqueleto não dependem da métrica. A categoria permite que os juízes dêem pontos por quão preciso o esqueleto do mapa está representado.

  - Utilidade - Um dos principais objetivos de fornecer um mapa é criar a possibilidade de uma primeira resposta ao utilizar o mapa para determinar quais áreas foram completadas, onde os riscos podem ser localizados, e onde as vítimas foram capturadas.

Fatores de penalidade:

- Operadores: se a equipe usa N operadores, a pontuação será dividia por N+1. Isso implica que cada equipe usará apenas um operador, para não perder pontos.
- Alarmes falsos: Para cada vítima que for incorretamente reportada é aplicada uma penalidade de cinco pontos. Um alarme falso é tido quando a vítima está posicionada a mais de um metro do local que fora relatado. No caso de uma vítima que esteja se movimentando isso se aplica à distância da sua trajetória.
- Choque: Cada vítima tocada por um robô irá incorrer numa penalidade de cinco pontos. Se um robô toca uma vítima, a equipe é penalizada com cinco pontos, porém se o mesmo robô tocar a mesma vítima várias vezes, não é penalizado mais de uma vez. Ou seja, para sofrer mais de uma penalidade neste caso, o robô teria que acertar vítimas diferentes. Choques com as vítimas são detectados automaticamente pelo servidor e suas informações são guardadas no arquivo de log. Bater nas paredes ou outras estruturas não acarretam pena, no entanto algumas estruturas podem ser instáveis e bater poderia causar um colapso. Operadores: Por questões de pontuação, um membro da equipe é considerado um operador logo que:
- Inicia um robô, entra com os dados de ponto de partida, ou exerça qualquer operação que seja necessária para o sucesso do início da missão dos agentes.
- Controla um robô de forma ativa.
- Pára um robô antes do término da execução. Para, por exemplo, prevenir um choque do robô com uma vítima.
- Estiver envolvido de qualquer forma com o processo de reconhecimento de vítima. De acordo com o descrito acima, cada equipe deve ter pelo menos um operador para configurar os robôs.

2.9 Política de Open Source

A equipe que for vencedora, e as equipes que apresentarem implementações de código consideradas inovadoras devem disponibilizar para os organizadores uma cópia completa e funcio-

nal dos softwares criados. Os softwares serão compartilhados em http://www.robocuprescue.org/wiki/index.php?title=Virtualrobots, dando os devidos créditos aos autores. O código fonte da competição realizada no ano anterior pode ser encontrado nessa mesma página.

2.10 Combinações de sensores

Qualquer combinação dos sensores listados abaixo é permitida, porém, combinações não-realísticas entre robôs e sensores podem ser vetadas. Por exemplo, um robô Zerg utilizando um sensor que ele não possui, como o SickLMS.

- Encoder (codificador)
- GPS
- Hokuyo
- INS
- Odometry (odometria)
- RobotCamera (câmera robótica)
- SickLMS
- Sonar
- VictSensor (sensor de vítima)
- RFIDSensor
- TouchSensor (sensor de toque)

2.11 Combinações de robôs

As equipes podem usar qualquer combinação dos robôs representados nas figuras(ROBôS. . . ,) 2.2 à 2.8:

Figura 2.2: atrvjr

Figura 2.3: hammer

Figura 2.4: p2at

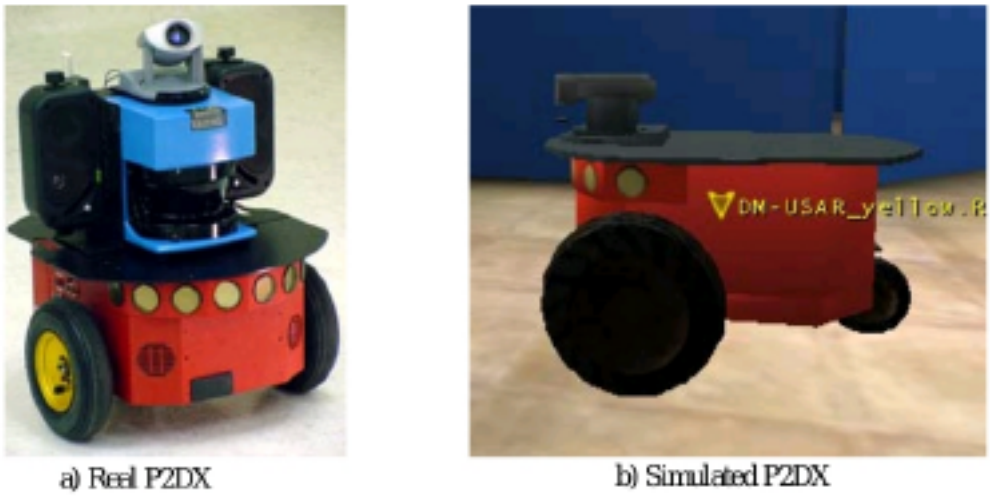

Figura 2.5: p2dx

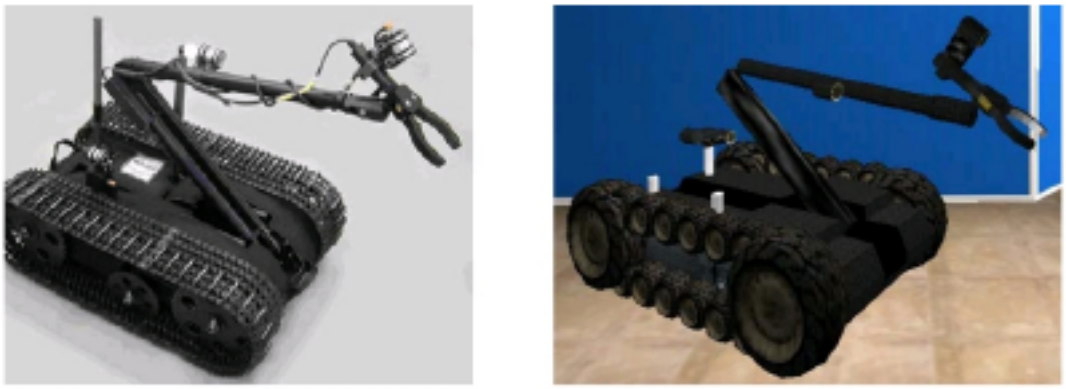

Figura 2.6: talon

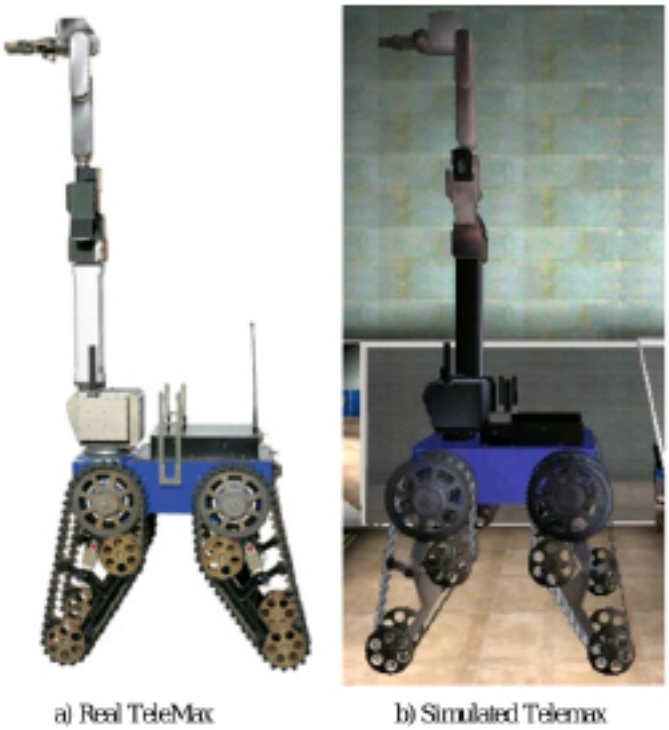

Figura 2.7: telemax

a) Real Zerg b) Simulated Zerg

Figura 2.8: zerg

3 *Tutorial para participação na Virtual Robots Competition*

Neste capítulo, serão tratadas as instalações das ferramentas necessárias para que se possa ingressar na Virtual Robots Competition.

3.1 Instalação no windows

Para que a instalação ocorra sem nenhum problema, abaixo são listados alguns requisitos desejáveis de hardware. É importante ressaltar que são requisitos recomendáveis, isto é são apenas requisitos em que se garante a instalação de forma aceitável já que não foram testadas instalações para configurações inferiores.

Requisitos de hardware

- Pentium III ou AMD Athlon 1.0 GHz
- Placa de vídeo 32 MB compatível com DirectX 9
- 128 MB RAM
- Espaço no HD: 6 Giga

O primeiro passo é fazer a instalação do jogo UnrealTournament 2004. Pois o simulador que é utilizado na competição é baseado na engine deste jogo. É necessário que seja a versão completa do UT2004.

Após a instalação do UnrealTournament 2004 é necessária a instalação do patch 3369 na mesma pasta onde foi instalado o UT2004. Esse patch pode ser encontrado em vários endereços indexados pelo buscador do Google. Este pacote conserta alguns problemas que influenciam o engine do jogo, e esse engine é usado pelo USARSim.

Problemas de engine consertados pelo patch 3369:

- Caso seja passada uma fonte inválida, o método UCanvas::WrappedPrint() faz um retorno de função ao invés de fazer uma assertiva, como era antes.
- -Mod= agora suporta exportação apropriada de headers.
- É feita conversão de 'l' para 'I' no nome dos players
- Copiar e colar no MidGamePanel.uc foi modificado, como era antes podia causar problemas em mudanças de level.
- Renderização de projetor. Neste caso, além de arrumar, foram feitas algumas otimizações.
- Suporte à 64 bits com várias modificações.
- Textura, antes a tela podia ser renderizada de cabeça para baixo
- Efeitos climáticos no OpenGLDrv.

Após ter sido completada a instalação do patch 3369, alguns arquivos devem ser baixados de mirror.optus.net/sourceforge/u/us/usarsim/. Faça o download de:

1. USARSimBaseFiles_3.11.zip
2. USARSimFull_3.11.zip
3. AAA_MapBaseFiles_V3.1.zip
4. DM-YellowArena_V3.1.zip

A instalação do USARSim e seus respectivos mapas encontram-se nestes zipados.

Descompacte todos os arquivos e copie seus conteúdos para a pasta UT2004, sobrescrevendo o que for necessário.

Dentro do diretório UT2004, entre na pasta System e dê um duplo clique no arquivo make.bat. Se tudo estiver correto será gerada a seguinte mensagem "Success - 0 error(s), 0 warning(s)"ao final da execução do make.

Para testar a instalação execute o arquivo DM-USAR_yellow_250.bat que se encontra no diretório do UT2004 na pasta \USAR_Maps_Files\RunServer. Se não houver erros durante a execução deste .bat, então a instalação do simulador foi concluída com sucesso.

O controlador

Faz parte da instalação a escolha de um controlador. Neste tutorial foi escolhido o SimpleUI_2.1, porém, este requer a instalação de uma das versões do Visual C++. Versões do Visual C++ podem ser encontradas em http://msdn.microsoft.com/en-us/visualc/default.aspx.

Baixe de mirror.optus.net/sourceforge/u/us/usarsim/ o arquivo SimpleUI_2.1.zip. Terminado o download descompacte o arquivo e copie seu conteúdo na pasta do UT2004.

Em \System dentro do diretório do UT2004 abra com um editor de texto o arquivo usar_s.bat

Usando o comando rem comente a primeira linha, que deve ficar assim:

- rem ucc server DM-ARDA_250?game=USARBot.USARDeathMatch?TimeLimit=0?GameStats=False -ini=USARSim.ini -log=usar_server.log

E descomente, removendo o comando rem, a segunda linha, que deve ficar assim:

- ucc server DM-USAR_yellow_250?game=USARBot.USARDeathMatch?TimeLimit=0?GameStats=False -ini=USARSim.ini -log=usar_server.log

Salve, feche este arquivo.

Para testar a instalação, dê um duplo clique em \System\usar_s.bat Se não der nenhum erro, então o server está funcionando. Sem fechar a tela em que o server está rodando, vá em \SimpleUI\Release no diretório do UT2004 e clique em SimpleUI.exe. Clique em Start. Deve abrir uma tela com o UnrealTournament 2004 e em seguida uma tela explicando como navegar entre o simulador e o controlador.

Pronto, daqui pra frente é só usar o Visual C++ para inserir seu código no controlador e participar da Competição!

4 O Problema

Como foi visto, para se participar da Virtual Robots Competition é necessário utilizar-se de muitas ferramentas, como o simulador e uma ferramenta de interação com o USARSim. Estas ferramentas de interação são conhecidas como controladores, e elas são as responsáveis por inserir os agentes, os dados e informações necessárias para a competição, além dos algoritmos de cada equipe. Como os controladores são responsáveis pela comunicação entre um agente e o operador, ele precisa ser de fácil acesso e de claro entendimento para qualquer pessoa. Porém ao participarem de algumas edições de eventos de simulação de resgate ligados à Virtual Robot Competition da RoboCup, muita gente percebe a dificuldade de aprender a linguagem ou a forma como entrar os dados nessas ferramentas. Existem algumas em que a curva de aprendizado não é tão grande, mas elas são de certa forma incompletas, pois não trazem em si algumas das coisas que a categoria necessita, deixando a cargo das equipes a implementação de coisas que deveriam ser intrínsecas ao programa de controle.

Em algumas dessas competições alguns alunos e professores do Instituto Tecnológico da Aeronáutica e da Universidade Estadual Paulista perceberam também essa dificuldade em introduzir e retirar informações do ambiente simulado. E eles concluíram que essas dificuldades se dão pela forma como as ferramentas controladoras tratam as informações, de modo não intuitivo e muitas vezes nem mesmo visual. Ao se depararem com este inconveniente, decidiram que uma boa maneira de contornar essa situação seria criar um controlador próprio.

O professor Alexandre da Silva Simões da Universidade do Estado de São Paulo e a professora Esther Colombini do Instituto Tecnológico da Aeronáutica fizeram uma primeira versão do controlador Brasil Virtual Robots, o brasilvr. Para uma nova versão deste controlador foram chamadas pessoas de outras instituições, foi criada uma nova arquitetura e foram delegadas as obrigações de cada membro na nova equipe, dentre as instituições agregadas ao novo time está UFSC. O objetivo da nova versão do controlador é ser multi plataforma, escalável e intuitivo o bastante para ser usado também como ferramenta didática de Inteligência Artificial.

O trabalho foi dividido entre as universidades, considerando-se o número de pessoas envol-

vidas em cada instituição, o conhecimento da área e a motivação de cada integrante. Coube a UFSC, na forma de trabalho de conclusão de curso, e a alguns integrantes da UNESP a parte de localização e mapeamento simultâneo (SLAM).

Para que o algoritmo de SLAM fosse criado, foi necessário estudar como são usadas as informações dos agentes e como é organizada a Virtual Robots Competition, além de estudar o problema do SLAM, suas formas de abordagem as formas de trabalhar para chegar ao melhor caso, claro, considerando uma precisão de mapa que seja capaz de passar pela arbitragem do evento.

4.1 O problema do SLAM

O problema de localização e mapeamento simultâneo consiste em fazer com que um agente inserido num ambiente, consiga gerar um mapa e ao mesmo tempo se localize neste mapa sem possuir informações adicionais, excetuando o ponto em que foi inserido no ambiente e a sua orientação inicial. Alguns autores definem o problema do SLAM como sendo uma forma de capacitar um robô a responder de forma adequada à pergunta “Onde eu estou?”. É importante ressaltar que a localização do agente e o mapa criado são dependentes de forma cíclica, em outras palavras, a entrada de dados de um é saída de dados do outro. Ou seja, como a localização depende do mapeamento e o mapeamento precisa da localização, um erro em um deles vai gerar um erro no outro e, portanto, todas as informações obtidas na seqüência vão ecoar este erro.

A figura 4.1 representa o SLAM em sua essência:

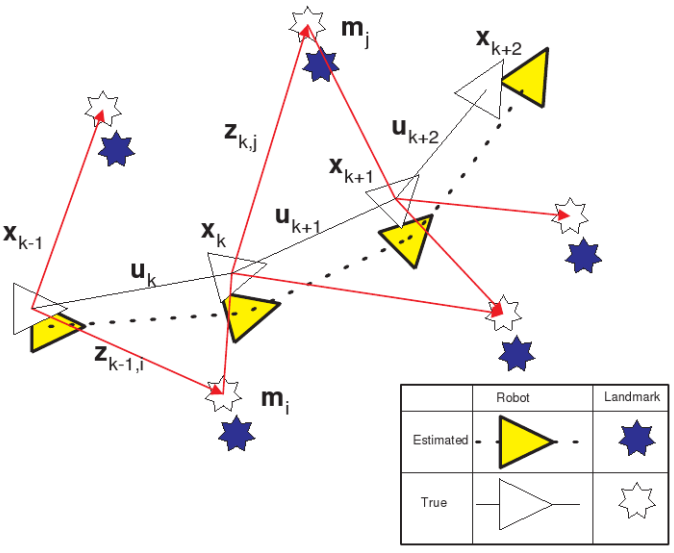

Figura 4.1: SLAM(BAILEY, 2006)

Considerando o modo como as informações estão descritas num ambiente e a forma de interação com estas informações, o SLAM se divide em quatro famílias, apresentadas abaixo:

1. Abordagens que provem uma grade de ocupação do mapa.
2. Abordagens que organizam de forma topológica o mapa.
3. Abordagens que são um híbrido das outras abordagens de SLAM.
4. Abordagens que empregam filtro de Kalman baseadas em características do mapa, como por exemplo, marcações (landmarks) e/ou obstáculos.

4.1.1 Aspectos intrínsecos ao SLAM

Alguns aspectos que são trazidos com o problema de localização e mapeamento simultâneo devem ser considerados, entre eles estão:

- Limitações dos sensores e ruídos nas medições;
- Erros de odometria e movimentação;
- Acúmulo de erros;
- Associação de dados;
- Ambientes e obstáculos dinâmicos;
- Exploração do desconhecido.

Limitações dos sensores e ruídos nas medições

Os robôs em geral possuem uma gama de sensores, que vão de raios infravermelhos até sensores de áudio (sonares). Dentre estes sensores, alguns são usados pelos algoritmos de SLAM. Porém, todos são passíveis de erros e limitações, podendo gerar leituras infiéis à realidade. Esses erros podem ser causados tanto pela qualidade dos sensores quanto pelos ruídos do ambiente. Um exemplo comum está nas superfícies translúcidas, onde não funcionam corretamente os lasers.

Erros de odometria e de movimentação

Já que para explorar um ambiente um agente precisa navegá-lo, é comum que alguns problemas de odometria (equipamento destinado a medir a distância percorrida) e movimentação surjam. Entre os fatores que geram erros na movimentação, podem ser elencados alguns comuns como, superfícies escorregadias, terrenos desiguais (buracos, morros, etc), solavancos decorrentes de obstáculos, entre outros fatores. Quando alguns destes problemas ocorrem, a leitura real da distância percorrida e a posição em que o agente se encontra podem ser equivocadas.

Acúmulo de erros

Como existem erros nas leituras dos sensores e na movimentação do agente, isso pode ocasionar erro no cálculo da posição do agente e o mapa por ele gerado. O erro seria simples se estas leituras fossem independentes, o que não é caso do problema do SLAM, já que a posição atual e o mapa gerado são entradas para o novo cálculo. Ou seja, um erro gerado numa primeira leitura afetará as demais leituras. O funcionamento do algoritmo consiste em estimar a posição do robô num momento seguinte, tendo apenas a sua posição atual, o vetor de orientação e as leituras dos sensores. Ao confrontar a posição atual e a estimativa de onde o robô deveria estar é feito o cálculo dos erros e os dados gerados servem como entrada para os cálculos de geração do mapa. O mapa gerado por sua vez é usado para o cálculo da estimativa da próxima localização do robô. Ou seja, a saída do algoritmo de localização é um dado de entrada para o cálculo do mapa, e a saída do cálculo do mapa é usado como entrada para o cálculo de localização. Isso acaba gerando um círculo vicioso e o erro gerado é conhecido como erro de acumulação. Visualmente, o que acontece é que o mapa não tem alta fidelidade com a disposição do ambiente e pode haver deslocamento de objetos por conta disso.

Associação de dados

Como o agente inserido no ambiente estará navegando por ele, pode ser que este robô passe pelo mesmo local diversas vezes. Essa “redundância” de informação tem um valor inestimável e pode ser um grande benefício para diminuir o acúmulo de erros, já que as informações geradas podem ser comparadas e aproximadas. Porém, essa informação pode ser interpretada de forma errada, como por exemplo, comparar duas leituras de coisas diferentes achando que se tratam da mesma informação, além do fato, de que, com o aumento do número de vezes que um agente passa pelo mesmo ambiente, aumentará também progressivamente as comparações.

Ambientes e obstáculos dinâmicos

Dependendo do ambiente em que o agente seja inserido, as informações prévias que ele possui podem estar desatualizadas. Por exemplo, em um cenário em que haja pessoas, veículos ou até outros agentes, estes podem ter se movido ou algum objeto pode ter sido retirado do local. Não existe uma solução geral para esse problema. Nestes casos, costuma-se fazer com que os agentes explorem por uma faixa de tempo as localidades do mapa considerando que o ambiente seja estático.

Exploração do desconhecido

Tipicamente, as informações que são passadas ao agente sobre o ambiente são mínimas, e o agente tem de estar preparado para lidar com situações imprevistas, como buracos ou escadas. O robô precisa ter em seu conjunto de estratégias algumas formas de lidar com situações inesperadas, podendo assim, contornar problemas e continuar sua navegação de forma autônoma, sem que isso atrapalhe seu objetivo principal de mapear e localizar-se simultaneamente.

Abordagens

Como foi comentado anteriormente, as formas de se lidar com SLAM podem ser agrupadas em quatros famílias, portanto, neste ponto será explicado de forma mais detalhada cada uma delas.

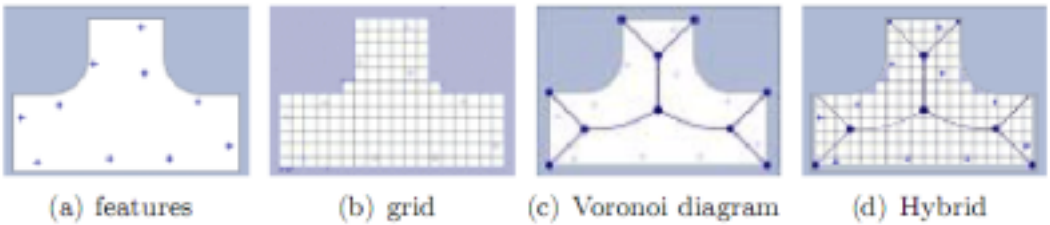

Figura 4.2: Famílias do SLAM(PFINGSTHORN, 2006)

Mapas métricos: Grade de Ocupação (b)

Grades de ocupação fornecem uma representação intuitiva de ambiente simulado. Os primeiros a usarem essa abordagem para mapeamento foram Elfes (ELFES, 1987) e Moravec (MORAVEC, 1988). As grades de ocupação são compostas por células, cada uma delas possui um valor que define se a mesma está ou não ocupada, ou seja se a região da célula mapeada está

ou não ocupada. Na forma mais simples, o valor agregado às células é binário, ou seja, apenas é garantido que naquela região mapeada pela célula possui ou não um obstáculo, sem considerar outras características. É possível também colocar um terceiro valor que defina regiões ainda desconhecidas aos agentes. Numa outra abordagem, usam-se valores em ponto flutuante, sendo que valores compreendidos entre zero e um representam a certeza da informação se a célula está ou não ocupada. A abordagem de grades de ocupação possui muitos benefícios quando comparada com as demais. O maior desses benefícios é que a representação fornecida do ambiente pode ser usada diretamente para a navegação do agente, facilitando também na criação dos algoritmos de desvio de obstáculos e dos algoritmos de aprendizado. Outro grande benefício de usar grades é que a resolução pode ser sintonizada até que seja possível detalhar ao máximo o mapa, por menores que sejam esses detalhes. Na realidade a única coisa que limita o nível de detalhamento do mapa gerado é a capacidade dos sensores e a sua taxa de erro. Por essa razão, essa é a abordagem mais usada quando se necessita de um mapa extremamente detalhado. O algoritmo de mapeamento é relativamente simples. Ele usa apenas a técnica de ray-casting e dois contadores em cada célula, um para acertos (hits) e outro para erros (misses). Ao início do algoritmo os contadores estão zerados. A técnica de ray-casting é quem vai incrementá-los. A técnica se utiliza de raios imaginários que são lançados pelos sensores. Quando um obstáculo é interceptado por essa reta traçada pelo raio, são incrementados os contadores de misses de todas as células que estão entre o robô e o obstáculo por ele detectado. No caso das células que tiverem sido atingidas pelo raio, os contadores de hits é que são incrementados. Para o cálculo da ocupação de cada célula é feito o cálculo de um limiar que define se a célula receberá o valor um (ocupada) ou o valor zero (não-ocupada). O limiar é definido usando-se o seguinte cálculo: (contador de hits)/(contador de hits + contador de misses). O resultado é arredondado para um ou para zero. Caso haja noção de região inexplorada basta verificar quais células possuem ambos os seus contadores zerados. Quando se trata de ocupação usando ponto flutuante, a taxa de acertos contra a taxa de acertos mais a taxa de erros é guardada na própria célula. O grande problema desta abordagem é a quantidade de dados que ela agrega, além do fato de não ser boa para ambientes abertos. Em resumo, essa abordagem com grades é excelente quando se necessita extrema precisão, porém para ambientes abertos ou muito grandes pode ser difícil de programar devido à quantidade de dados que precisam ser processados.

Mapas topológicos: Grafos e Diagramas de Voronoi (c)

Nessa abordagem são usadas, em geral, duas formas de representar o ambiente, grafos e diagramas de Voronoi. A primeira coisa que pode ser ressaltada neste tipo de abordagem é a facilidade que ela traz para os algoritmos de planejamento de caminho. Dentro do ambiente,

os agentes necessitam sair de um ponto em chegar a outro. Com um mapa topológico isso se torna fácil, pois é fácil extrair esse tipo de informação de grafos ou de diagramas de Voronoi. Outro ponto importante aqui, é que nesta abordagem, os mapas são representados de forma compactada, por isso, a quantidade de dados agregados é menor. O fato é que essa abordagem é uma forma de explorar a navegabilidade do mapa, portanto foi realmente pensado em como seriam as melhores formas de se locomover nas melhores rotas, e para isso, tanto grafos quanto diagramas de Voronoi são excelentes. Neste caso de SLAM, a única diferença entre os grafos e os diagrams de Voronoi é a forma como as ligações são tratadas. Nos grafos, as ligações são construídas apenas para caminhos percorridos realmente, já nos Diagramas de Voronoi elas são estimadas e as suas posições são afixadas considerando as posições dos obstáculos. Portanto grafos vão mostrar o caminho percorrido que irá garantir a sua consistência. Por outro lado, Diagramas de Voronoi são capazes de generalizar para além dos caminhos percorridos. Na verdade, com isso pode-se inferir o percurso mais seguro com base em estimativas do obstáculo. Em suma, usar a abordagem topológica é excelente para encontrar caminhos seguros, porém, essa abordagem não possibilita a captura de informações adicionais do ambiente, e os detalhes da geometria do local são poucos.

Mapas híbridos (d)

Nessa abordagem podem surgir inúmeras combinações, o importante aqui é tentar agregar das outras abordagens as vantagens e minimizar as desvantagens, como por exemplo, mesclar uma mapa de grade com uma abordagem topológica, fazendo que se possam coletar informações precisas do ambiente à medida que os agentes façam um caminho mais seguro e melhor. Desta abordagem surgiram duas abordagens muito usadas, o Atlas e o FastSLAM, sendo a segunda a utilizada como base para este trabalho, e será, adiante, melhor explicada.

Mapas baseados em características: Filtro de Kalman (a)

É a abordagem mais antiga de todas e, segundo Bayu Slamet e Max Pfingsthorn, é a mais influente dentre as existentes. Houve uma série de estudos em cima do filtro criado por Rudolf Emil Kálmán, que resultaram na abordagem atual utilizada pelo problema do SLAM. Mapas em que se usa o Filtro de Kalman são limitados apenas pela posição atual do agente e pelas posições estimadas das marcações, chamadas aqui de landmarks. A idéia básica desta abordagem é descrever o mapa como uma distribuição gaussiana usando-se a combinação da posição do robô e posição dos landmarks. O mapa é representando por μ que é um vetor-estado que possui todas as variáveis relevantes para a distribuição gaussiana. Este vetor-estado possui um

tamanho (3 + 2N) que é constituído pelas informações de posição (em 2D) do robô (x, y, θ) e a posição representada por duas informações (x, y) de todos os N landmarks.A covariância desta distribuição Gaussiana representa a incerteza sobre a combinação posição atual do robô e posição dos landmarks. Através da recursão é encontrada e atualizada a distribuição Gaussiana, ou seja, encontrando as soluções mais prováveis do mapa. A princípio, o algoritmo possui um mapa sem landmarks. Então, o mapa consiste apenas na posição do robô e a matriz de covariância que representa a incerteza do posicionamento. Assim que novos landmarks são encontrados, tanto o vertor-estado quanto a matriz de covariância são adequados para comportar as novas informações. A posição dos primeiros landmarks está apenas co-relacionados com a medição da distância entre o marco e o robô. Essa medição é projetada num quadro global de coordenadas. A covariância, num primeiro momento, segue apenas as incertezas nas medições e no posicionamento. Os landmarks inicialmente são baseados apenas na posição do robô, à medida que o mapa é re-estimado a correlação diminui e a posição do landmark na última estimativa e na nova estimativa é relacionada. Para fazer apenas uma atualização no mapa são necessários vários passos. O primeiro passo deste processo é atualizar a posição do robô considerando um modelo linear de movimentação que descreve como os atuadores afetam a pose do robô na fatia de tempo anterior e na atual fatia de tempo. Quando um landmark é encontrado, isso pode gerar uma nova marcação no mapa ou uma remarcação, já que pode ser apenas a releitura de um landmark anteriormente detectado. Apena novos landmarks são incorporados ao mapa. Com a diminuição das releituras dos landmarks, diminui também a precisão do mapa dado que os atuadores do agente possuem falhas. Essas falhas são replicadas na matriz de covariância que faz que com o valor da posição do robô seja cada vez maior, isso acontece até que uma nova releitura seja feita. Em geral, as releituras fazem com que haja uma queda brusca no valor das incertezas. E uma queda no valor das incertezas torna mais preciso o valor da posição do agente, que por sua vez faz com que o valor da posição dos landmarks aumente a precisão também. O processo do SLAM baseado no filtro de Kalman pode ser resumido nos passos abaixo:

- Dado: Estimativa anterior do mapa μt-1 e Σt-1
- Resultado: Estimativa atualizada do mapa μt e Σt

1. μt = μt-1 + But
2. Σt = Σt-1 + Σatuador
3. Kt = Σt C T (CΣt C T + Σmedições)-1

4. $\mu t = \mu t + Kt\ (zt - C\mu t\ )$

5. $\Sigma t = (I - Kt\ Ct\ )\Sigma t$

No primeiro passo, o mapa é atualizado considerando o modelo linear de movimentação(B) e o comando de movimentação que foi passado ao agente (ut). No segundo passo as incertezas do mapa são atualizadas considerando o somatório das incertezas dos atuadores (Σatuador). Nas equações acima é considerado que os landmarks são fixos, e por isso apenas a posição do robô é atualizada. Em casos em que não se assegura a estática como característica dos landmarks, estes também são atualizados a cada nova observação. No terceiro passo é calculado o ganho de Kalman, é aqui que são propagadas as incertezas para o mapa. Neste cálculo também são incorporadas as incertezas nas medições (Σmedições), considerando a observação atual (zt). A matriz C é apenas uma conveniência que descreve o mapeamento do vetor-estado numa observação atual(zt). A matriz do ganho de Kalman (Kt) tem um tamanho 3 pelo tamanho do vetor-estado, e geralmente essa matriz não é esparsa. Nos passos quatro e cinco o vetor-estado e a matriz das incertezas são atualizados afetando diretamente no conhecimento do ambiente que tem o agente. A principal característica da abordagem com filtro de Kalman é que todo o estado posterior é estimado sobre o mapa de forma on-line. Isso implica na manutenção em tempo integral das incertezas no mapa. Outra característica importante da abordagem de Kalman, é que todo ruído é governado por uma distribuição Gaussiana. Isso pode ser considerado como um problema que impõe severas limitações ao algoritmo, já que, com exceção da associação dos dados, os outros ruídos gerados são governados por outras funções ou distribuições, como no caso do ruído do odômetro, onde as funções são trigonométricas. Em resumo, filtro de Kalman é uma solução completa para SLAM, que se baseia em um algoritmo incremental. Além disso, ele tem várias propriedades interessantes, e é relativamente fácil de compreender e aplicar. No entanto, alguns dos pressupostos que são impostos pelo modelo Gaussiano são claramente pouco precisos. Além disso, a abordagem coloca alguns desafios significativos a ultrapassar antes de poder ser aplicado com sucesso. O principal desafio envolve o esforço significativo que se tem para colocar em prática os detectores e sensores. Outro desafio é conseguir balancear o número de landmarks para que se alcance um mapa desejado.

5 *Algoritmos e Implementações: Experimentos e Resultados*

Neste capítulo, será mostrado como foi feito o algoritmo de SLAM considerando as escolhas e abordagens. Para melhor entendimento, sugere-se que previamente seja feita a leitura do capítulo que trata do problema do SLAM.

5.1 A abordagem escolhida

Para escolher qual abordagem seria usada no problema do SLAM dentro da Virtual Robots, foi necessário entender como são tratadas as informações por cada uma delas, estudando inclusive aquelas que são híbridas. Além das características das abordagens, outro fato importante que necessitou apreciação foi a forma como são disponibilizados os dados e como eles podem ser processados considerando-se todas as regras, e restrições impostas pela organização do evento. Como no caso da Virtual Robots Competition, só se é sabido previamente o local do mapa em que serão colocados os agentes e suas orientações, este fato foi quem peneirou as abordagens, deixando como melhores opções àquelas baseadas em landmarks, ou seja, aquelas que usam obstáculos para fazer marcações no mapa, ajustando-o gradativamente com embasamento numa distribuição Gaussiana. Nas abordagens tradicionais que se baseiam no estudo de Kalman os mapas não podem ser dinâmicos, portanto, obstáculos devem ser imóveis. Já que no caso da competição da RoboCup esta estaticidade não pode ser garantida, as abordagens tradicionais também foram descartadas, e por isso foi escolhido o fastSLAM que usa como princípio a fatoração de Cholesky, considerando a simulação de Monte Carlo e o , ou Extended Kalman Filter, que é uma evolução do filtro de Kalman.

5.2 FastSLAM

É uma abordagem eficiente que se baseia numa simples fatoração para a resolução do problema de mapeamento e localização simultâneos. Com o FastSLAM, passam a ser considerados modelos não-lineares e distribuições não-gaussianas. Em termos probabilísticos, o problema do SLAM pode ser descrito na seguinte forma:

$$P(x_k, m|Z^k, U^k, x_0)$$

Figura 5.1: Fórmula (distribuição probabilística)(NIETO JOSE GUIVANT, 2003)

Onde x_k representa o vetor de posicionamento do agente no tempo k, m é o vetor-estado que representa o mapa, Z_k é o conjunto de observações do mundo até o tempo k, U_k é o conjunto de entradas de controle de movimentação e o vetor x_0 que representa o ponto inicial onde foi inserido o agente. Em termos probabilísticos, o problema do SLAM é um processo de Markov. Ou seja, um processo estocástico, portanto o estado do mapa no tempo k-1 engloba toda a informação necessária para propagar o sistema para o estado no tempo k. O modelo de movimentação do robô é considerado um processo de Markov e em termos de distribuição probabilística pode ser descrito assim:

$$P(x_k|x_{k-1}, u_k)$$

Figura 5.2: Fórmula (distribuição probabilística considerando processo de Markov)(NIETO JOSE GUIVANT, 2003)

O modelo de observação, que diz respeito às observações do estado do ambiente, é descrito por um modelo probabilístico que obedece, também, à propriedade de Markov:

$$P(z_k|x_k, m)$$

Figura 5.3: Fórmula (modelo de observação probabilístico)(NIETO JOSE GUIVANT, 2003)

Para se atingir o mapeamento e localização simultâneos, usa-se o filtro de Bayes. Porém, para mapas tridimensionais acaba se tornando uma opção muito custosa. A alternativa para o filtro Bayessiano é a utilização do Extended Kalman Filter. O EKF lineariza funções não-lineares, e pode ser descrito da seguinte maneira;

1. xk = f(xk-1, uk) + wk

2. zk = h(xk) + vk

A função f calcula o próximo estado (estado previsto) considerando o estado anterior e a função h calcula a previsão de medição considerando o estado previsto. Porém, essas funções não podem ser aplicadas diretamente à covariância. Ao invés disso, é computada a matriz jacobiana. Esta matriz é atualizada a cada tempo k, com os valores das predições. Essas matrizes por sua vez, são aplicadas no filtro de Kalman. Ou seja, lineariza-se a função que depois é filtrada com o filtro de Kalman. O EKF, porém, pode ser frágil, e pode falhar catastroficamente em algumas situações, já que só lidam com distribuições unimodais (distribuições que possuem apenas um máximo local). A idéia por trás do FastSLAM é a fatoração. Essa idéia é baseada na observação de que se o caminho do robô fosse realmente conhecido, então, todas as marcações (landmarks) seriam mutuamente independentes. Na prática, o caminho é obviamente desconhecido, porém, essa independência condicional nos permite estimar a posição posterior do robô da seguinte forma fatorada:

$$\begin{aligned} & P(x^k, m | Z^k, U^k, x_0) \\ &= P(m | x^k, Z^k, U^k, x_0) P(x^k | Z^k, U^k, x_0) \\ &= \prod_{i=1}^{N} P(m_i | x^k, Z^k, U^k, x_0) P(x^k | Z^k, U^k, x_0) \end{aligned}$$

Figura 5.4: Fórmula (Posição posterior na forma fatorada)(NIETO JOSE GUIVANT, 2003)

A fatoração é a idéia fundamental por trás do FastSLAM. Com ela, o problema é decomposto em localização e estimativa da posição dos N landmarks. Para isso, o FastSLAM necessita de um filtro de partículas para poder estimar a posição posterior do agente:

$$P(x^k | Z^k, U^k, x_0)$$

Figura 5.5: Fórmula (Predição do próximo estado)(NIETO JOSE GUIVANT, 2003)

Esse filtro pode ser atualizado de forma constante para cada partícula no filtro. Isso implica em N Kalman filters para estimar os N landmarks:

$$P(m_i | x^k, Z^k, U^k, x_0)$$

Figura 5.6: Fórmula (Estimando os N landmarks)(NIETO JOSE GUIVANT, 2003)

Em (MONTEMERLO S. THRUN, 2002) foi mostrado que o filtro inteiro pode ser atualizado em tempo logarítmico em função dos N landmarks. Claro que isso também é conseguido

através de outras abordagens, porém o diferencial do FastSLAM é fazer isso considerando modelos não-lineares de movimentação, além de distribuições não-gaussianas para cálculo de erros nos sensores e atuadores.

5.3 Os passos do FastSLAM deste trabalho

O FastSLAM implementado tem a seguinte ordem de execução:

Passo 1 - São computados os valores necessários para o agente poder se movimentar e se localizar, ou seja, conjunto de waypoints, conjunto de landmarks, posição atual do agente no mapa e o sua respectiva orientação, representada por um ângulo.

Passo 2 - Na seqüência são adicionados os ruídos de controle.

Passo 3 – Predição da posição posterior do robô. 9.3 – Se houver novos marcos, estes serão atualizados neste passo.

$$\mathbf{x}_{v_k} = \mathbf{f}_v\left(\mathbf{x}_{v_{k-1}}, \mathbf{u}_k\right) = \begin{bmatrix} x_{v_{k-1}} + V_k \Delta T \cos(\phi_{v_{k-1}} + \gamma_k) \\ y_{v_{k-1}} + V_k \Delta T \sin(\phi_{v_{k-1}} + \gamma_k) \\ \phi_{v_{k-1}} + \frac{V_k \Delta T}{B} \sin(\gamma_k) \end{bmatrix}$$

Figura 5.7: Fórmula (predição)(BAILEY, 2006)

Passo 3 - Predição da posição posterior do robô.

Passo 4 - V (velocidade do agente) e G (ângulo do agente) são acrescidos de erros de controle, calculados usando covariância.

Passo 5 - Predição do mapa no momento seguinte. Fórmula do passo 3 aplicado à cada marco que compõe o mapa.

Passo 6 - As observações dos sensores do robô são coletadas.

Passo 7 - São adicionados erros de observação.

Passo 8 - É calculada a associação de dados dos marcos coletados.

Passo 9 - É feita atualização do mapa:

9.1 - São computados os pesos dos marcos. Cada peso representa a confiança, ou a importância, deste estado para a criação do mapa.

9.2 - Neste ponto são atualizadas os marcos. Para cada umu é calculado o ganho de Kalman. Neste passo o agente cria o mapa com essas informações.

9.3 - Se houver novos marcos, estes serão atualizados neste passo.

5.4 Trechos de códigos e devidas explicações

Como foi explicado, a abordagem FastSLAM foi escolhida para resolver o problema do SLAM para o controlador que está sendo criado para a Virtual Robots Competition. Porém, foi adicionada a característica de caminho decidido, ou seja, alguns pontos serão passados em cada atualização para o robô, de forma que se consiga fazer com que o agente passeie por todo o mapa, descobrindo todo o ambiente. As coordenadas passadas ao agente foram chamadas por convenção de waypoints e, a menos que haja obstáculos, esses são os vetores (x,y) pelo qual o robô deve passar. Antes de tudo, é necessário explicar que a classe DMatrix (BECERRA, 2008) está sendo usada para manipular matrizes, pois esta classe possui várias fórmulas aplicáveis à elas. Dentro da divisão do grupo formado pelas universidades, fiquei responsável por criar os algoritmos do SLAM sem preocupar-me com a ordem de execução deles. A ordem e a parte gráfica do SLAM ficaram para outros membros da equipe, portanto, será explicada a ordem em que são chamadas as implementações que fiz, porém, pode ser que, por decisão do grupo, a ordem se modifique. O primeiro passo do processo é computar os valores de posicionamento, ou seja, conjunto de waypoints, conjunto de landmarks, posição atual do robô e o seu respectivo ângulo. Na seqüência são adicionados os ruídos de controle:

Acima (figura 5.8) temos G que é o ângulo que indica o direcionamento do agente, V representa a velocidade do agente, que, junto com G acumula os erros de controle e Q, denota a matriz de covariância. O algoritmo acima funciona da seguinte maneira, primeiramente, computa os valores atuais tanto de V quanto de G na matriz VG, que representa os erros de controle. Na seqüência verifica-se a necessidade, nesta iteração, de adicionar erros de controle (isso é decidido na chamada do método), caso seja, o algoritmo de multivariação gaussiana calculará o erro no controle. Os erros são atualizados nas variáveis V e G, e estas por sua vez são retornadas em forma de matriz de erros de controle. Para facilitar o entendimento, vejamos abaixo, na figura 5.9, o código da multivariação gaussiana:

```
DMatrix FastSlam::add_control_noise(double V, double G, DMatrix Q, double addnoise)
{
        // Add random noise to nominal control values
        DMatrix C = zeros(2,1);
        DMatrix VG = zeros(2,1);
        VG(1) = V;
        VG(2) = G;
        if(addnoise == 1)
        {
                C = multivariate_gauss(VG , Q, 1); // if Q might be correlated
                V= C(1);
                G= C(2);
        }
        VG(1) = V;
        VG(2) = G;
        return VG;
}
```

Figura 5.8: Trecho de código (addControlNoise)

```
DMatrix FastSlam::multivariate_gauss(DMatrix x, DMatrix P, int n)
{
        // INPUTS:
        //   (x, P) mean vector and covariance matrix
        //   obtain n samples
        // OUTPUT:
        //   sample set
        DMatrix X = tra(P);
        DMatrix Si = Chol(X);
        DMatrix S = zeros(2,2);
        S(1,1) = Si(1,3);
        S(2,2) = Si(2,3);
        X = randn(2,1);
        DMatrix s = S*X + x*ones(1,n);
        return s;
}
```

Figura 5.9: Trecho de código (multivariateGauss)

Neste trecho de código (figura 5.9) a matriz de covariância é transposta e, em seguida é fatorada usando-se o método de Cholesky. Depois é multiplicada por uma matriz gerada de forma randômica, usando-se uma distribuição normal com média zero e variância unitária . Resumindo, este método devolve uma amostra aleatória de distribuição Gaussiana multivariada.

Na seqüência do FastSLAM, o próximo método a ser chamado seria o de predição:

```
FastSlam::Particle FastSlam::predict(Particle particle, double V, double G,
DMatrix Q, double WB, double dt, int addrandom)
{
.       // add random noise to controls
.       DMatrix VG = zeros(2,1);
        DMatrix temp = zeros(2,1);
.       temp(1,1) = V;
.       temp(2,1) = G;
.       if(addrandom == 1)
.       {
.       .       VG = multivariate_gauss(temp , Q, 1);
.       .       V = VG(1,1);
.       .       G = VG(2,1);
.       }
.       // predict state
.       DMatrix xv;
.       xv = particle.xv;
.       particle.xv(1) = xv(1,1) + V*dt*cos(G+xv(3,1));
.       particle.xv(2) = xv(2,1) + V*dt*sin(G+xv(3,1));
.       particle.xv(3) = pi_to_pi(xv(3) + V*dt*sin(G)/WB);
.       return particle;
}
```

Figura 5.10: Trecho de código (predict)

Neste método, expresso na figura 5.10, fica a lógica da predição do próximo estado, então é feita a adição de ruídos de controle, ou seja será atualizado o valor em V e em G. Isso ocorre para cada partícula. As partículas representam os estados do agente em todo o decorrer do algoritmo.

No próximo passo do FastSLAM, são buscadas as observações do agente:

```
FastSlam::DMatrixArray FastSlam::get_observations(DMatrix x, DMatrix lm, DMatrix idf, double rmax)
{
.       DMatrixArray observations;
        // [lm,idf]
        observations = get_visible_landmarks(x , lm, idf, rmax);
        lm = observations.array[0];
        DMatrix z = zeros(2,2);
        z = compute_range_bearing(x,lm);
        observations.array[0] = z;
        return observations;
}
```

Figura 5.11: Trecho de código (getObservations)

Dando seqüência ao FastSLAM, adiciona-se ruídos nas leituras das observações. Então são calculadas as associações de dados. Pelo tamanho do código, para este caso, apenas irei citar o que é feito. Primeiramente são buscadas as associações e novas características, essas novas características são adicionadas ao mapa e a tabela de associação de dados é devolvida.

No momento seguinte, se for encontrado algum landmark, são computados, através do método compute_weight, os pesos relativos à cada landmark. Pesos esses que garantem o nível de confiança de cada posição do robô. Além disso, o método feature_update é chamado. O primeiro passo deste método é calcular a matriz jacobiana, que como foi explicado anteriormente, é usada no filtro de Kalman. Veja abaixo, na figura 5.12, alguns trechos do código do método feature_update:

```
FastSlam::Particle FastSlam::feature_update(Particle particle, DMatrix z, DMatrix idf, DMatrix R)
{
.       .
.       .compute\underline{\space}weight
.       .
.       // z - zp
        v = z-(compJacobians.array[0]);
.       .
.       .
.       .
.       DMatrixArray xfPf = KF_cholesky_update(xf1,Pf1, vaux, R, Hfaux);
        xf1 = xfPf.array[0];
        Pf1 = xfPf.array[1];
.       .
.       .
.       .
.       xfPf = KF_cholesky_update(xf2, Pf1, vaux, R, Hfaux);
        xf2 = xfPf.array[0];
        Pf2 = xfPf.array[1];
        Pf = Pf1 && Pf2;
        xf = xf1 && xf2;
        particle.xf = xf;
        particle.pf = Pf;
        return particle;
 }
```

Figura 5.12: Trecho de código (featureUpdate)

Neste trecho podemos ver que o valor verdadeiro da posição passada como parâmetro (z) é subtraído do valor que foi predito, esse valor é usado para recalcular as variáveis que serão utilizadas no filtro de Kalman estendido. O método KF_cholesky_update é quem faz a simulação de Monte Carlo. Ele é utilizado para encontrar a matriz triangular inferior da matriz de covariância P. É aplicado à P um vetor de choques não co-relacionados, chamado aqui de u, que produz um vetor de choques chamado de Lu. Esse vetor possui as propriedades de covariância do sistema modelado. O método KF_cholesky_update pode ser visto abaixo, na figura 5.13:

```
FastSlam::DMatrixArray FastSlam::KF_cholesky_update(DMatrix x, DMatrix P,
DMatrix v,DMatrix R, DMatrix H)
{
        DMatrix PHt,S,SCholAux;
        PHt = P*tra(H);
        S = H*PHt + R;
        S = (S+tra(S))*0.5; // make symmetric
        SCholAux = Chol(S);
        DMatrix SChol = zeros(2,2);
        SChol(1,1) = SCholAux(1,3);
        SChol(1,2) = SCholAux(2,1);
        SChol(2,1) = 0;
        SChol(2,2) = SCholAux(2,3);
        DMatrix SCholInv = zeros(2,2);
        SCholInv = inv(SChol); // Triangular matrix
        DMatrix W1 = zeros(2,2);
        W1 =  PHt * SCholInv;
        DMatrix W = zeros(2,2);
        W = W1 * tra(SCholInv);
        x = x + W*v; // Update
        P = P - W1*tra(W1);
        DMatrixArray cholesky;
        cholesky.array = new DMatrix[2];;
        cholesky.array[0] = x;
        cholesky.array[1] = P;
        return  cholesky;
}
```

Figura 5.13: Trecho de código (KFCholeskyUpdate)

Este método calcula o EKF considerando o estado anterior (x e P) , a inovação (v, R) e o modelo de observação (H) devidamente linearizado.

Dando seqüência ao FastSLAM, caso seja encontrado um novo landmark, este landmark é adicionado à matriz de partículas.

E assim se reinicia o ciclo, que só é finalizado quando acabam os waypoints ou quando se esgota o tempo.

6 *Conclusões*

A criação dos algoritmos do FastSLAM permitiu atingir parcialmente os objetos deste trabalho. A parcialidade é devida à dependência do desempenho obtido pelas demais universidades envolvidas neste processo. Porém, o que foi delegado para a equipe da Universidade Federal de Santa Catarina foi atingido em sua completude. Para que tudo que se previu seja alcançado falta apenas à integração com os demais módulos do projeto. O primeiro momento neste projeto foi descobrir como funcionava a Virtual Robots Competition. Saber o que poderia ser usado para o mapeamento do mundo, quais controladores estavam disponíveis, como interagir com o ambiente e com os agentes nele inseridos. Para se atingir o objetivo da criação do algoritmo de FastSLAM foi necessário passar por sua base, que consistia nos primeiros algoritmos de SLAM, entender as abordagens mais comuns e reaprender sobre alguns conceitos da matemática. O passo seguinte, foi escolher, de forma empírica, aquela que poderia ser a melhor abordagem, considerando-se o ambiente proposto pela Virtual Robots Competition. E por fim houveram definições de como implementar a abordagem e como fazer a integração com o resto do sistema. Acredita-se que este trabalho terá relevância para a academia por se tratar de um assunto atual e de interesse à comunidade científica e aos profissionais de Ciências da Computação, automação e Inteligência artificial. Pois nele são apresentados conceitos que podem servir de base para demais trabalhos usando técnicas de mapeamento e localização simultâneos.

6.1 Trabalhos futuros

Como este é um trabalho evolutivo, alguns dos trabalhos que poderiam ajudar neste processo seriam:

- Criar as chamadas e gráficos para testar visualmente o projeto;
- Testar em autômatos reais;
- Participar de competições vínculadas à Virtual Robots para testar e melhorar os algoritmos.

Referências Bibliográficas

BAILEY, T. *SLAM*. 2006. Disponível em: <http://sourceforge.net/project/showfiles.php?groupi d = 145394>.

BECERRA, V. M. *DMATRIX: A C++ Matrix Class*. 2008.

ELFES, A. *Sonar-based real-world mapping and navigation*. 1987.

MOAST. Disponível em: <http://sourceforge.net/projects/moast/>.

MONTEMERLO S. THRUN, D. K. e. B. W. M. *FastSlam:A Factored Solution to the Simultaneous Localization and Mapping Problem*. 2002.

MORAVEC, H. *Sensor fusion in certainty grids for mobile robots*. 1988.

NIETO JOSE GUIVANT, E. N. e. S. T. J. *Real Time Data Association for FastSLAM*. 2003.

PFINGSTHORN, B. S. M. *ManifoldSLAM : a Multi-Agent Simultaneous Localization and Mapping System for the RoboCup Rescue Virtual Robots Competition*. 2006. Disponível em: <http://www.springerlink.com/content/c554763j202202um>.

ROBOCUP. 1998. Disponível em: <http://www.robocup.org/>.

ROBôS USARSim. Disponível em: <http://sourceforge.net/project/showfiles.php?groupi d = 145394>.

RULES of the Virtual Robots Competition. 2008. Disponível em: <http://www.robocuprescue.org/wiki/images/Rules2007V5.pdf>.

UNREAL Tournament. Disponível em: <http://www.unrealtournament.com>.

USARSIM. Disponível em: <http://sourceforge.net/projects/usarsim/>.

VIRTUAL Robots Competition. 2006. Disponível em: <http://www.robocuprescue.org/rescuerobots.html>.

VIRTUAL Robots Competitions Rules Figura SW/HW configuration. Disponível em: <http://www.robocuprescue.org/wiki/images/Rules2007V5.pdf>.