Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

ARISTÓTELES TERCEIRO NETO

FILTRO DE KALMAN ESTENDIDO UTILIZADO PARA LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEO

Campina Grande, Paraíba.
Agosto de 2012

ARISTÓTELES TERCEIRO NETO

FILTRO DE KALMAN ESTENDIDO UTILIZADO PARA LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Robótica

Orientador:
Professor Bruno Barbosa

Campina Grande, Paraíba.
Agosto de 2012

ARISTÓTELES TERCEIRO NETO

FILTRO DE KALMAN ESTENDIDO UTILIZADO PARA LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEO

Trabalho de Conclusão de Curso submetido à
Unidade Acadêmica de Engenharia Elétrica da
Universidade Federal de Campina Grande como
parte dos requisitos necessários para a obtenção do
grau de Bacharel em Ciências no Domínio da
Engenharia Elétrica.

Área de Concentração: Robótica

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Bruno Barbosa.
Universidade Federal de Campina Grande
Orientador, UFCG

Agradecimentos

Queria agradecer a Deus, por sempre me iluminar nos momentos bons e ruins da minha vida.

A minha antiga equipe do eROBOTICA, principalmente a Débora e Thiago de Freitas.

A minha família, Ana Maria, Carlos Antonio e Carlos Antônio Junior, por me apoiarem e me ajudarem com todo seu amor e carinho.

A minha namorada Mariana Costa, que há muito tempo está do meu lado dando seu amor, carinho e contribuindo muito para realização deste trabalho.

Ao professor Bruno Barbosa, por me orientar e acreditar no meu desempenho.

E a todos meus amigos e professores, que, direta ou indiretamente, ajudaram-me a concluir meu curso e meu trabalho.

RESUMO

Localização e mapeamento simultâneos é uma técnica utilizada por robôs e veículos autônomos para construir um mapa de um ambiente ao tempo em que se localiza. O problema SLAM geral tem sido objeto de investigação substancial pela comunidade de pesquisa em robótica. Uma série de abordagens têm sido propostas para resolver tanto o problema SLAM e também problemas de navegação mais simplificados onde mapas adicionais ou informações de localização do veículo é disponibilizado. Em termos gerais, estas abordagens adotam uma das três principais filosofias. O mais popular deles é a estimativa de teoria ou abordagem baseada no filtro de Kalman. O EKF-SLAM é uma ferramenta bastante estudada, e nesse trabalho será apresentado uma introdução sobre esse tema.

Palavras-chave: SLAM, Filtro Estocástico, TCC, Kalman, Robótica.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AGV - Veiculo Autonomo Guiado

GPS – Sistema de Posicionamento Global

SLAM – Mapeamento e Localização Simultâneos

DLSDS - discrete-time linear stochastic dynamic system

KF – Filtro de Kalman

EKF- Filtro de Kalman Estendido

AMF – Filtro do Mapa

1 INTRODUÇÃO

O campo da robótica tem realizado significativos progressos nos últimos anos. A utilização da robótica móvel se dá praticamente em todas as áreas imagináveis, que vão desde a exploração submarina, exploração do espaço para tarefas de transportes e domesticas, ao poucos os robôs estão substituindo os seres humanos nestes domínios. No passado, os robôs eram utilizados, geralmente, limitados a linhas de montagens e de utilização industrial onde eram trabalhos exaustivamente repetitivos. No entanto, como consequência de recentes pesquisas, vários sistemas robóticos bem sucedidos foram mobilizados para realizarem uma variedade de tarefas, muito mais desafiadores do que antes. O desafio nessas tarefas é devido principalmente à incerteza do ambiente, que torna um sistema determinístico inútil. A incerteza em um ambiente pode ser devido a uma série de razões diferentes: o ruído, as limitações dos sensores e as limitações de modelar o mundo. Essa incerteza, por muito tempo tem sido um dos principais gargalos para a construção de sistemas robustos em todos os domínios. Especificamente, resolver o problema da incerteza tem sido a chave que falta para realizar as tarefas de navegação autônoma, que é um pré-requisito para a aplicação de robótica para toda e qualquer área.

A navegação autônoma proporciona a habilidade de um robô móvel navegar por um ambiente sem ajuda externa, isso é considerado o santo gral da robótica. Os primeiros sistemas robóticos contavam com a sinalização externa, como fitas, sinalizadores ou operadores humanos para localizar-se. Como estes sistemas são inerentemente determinista na natureza, eles permanecem inflexíveis pelas restrições impostas pelo determinismo. Assim, um sistema determinístico não pode operar em um ambiente de incertezas. Para serem capaz de funcionar em ambientes realistas, os robôs precisam ser capazes de representar a informação probabilisticamente. Esta realização liderou a transição da robótica determinística para robótica probabilística em meados da década de 1990. Simplificando, um robô navegando de forma autônoma não tem uma estimativa única da sua localização em qualquer ponto no tempo. Pelo contrário, ele representa sua estimativa de posição como uma distribuição de probabilidade. Dado um modelo do ambiente e da capacidade para estimar a sua localização, qualquer ponto no mapa pode ser atravessada por um robô móvel. Assim, um mapa *a priori* é exigido por um robô para ser capaz de se mover de forma autônoma dado um destino. Este é um mapa *a priori* é usado de duas maneiras. Este mapa é usado para planejar o caminho do robô, evitando obstáculos enquanto o robô também corrige estimativa posição com respeito ao mapa (localização). Nessas situações no qual

possuímos um mapa *a priori* torna-se a tarefa trivial, o problema de navegação autônoma pode ser reduzido a um problema de planejamento de localização e de caminho. No entanto, a navegação autônoma por meio de um mapa estático em condições reais pode não ser viável. Essa inadequação pode ser resolvida de forma autônoma, se um robô é capaz de construir o próprio mapa [7].

Inicialmente, o estudo de mapeamento e localização foram feitos de formas desassociadas onde um não afetaria o desempenho do outro. A correlação entre a posição estimada do robô e a localização estimada dos pontos de referencias, é onde surge o erro de estimativa da posição do robô, deste modo foi identificado como chave para resolver o problema de construção do mapa. Em outras palavras, a construção de um mapa preciso requer que o robô tenha uma grande precisão na estimativa de sua posição e uma precisa estimativa da posição do robô requer um mapa preciso. Deste modo, o problema de construção do mapa é tratado conjuntamente com o problema de localização do robô. Numa situação utópica onde a posição do robô é sempre conhecida, a construção do mapa é reduzida a uma tarefa de transformar modelos observados do ambiente em uma estrutura global unificada. No entanto, a posição do robô raramente é conhecida com precisão, devido aos erros de medição inerente aos sensores. Do mesmo modo, se o robô pode utilizar o modelo do ambiente com precisão, pode então localizar-se pela utilização de uma simples geometria. É necessário um bom mapa para localização e uma estimativa precisa de posição, onde estes dados serão necessários para a construção de um mapa. Como estas tarefas não pode ser dissociados um do outro, SLAM (Localização e mapeamento simultâneos) é muitas vezes referida como um problema da galinha ou o ovo.

A falta de sensores exteroceptivos precisos e de um sensor de odometria preciso, tornam a tarefa de construir um mapa bastante difícil. Assim, mapeamento e localização simultâneos é basicamente a tarefa de estimar a posição do robô e ter uma estimativa do modelo do ambiente a partir de estimativas erradas, tanto da posição do robô, quanto do modelo de ambiente.

1.1 MOTIVAÇÃO

Um dos requisitos fundamentais requeridos para existência de um verdadeiro agente automatizado é a habilidade de navegar em ambientes. A navegação é uma ciência que interfere no curso de um veiculo baseado nas informações de uma variedade de sensores. Por exemplo, navegadores de embarcações contam com as estrelas para poderem se localizar em mares desconhecidos. Esta forma de navegação depende de um mapa conhecido dos corpos celestes e

é análogo a muitos esquemas de localização baseada em mapeamento que são relevantes no ramo de robôs autônomos.

Localização tem sido uma exigência primária em vários sistemas robóticos moveis desde o inicio da robótica móvel. Inicialmente, durante a década de 50, os robôs eram todos guiados por linhas, onde eram chamados de AGVs (*Autonomous Guided Vehicles*). Os sensores dos AGVs reconheciam o caminho através de frequência de rádio e seguiam a trajetória. Estes sistemas de localização tinham seus caminhos a percorrer fixados de antemão. Enquanto isso, entre a década de 60 e 70, os veículos desenvolvidos pela Universidade de Stanford foram utilizados para estudar o controle do Rover na lua. O Rover pode ser visto na Figura 1.

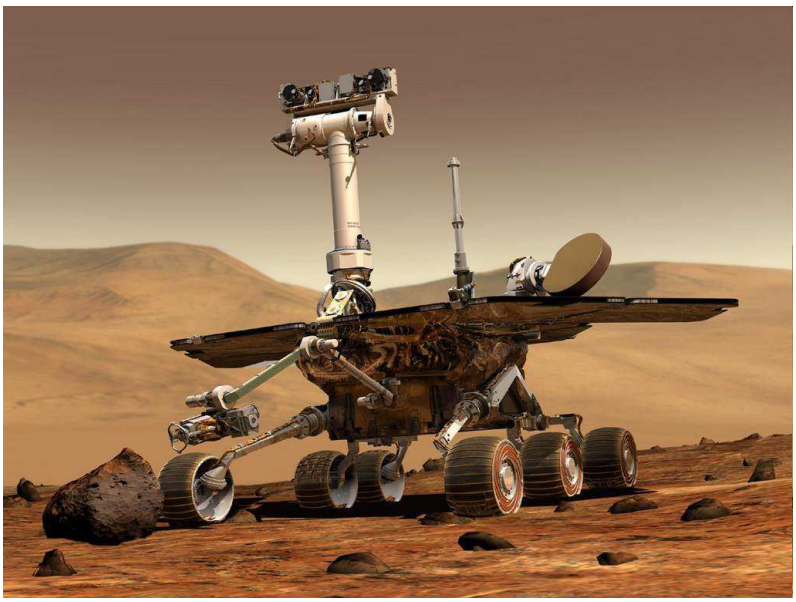

Figura 1 - Robô Rover enviado para Marte.

Então na década de 70, um simples sistema era desenvolvido, onde os robôs móveis (também chamados de AGVs) usavam fitas magnéticas ou fitas coloridas com um sensor apropriado para seguir o caminho traçado pelas fitas. Os AGVs tinham como vantagem a fácil construção do seu sistema e o este sistema podia ser realocado e mudado para onde quiser sem muita dor de cabeça. Entretanto, as habilidades dos robôs de seguirem uma linha eram bastante limitadas e impossibilitava a construção de um sistema mais robusto e eficiente.

Um sistema simples baseado em odometria, também chamado de *Dead Recoking*, foi desenvolvido. Esse sistema calculava a rotação das rodas ou então, através de um sistema de navegação inercial, media a força experimentada durante a aceleração e verificava se não sofria falhas no movimento seguinte. Entretanto, esse sistema sofria por um erro incremental o qual gerava erros de medição. Os primeiros sistemas confiáveis de localização eram baseados em sinalizadores. Sistemas baseados em sinalizadores tinham alguns sinalizadores instalados em posições conhecidas em um ambiente e o agente equipado com sensores podia observar esses

sinalizadores. Esse método não tinha um crescimento acumulado dos erros, já que o robô tinha conhecimento prévio e preciso da localização dos sinalizadores. Porém ocorria um problema que a implantação e manutenção dos sinalizadores no ambiente eram um processo bastante caro. O GPS (*Global Positioning System*) usa um sistema baseado em sinalizadores ativos. Um sistema baseado em GPS recebe sinais provenientes de satélites, onde esses sinais contêm selos de tempo e um identificador único para cada satélite. Conhecida a posição do satélite e o tempo de envio do sinal obtido através do tempo de selo, a localização do AGV era estimada. Um dos grandes problemas na utilização de um sistema baseado em GPS é a falta de precisão da posição informada e não funciona em ambientes internos ou sem sinal.

O rápido crescimento da tecnologia tem exercido uma grande melhora nos sensores, e também nas funcionalidades e interfaces provenientes para o controle do robô. Com o advento das novas tecnologias, incluindo a fabricação de sensores mais baratos e um grande aumento na velocidade computacional, tem havido um grande esforço para aumentar o nível de autonomia dos agentes remotos e reduzir a intervenção humana para a mínima possível. Os métodos de localização utilizados atualmente não dependem de sinalizadores, mas estimar uma posição comparando o modelo atual fornecido do ambiente extraído através de um sensor com o mapa do ambiente a priori. Essa é uma melhoria bastante significativa, tal como elimina a necessidade de instalar sinalizadores nos ambientes, e é mais confiável a utilização de modelos naturais do ambiente que já agem como sinalizadores. Esses modelos que são utilizados para estimar a posição do robô são chamados de características e recuperar esses modelos de dados do sensor é chamado de extração de características.

Os exemplos acima mencionados de localização partem de um pré-requisito comum: um mapa priori. Um mapa priori pode ser fornecido de várias maneiras para veículos terrestres autônomos. As fitas em linhas/tiras/cores atuam como linhas para robôs seguirem no mapa, enquanto as constantes coordenadas dos sinalizadores agem como um mapa para sistemas baseados em sinalizadores. Um mapa priori também pode ser construído através da teleoperação do robô (onde um humano opera o robô) e comparando as observações de forma iterativa. Essa abordagem, no entanto, sofre por um sério problema. Para os AGVs utilizarem características naturais, é necessário que os mapas a priori sejam fornecidos através de medições manuais, logo, como consequência da utilização deste mapa, a mobilidade do robô mantém-se limitada ao meio conhecido. Ou seja, para cada ambiente em que o robô precisar navegar será necessário ser entregue um novo mapa. Além disso, qualquer mudança no ambiente deve refletir no mapa dado ao robô. Assim, qualquer mapa desenvolvido com a intervenção humana será obrigado de uma assistência humana para contemplar as alterações sofridas pelo ambiente. Para superar este inconveniente, os pesquisadores têm trabalhado sobre a possibilidade de o robô autônomo construir seu próprio mapa e usá-lo para localizar-se. Isto conduziu ao surgimento do popular

problema de Simultânea localização e mapeamento. Considerado o santo graal da robótica, o problema de SLAM é considerado como a única lacuna para obtenção da navegação autônoma [2]. Fazer mapeamento não é trivial, já que um robô tem de saber sua localização para ser capaz de navegar em um mapa, e a localização só pode ser determinada utilizando as características de um ambiente. Muitas vezes em paralelo são traçados entre este problema e o da galinha e o ovo, que pergunta qual dos dois veio primeiro a existir, como o problema do SLAM é uma questão de o que fazer primeiro. A tarefa de construir o mapa é trivial dado a informação precisa da posição. Dada uma estimativa de posição e de uma observação do meio do ambiente, a observação pode ser fundida no mapa existente, para criar uma estimativa atualizada do mapa. Muitas abordagens baseadas em rede dependem deste tipo de construção de mapa para gerar um mapa de ocupação do ambiente [3,4]. Esses mapas baseados em características geralmente codificam uma relação entre diferentes características do ambiente. O mapeamento apresentado nesse trabalho produz mapas de características baseadas no EFK-SLAM, que são mapas métricos de partes que possuem identificação única no ambiente.

A implementação de sistemas baseados em SLAM possibilita a redução do esforço humano de forma continua em torno de áreas sensíveis e de até reportar sobre áreas que possuem altos níveis de radioatividade.

1.2 DEFINIÇÃO DO PROBLEMA E OBJETIVO

Sistemas complexos reais são difíceis de modelar, estão sujeitos a falhas e, na situação em que vidas humanas ou muito dinheiro ou ambos estão em jogo, precisam satisfazer critérios rigorosos de confiabilidade. Além disso, muitos resultados notáveis do campo da teoria do controle mostram-se fabulosos quando simulados em computadores de última geração, mas são absolutamente infactíveis quando se está tratando de um processador embarcado em um satélite ou um robô móvel com restrições sérias de espaço e energia. Ironicamente, estes sistemas operando sob-restrições são muitas vezes aqueles que precisam lidar com as caracteristicas não lineares, a saber, incertezas de modelo, corrupção de medidas sensoriais, falhas de instrumentação e estruturais, requisitos de confiabilidade em situações extremas, etc. [5]

Motivado pelo problema do estudo da localização e mapeamento simultâneo para ambientes internos, esse trabalhou visou a descrição do desenvolvimento de um algoritmo básico para sistemas de localização robóticos. Devido o grande número de erros que os sensores ocasionam, foi necessário a implementação de um filtro estocástico não-linear, como esta descrito logo adiante, o filtro de Kalman estendido. Logo, no decorrer do trabalho será descrito o problema do SLAM em geral e como funciona esta ferramenta.

2 LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS

O problema de localização e mapeamento simultâneo (SLAM) pergunta[7] se é possível um robô móvel ser colocado em um ambiente com a posição desconhecida e uma localização desconhecida e o robô incrementalmente construir um mapa consistente do ambiente enquanto determina sua localização nesse mapa. A solução do problema de SLAM é considera o "Santo Graal" para a comunidade de robótica móvel e pode prover o verdadeiro robô móvel autônomo. A Figura 2 mostra como funciona a problema de localização e mapeamento no simultâneo.

A solução do problema de SLAM tem sido um dos sucessos notáveis da comunidade de robótica durante a última década. O SLAM foi formulado e resolvido como um problema teórico em uma variedade de formas diferentes. SLAM também foi implementado em vários ramos da robótica, como por exemplo, para sistemas em ambientes ao ar-livre, ambientes fechados, debaixo de água e sistemas aéreos. Em nível teórico e conceptual, o SLAM já pode ser considerado um problema quase resolvido.

A localização e mapeamento simultâneo (SLAM) é um processo de construção de um mapa de um ambiente e ao mesmo tempo utilizando o mapa para estimar a localização do robô móvel. O robô depende principalmente da capacidade dos seus sensores para extrair informações úteis para navegação. Tipicamente, o robô começa em um local desconhecido sem nenhuma informação a priori do meio ambiente. Utilizando as observações relativas das características obtidas através dos sensores, uma estimativa da posição do robô e das coordenadas das características é simultaneamente obtida. Atravessando o ambiente, o veiculo constrói um mapa completamente baseado em características e utilizando este mapa para prover a melhor posição do robô. É o rastreamento da posição relativa das características através do robô que nos leva a simultânea estimação da característica e posição. O âmbito de utilização de tal sistema de navegação autônoma é enorme.

Atualmente uma das ferramentas para resolução de problemas de sistemas estocásticos mais utilizados no mundo é o Filtro de Kalman. Logo, SLAM baseados no Filtro de Kalman ganhou bastante popularidade e imergiu um grande interesse da comunidade cientifica. Ao desenvolver os estudos percebe-se a necessidade de um método para resolução de problemas de sistemas estocásticos não lineares, então uma alternativa proposta para resolver este tipo de problema foi a utilização do Filtro de Kalman Estendido (EKF).

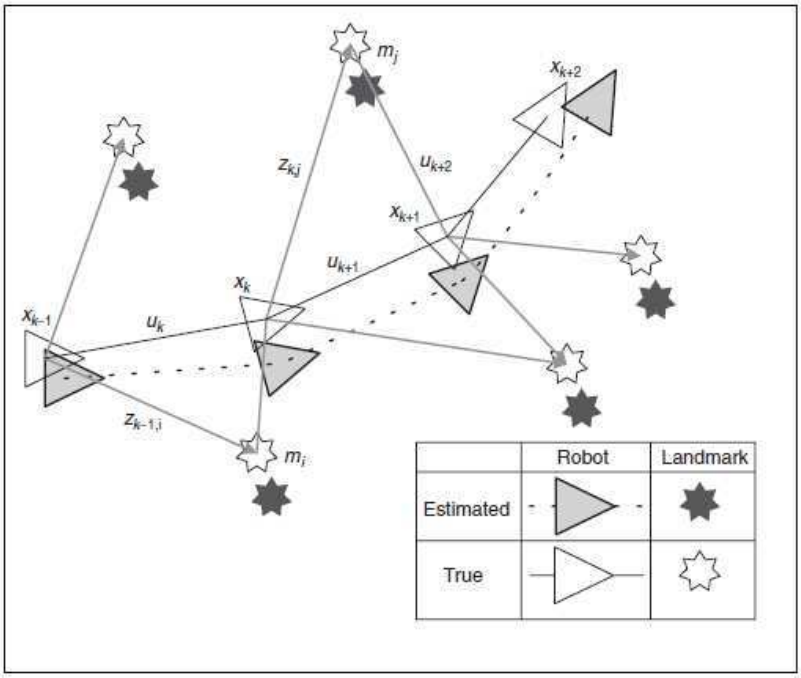

Figura 2 - Essência do Problema do SLAM, estimativas correlacionadas.

2.1 INTRODUÇÃO

O SLAM consiste em três passos básicos, que são reiterados em cada intervalo de tempo:

1. **O robô move**, atingindo um novo ponto de vista da cena. Devido ao ruído inevitável e aos erros de estimativa, este movimento aumenta a incerteza sobre à auto-localização. Qualquer solução automatizada requer um modelo matemático para este movimento. Nós chamamos este de modelo de movimento.
2. **O robô descobre características interessantes no ambiente**, que precisam ser incorporadas ao mapa. Chamamos esses marcos de características, como dito anteriormente. Devido a erros nos sensores, a localização destes pontos de referência será incerta. Além disso, como a posição do robô é incerta, as duas incertezas devem ser devidamente composta. Uma solução automatizada requer um modo de determinar a posição dos pontos de referência na cena a partir de dados obtidos pelos sensores. Chamamos a isto de modelo de observação inversa.
3. **O robô observa as referências que tinham sido previamente mapeadas**, e as usa para corrigir a sua auto-localização e a localização de todas as marcas de referências no espaço. Neste caso, portanto, tanto a localização e as referências tem diminuído a incerteza. Uma solução automatizada requer um modo para prever os valores de

medição a partir do local de referência previsto e a localização do robô. Chamamos isto de modelo de observação direta.

Com esses três modelos mais uma ferramenta para estimação, podemos construir uma solução automatizada para o problema de SLAM. O estimador é responsável pela propagação adequada da incerteza a cada vez que uma das três situações de acima ocorrer.

A abordagem do EKF-SLAM é prover um processo de filtragem para o sistema. Esse trabalho irá tratar de um SLAM bidimensional num ambiente tridimensional, o qual é feito usando o *scanner* e a odometria. Assim o mapa é representado como um mapa bidimensional. Isso também adiciona o pressuposto subjacente de ter obstáculos no ambiente que são ortogonais ao plano do solo. O sistema EKF-SLAM é semelhante à estimativa de um sistema linear dinâmico, salve algumas restrições que aumentam sua complexidade. Portanto, o sistema de SLAM é mostrado neste trabalho como uma extensão do algoritmo de estimativa genérico para a estimativa de um sistema de resposta linear.

2.2 ESTIMAÇÃO DE UM SISTEMA DINÂMICO

De acordo com [6], o problema de estimação de estados de sistemas dinâmicos estocásticos a partir de medidas ruidosas é um assunto de importância central no campo da engenharia. Registros de trabalhos nesta aérea datam de mais de dois séculos, quando Gauss propôs o método de estimação de mínimos quadrados (*least square estimation*) ao tentar estabelecer os parâmetros de órbitas por meio de observações de corpos celestes. Sistemas físicos (máquinas industriais, redes de transmissão de energia, aeronaves, robôs, aparelhos domésticos, etc.) são projetados e construídos para executar tarefas específicas e atender a objetivos. Para poder avaliar o desempenho de um sistema e, se necessário, tomar medidas corretivas, um controlador (humano ou não) precisa saber o que o sistema está "fazendo" a todo instante de tempo, ou seja, precisa saber o seu estado. Contudo, sistemas reais sofrem com perturbações do ambiente, podendo eles mesmos apresentar comportamentos aleatórios. Visando à determinação do estado de um sistema, pode-se valer das leituras de instrumentos que medem grandezas relacionadas a ele, mas também estas medidas são corrompidas por ruído. Tendo isto em vista, o problema de estimação de estados a partir de modelos e medidas incertos (ruidosos) é chamado de estimação ou filtragem.

Então pode-se entender do que é uma "Estimativa":

"Estimativa é o processo pelo qual se infere o valor de uma grandeza de interesse, x, através do processamento de dados, que é de algum modo dependente do x." [7]

A incerteza é o principal problema de estimação (e na robótica). Afinal, se não fosse por problemas de incerteza muitos desses problemas teriam uma solução algébrica simples - "dá-me as distâncias a três pontos conhecidos e é algebricamente determinado que a intersecção de três círculos vá definir a minha localização". Estaremos tentando encontrar respostas para as perguntas mais realistas, como "Eu tenho três medições (todos com incerteza diferente) para três pontos pesquisados (conhecidos aproximadamente) – qual é a sua melhor estimativa da minha localização?".

Pode-se observar na figura 3 um problema gráfico de filtragem estocástica.

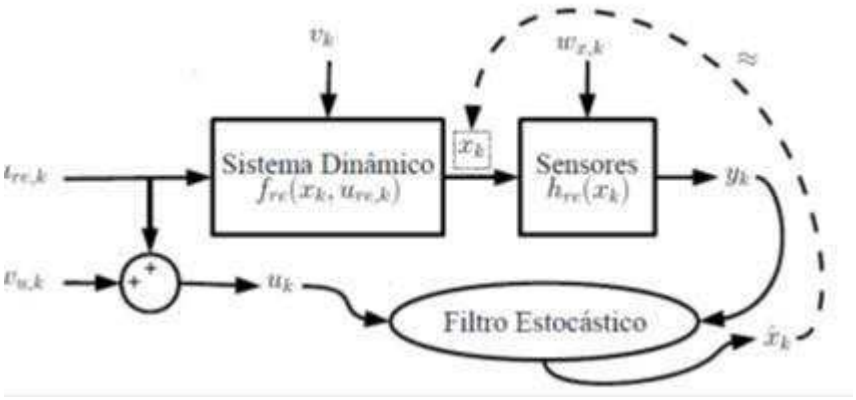

Figura 3 - Diagrama dos componentes envolvidos num filtro estocástico.

Idealmente, o objetivo do filtro estocástico é fornecer, a todo instante k, o valor real do estado x_k do sistema, em que x_k pode também envolver parâmetros desconhecidos do modelo matemático do sistema dinâmico. Para que isso fosse, de fato, possível, o filtro estocástico necessitaria ter acesso às entradas reais $u_{re,k}$ aplicadas ao sistema, juntamente com o modelo matemático real.

$$x_k = f_{re}(x_{k-1}, u_{re,k-1}), \quad \text{(E.1)}$$

$$y_k = h_{re}(x_k), \quad \text{(E.2)}$$

e medidas y_k dos sensores absolutamente livres de distúrbios, em que (E.1) descreve a maneira como o estado x_k evolui de acordo com grandezas do instante anterior e (E.2) é a função que relaciona as medidas dos sensores ao estado x_k atual do sistema. Contudo, esta situação jamais é verdadeira para sistemas dinâmicos reais. Sempre há imprecisões na definição da função de medição h_{re} e as saídas y_k dos sensores sempre são perturbadas por elementos de ruído, estando todos estes efeitos concentrados no termo de distúrbio w_k. Da mesma forma, as entradas u_k medidas são versões corrompidas por ruído $w_{u,k}$ das entradas reais $u_{re,k}$. Por fim, o termo v_k modela os desvios referentes à função f_{re} de evolução do processo, que nunca é totalmente conhecida. Como não se pode ter acesso a (E.1)-(E.2), o modelo matemático do sistema usado pelo filtro deve levar em conta as incertezas e ruídos explicitamente, dando origem a

$$x_k = f_{re}(x_{k-1}, u_{re,k-1}) + v_{k-1}, \quad \text{(E.3)}$$

$$y_k = h_{re}(x_k) + w_k, \quad \text{(E.4)}$$

Em que v_{k-1} e w_k são ruídos do processo branco Gaussiano tais que $w_{k-1} \sim N(0, Q_{k-1})$ e $v_k \sim N(0, R_k)$; e $u_{k-1} \sim N(\bar{u}_{k-1}, P_{u,k-1})$ é o vetor de entrada do sistema, ao qual também está associada uma perturbação Gaussiana $v_{u,k-1} \sim N(0, P_{u,k-1})$. Caso o sistema (E.1)-(E.2) seja linear, considera-se a forma

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + v_{k-1}, \quad \text{(E.5)}$$
$$y_k = C_k x_k + w_k, \quad \text{(E.6)}$$

Uma prática comum no modelamento de sistemas dinâmicos reais é concentrar toda a incerteza relacionada a (E.3) ou (E.5) no termo de ruído v_{k-1}. Nestas situações, faz-se $P_{u,k-1}$=0, ou seja, elimina-se toda a incerteza associada ao vetor de entradas u_{k-1}, que passa a ser considerado completamente conhecido. Esta formulação é particularmente útil quando é difícil caracterizar as perturbações sobre u_k ou nas situações em que não te se acesso a este vetor, passando-se a depender exclusivamente das medidas y_k e dos modelos matemáticos para estimação de estados. Nestas situações, as medidas $u_{1:k}$ podem ser eliminadas da análise.

Em posse de (E.3) - (E.4) ou (E.5) - (E.6) e de medidas de entrada $u_{1:L} = \{u_1, u_2, \dots, u_L\}$ e saída $y_{1:L} = \{y_1, y_2, \dots, y_L\}$ corrompidas por ruído, o problema de estimação discreta de estado consiste em procurar pela estimativa $\hat{x}_k$, que melhor aproxima $x_k$em algum sentido, visto que nunca é possível chegar a $x_k$diretamente. Se k < L, i.e., existem medidas de entrada e saída posteriores ao instante de filtragem k, o problema é chamado de suavização discreta (*discrete smoothing*). Agora, se k > L, está-se tentando estabelecer uma estimativa de estado $\hat{x}_k$,para um instante de tempo para o qual ainda não há medidas, sendo este o problema de predição discreta (*discrete prediction*). O último caso, k = L, é o problema de filtragem discreta (*discrete filtering*) e será doravante assumido.

Então, dado um estado de um sistema dinâmico estocástico linear discreto no tempo (DLSDS) no instante no tempo discreto l < k, e A_{k-1}, B_{k-1} são matrizes e o ruído v_k nos tempos 1, ... , k o sistema no instante k^{th}, pode ser escrito como:

$$x_{k-1} = \left[\prod_{j=0}^{k-l} A_{k-j-1}\right] x_l + \sum_{i=l}^{k} \left[\prod_{j=0}^{k-i-1} A_{k-j-1}\right] [B_{i-1}u_i + v_i] \quad \text{(E.7)}$$

O propósito do desenvolvimento de um modelo é para ser capaz de capturar a evolução do sistema com uma estrutura matemática simples. A propriedade de Markov simplifica o modelo do sistema significativamente.

2.2.1 PROPRIEDADE DE MARKOV

Considerando o sistema no instante l^{th} de tempo. O ruído do processo v_i, i = l,... , k são independentes do sistema x_l e todos os estados anteriores. Isto é, se

$$X^l \triangleq x_1, x_2, \dots, x_l = \{x_i\}_{i=0}^{l} \quad \text{(E.8)}$$

Temos que X^l depende somente do ruído gerado até o instante de tempo l, i.e, v_i onde i= 0,... , l – 1. Então, a função de distribuição de probabilidade para o estado do sistema depois do tempo k + 1, torna-se

$$p(x_{k+1}|X^l, U^k) = p(x_{k+1}|x_l, U_l^k) \; \forall \; k > l \quad \text{(E.9)}$$

onde $U_l^k \triangleq \{u_i\}_{i=l}^{l}$. A distribuição de probabilidade condicional dos estados futuros para o processo descrito acima, dado o estado presente e estados passados, depende somente do estado presente. O passado é irrelevante porque não importa como o estado atual foi obtido, apenas que o estado atual engloba todas as informações necessárias para estimar o próximo estado. Esta propriedade é chamada de propriedade de Markov e um processo com a propriedade é chamado de um processo de Markov.

Dado o estado atual de um DLSDS, a entrada de controle e o erro, todos os estados passados, então as entradas de controle e os erros de processo não são obrigatórios para prever o próximo estado. Esta propriedade é usada como a base para o modelo de evolução de tal sistema.

2.2.2 MÉDIA E PROPAGAÇÃO DA VARIÂNCIA

Para os DLSDS em consideração, as regras de controle da evolução dos estados o estado do robô é dado num instante de tempo. Isto torna obrigatório para modelar o sistema usando regras de sua evolução. O estado e a evolução do sistema de variância são discutidos a seguir, que é então seguido pelo desenvolvimento de um modelo para a sua estimativa.

Para um sistema dinâmico linear qualquer no passo de tempo discreto em k^{th},

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + v_{k-1} \quad \text{(E.10)}$$

onde a entrada de controle é fornecida e o ruídos v(k) é assumido como sendo a média nula da distribuição normal, o valor esperado do sistema

$$\hat{\mathrm{x}}_k \triangleq E[\mathrm{x}_k] \text{ (E.11)}$$

evoluindo de acordo com a seguinte equação

$$\hat{\mathrm{x}}_{k+1} = A_{k-1}\hat{\mathrm{x}}_k + \mathrm{B}_{\mathrm{k}-1}\mathrm{u}_\mathrm{k} \text{ (E.12)}$$

A matriz de covariância dos erros do estado é calculada aplicando o operador esperança a equação E.10

$$P_{xx,k} \triangleq E[[x_k - \hat{\mathrm{x}}_k][x_k - \hat{\mathrm{x}}_k]^T] \text{ (E.13)}$$

Subtraindo a equação E.12 dá equação E.10

$$\mathrm{x}_k - \hat{\mathrm{x}}_k = A_{k-1}\mathrm{x}_k + \mathrm{B}_{\mathrm{k}-1}\mathrm{u}_\mathrm{k} + \mathrm{v}_\mathrm{k} - (A_{k-1}\hat{\mathrm{x}}_k + \mathrm{B}_{\mathrm{k}-1}\mathrm{u}_\mathrm{k}) \text{ (E.14)}$$

o qual pode ser simplificado para

$$\mathrm{x}_k - \hat{\mathrm{x}}_k = A_{k-1}(\mathrm{x}_k - \hat{\mathrm{x}}_k) + \mathrm{v}_\mathrm{k} \text{ (E.15)}$$

então

$$P_{xx,k} \triangleq E[[A_{k-1}(\mathrm{x}_k - \hat{\mathrm{x}}_k) + \mathrm{v}_\mathrm{k}][A_{k-1}(\mathrm{x}_k - \hat{\mathrm{x}}_k) + \mathrm{v}_\mathrm{k}]^T] \text{ (E.16)}$$

o qual pode ser reduzido para

$$P_{xx,k} \triangleq E\left[\left[A_{k-1}(\mathrm{x}_k - \hat{\mathrm{x}}_k)(\mathrm{x}_k - \hat{\mathrm{x}}_k)^{'A_{k-1}'} + \mathrm{v}_\mathrm{k}(\mathrm{x}_k - \hat{\mathrm{x}}_k)^{'A_{k-1}'} + A_{k-1}(\mathrm{x}_k - \hat{\mathrm{x}}_k)\mathrm{v}_\mathrm{k}' + \mathrm{v}_\mathrm{k}\mathrm{v}_\mathrm{k}'\right]\right] \text{ (E.17)}$$

onde a covariância entre o estado e o vetor de ruído é zero. Usando a independência dos ruídos e do estado e assumindo que a variância do ruído v_k pode ser Q_k,a matriz de covariância do erro é encontrado e evolui de acordo com a equação.

$$P_{xx,k+1} = A_{k-1}P_{xx,k}A'_{k-1} + \mathrm{Q}_\mathrm{k} \text{ (E.18)}$$

2.3 FILTRO DE KALMAN

2.3.1 INTRODUÇÃO

Primeiramente proposto por Rudolf E. Kalman em seu trabalho seminal [8], o Filtro de Kalman é o estimador linear ótimo para sistemas lineares.

$$x_k = A_{k-1}x_{k-1} + B_{k-1}u_{k-1} + v_{k-1} \qquad \text{(E.5)}$$

$$y_k = C_k x_k + w_k, \qquad \text{(E.6)}$$

$w_{k-1} \sim N(0, Q_{k-1})$ e $v_k \sim N(0, R_k)$ são processos de ruído branco Gaussiano e descorrelacionados, ou seja,

$$E(w_i w_j) = \begin{cases} Q_i, i = j \\ 0, i \neq j \end{cases} = \begin{cases} R_i, i = j \\ 0, i \neq j \end{cases}, E(w_i v_j) = 0, \forall i, j \in \aleph,$$

e $u_{k-1} \sim N(\bar{u}_{k-1}, P_{u,k-1})$ é o vetor de entrada do sistema ao qual também está associada uma incerteza. Caso esta incerteza seja nula, basta fazer $P_{u,k-1}$=0. De acordo com [9], mesmo que w_{k-1} e v_k não sejam Gaussianos, o Filtro de Kalman ainda é o melhor filtro linear para (A.5)-(A.6), ainda que possa existir um estimador não-linear com melhor desempenho.

Seguindo o procedimento adotado no artigo original de Kalman [8], Jazwinski demonstra no Capítulo 7 de [6] que o Filtro de Kalman é o estimador linear ótimo de (E.5)-(E.6) por meio do método de projeções ortogonais. Simon, por outro lado, chega ao mesmo resultado no Capítulo 5 de [9], onde determina o estimador linear que minimiza o traço da matriz de covariâncias do erro de estimação. Nesta última demonstração, no entanto, o autor considera a forma do Filtro de Kalman conhecida, atendo-se apenas à determinação do ganho de Kalman ótimo. A exposição mais detalhada a respeito do Filtro de Kalman se justifica, principalmente, pelo fato do Filtro de Kalman Estendido, ser uma extensão linear de primeira ordem do Filtro de Kalman para sistemas não-lineares.

2.3.2 ALGORITMO DO FILTRO DE KALMAN

Partindo de uma estimativa inicial $\hat{\mathrm{x}}_0$ e $\hat{\mathrm{p}}_0$, em que x_0 é suposta uma variável aleatória Gaussiana, o Filtro de Kalman utiliza o modelo de evolução (E.5) para gerar estimativas preditas de $\bar{\mathrm{x}}_k$ até a chegada de uma nova medida y_k obtida de acordo com (E.6). Neste

momento, utilizamos as equações (E.21) e (E.22) para corrigir a estimativa predita de $\bar{x}_k$, dando origem a $\hat{x}_k$ e $\hat{p}_k$.

Considere o sistema linear (E.5)-(E.6). Além disso, suponha que o vetor inicial de estados x_0 segue uma distribuição Normal. Partindo de condições iniciais

$$\hat{x}_0 \triangleq E[x_0], \hat{P}_0 \triangleq E[[x_0 - \hat{x}_0][x_0 - \hat{x}_0]^T],$$

e assumindo conhecidas as estimativas $\hat{x}_{k-1}$ e $\hat{p}_{k-1}$ do passo anterior, a recursão do algoritmo do Filtro de Kalman para as estimativas do instante k atual são dadas por:

1. **Predição das Estimativas**

$$\bar{x}_k = A_{k-1}\hat{x}_{k-1} + B_{k-1}\bar{u}_{k-1}, \text{ (E.19)}$$
$$\bar{P}_k = A_{k-1}\hat{p}_{k-1}A^T_{k-1} + B_{k-1}P_{u,k-1}B^T_{k-1} + Q_{k-1}, \text{ (E.20)}$$

em que $u_{k-1} \sim N(\bar{u}_{k-1}, P_{u,k-1})$.

2. **Correção das Estimativas**

- Se uma medida y_k estiver disponível no instante atual, faça

$$K_k = \bar{P}_k C_k^T (R_k + C_k \bar{P}_k C_k^T)^{-1},$$
$$\hat{x}_k = \bar{x}_k + K_k(y_k - C_k\bar{x}_k), \text{ (E.21)}$$
$$\hat{P}_k = (I - K_k C_k)\bar{P}_k(I - K_k C_k)^T + K_k R_k K_k^T. \text{ (E.22)}$$

- Caso contrário

$$\hat{x}_k = \bar{x}_k, \text{ (E.23)}$$
$$\hat{P}_k = \bar{P}_k \text{ (E.24)}$$

As equações (E.23) e (E.24) são úteis quando medidas de sensores não estão disponíveis a todo instante kT de cálculo do Filtro de Kalman, em que T é o período de amostragem. Sem dúvida, a matriz Q_{k-1} de ruído de processo é o parâmetro do filtro que necessita da maior quantidade de esforço para o seu ajuste, uma vez que R_k é geralmente obtida a partir das especificações técnicas dos sensores. Sugere-se escolher os elementos da diagonal principal de Q_{k-1} de tal forma que seus intervalos $3\sigma_{sd}$, em que σ_{sd} denota o desvio-padrão, correspondam aos máximos erros possíveis entre o modelo (E.5) e a evolução efetiva do estado x_k do sistema.

2.4 FILTRO DE KALMAN ESTENDIDO

2.4.1 INTRODUÇÃO

O Filtro de Kalman Estendido (FKE), inicialmente proposto por Jazwinski em [6], talvez seja a extensão mais simples e direta do Filtro de Kalman para sistemas não-lineares

$$x_k = f(x_{k-1}, u_{k-1}) + v_{k-1}, \quad \text{(E.3)}$$
$$y_k = h(x_k) + w_k, \quad \text{(E.4)}$$

A ideia por trás do algoritmo é tomada do Cálculo diferencial: para desvios suficientemente pequenos em relação a uma posição de referência, funções deriváveis não-lineares podem ser localmente aproximadas por retas. Considerando incrementos Δ_x e Δ_u pequenos, a aproximação linear de primeira ordem é

$$f(x_{k-1} + \Delta_x, u_{k-1} + \Delta_u) \approx f(x_{k-1}, u_{k-1}) + J_{f,x}(x_{k-1}, u_{k-1})\Delta_x + J_{f,u}(x_{k-1}, u_{k-1})\Delta_u \quad \text{(E.25)}$$

torna-se uma boa aproximação local de (E.3) avaliada no ponto (x_{k-1}, u_{k-1}), em que

$$J_{f,x}(x_{k-1}, u_{k-1}) = \left.\frac{\partial f}{\partial x_{k-1}}\right|_{x_{k-1}, u_{k-1}}, \; J_{f,u}(x_{k-1}, u_{k-1}) = \left.\frac{\partial f}{\partial u_{k-1}}\right|_{x_{k-1}, u_{k-1}}, \quad \text{(E.26)}$$

são, respectivamente, as matrizes Jacobianas de $f(x_{k-1}, u_{k-1})$ em relação à x_{k-1} e u_{k-1}. Análise semelhante leva a proposição do modelo linearizado

$$h(x_k + \Delta_x) \approx h(x_k) + J_{h,x}(x_k)\Delta_x,$$
$$J_{h,x}(x_k) = \left.\frac{\partial h}{\partial x_k}\right|_{x_k}, \quad \text{(E.27)}$$

para equação não-linear de saída (E.4).

Resta, ainda, definir em torno de qual ponto (x_{k-1}, u_{k-1}) as derivadas em (E.26) e (E.27) devem ser avaliadas em cada passo do filtro. Procurando assegurar que Δ_x e Δ_u sejam pequenos e que, por conseguinte, (E.25) seja válida, [6] propõe que (E.3) e (E.4) sejam linearizados em torno da melhor estimativa disponível no momento para o estado real x_k.

Para (E.26), usada na linearização da função de evolução de processo (E.3), a melhor estimativa de $(\mathrm{x}_{\mathrm{k}-1}, \mathrm{u}_{\mathrm{k}-1})$ é dada pelas estimativas anteriores $(\hat{x}_{k-1}, \hat{u}_{k-1})$, ao passo que (E.27) é linearizado em torno da estimativa $\bar{\mathrm{x}}_{\mathrm{k}}$. Deve ressaltar que atenção especial deve ser dedicada à garantia das condições de linearidade necessárias ao Filtro de Kalman Estendido, visto que os modelos linearizados (E.25)-(E.27) podem gerar filtros fortemente instáveis caso as hipóteses de linearidade local sejam violadas.

2.4.2 ALGORITMO DO FILTRO DE KALMAN ESTENDIDO

As semelhanças entre os algoritmos do Filtro de Kalman e do Filtro de Kalman Estendido são patentes. De fato, são exatamente as mesmas equações da Seção 2.3.1 considerando o modelo linearizado (E.25)-(E.27) para propagação das incertezas através de (E.3)-(E.4). Contudo, não se pode garantir que as estimativas $\hat{\mathrm{x}}_{\mathrm{k}}$ obtidas pelo Filtro de Kalman Estendido para sistemas não-lineares genéricos sejam ótimas em algum sentido. Para sistemas superamostrados, nos quais a resposta dinâmica do sistema é incremental no intervalo de tempo τ entre amostras do filtro, e também para sistemas fracamente não-lineares, o modelo (E.3)-(E.4) aproxima-se de (E.5)-(E.6), dando ao Filtro de Kalman Estendido a capacidade de prover boas estimativas. Fora destas situações, no entanto, a violação das hipóteses de linearidade necessárias ao Filtro de Kalman Estendido pode tornar o seu desempenho ruim. Tomadas as devidas precauções, no entanto, para assegurar seu correto funcionamento, o Filtro de Kalman Estendido é uma opção de fácil e rápida implementação para diversas situações práticas de filtragem de sistemas não-lineares, sendo um dos filtros mais comumente adotados para estimação de estados deste tipo de sistema.

Em razão das diferentes notações disponíveis na literatura para representação de matrizes Jacobianas, descreve-se rapidamente a forma adotada neste trabalho antes da apresentação do algoritmo do Filtro de Kalman Estendido. Para uma função genérica

$$g: \mathrm{R}^{nx} \rightarrow \mathrm{R}^{ny}$$

$$x \rightarrow g(x)$$

em que $x = [x_1, x_2, \dots, x_{nx}]^T$ e $g(x) = [g_1(x), g_2(x), \dots, g_{nx}(x)]^T$, matriz jacobiana $J_{g,x}$ é dada por

$$J_{g,x} = \frac{\partial g(x)}{\partial x} = \begin{bmatrix} \frac{\partial g_1(x)}{\partial x_1} & \cdots & \frac{\partial g_2(x)}{\partial x_{nx}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{ny}(x)}{\partial x_1} & \cdots & \frac{\partial g_{ny}(x)}{\partial x_{nx}} \end{bmatrix} \quad \text{(E.28)}$$

Considere o sistema não-linear (E.3)-(E.4). Partindo de condições iniciais

$$\hat{\mathrm{x}}_0 \triangleq E[\mathrm{x}_0], \hat{P}_0 \triangleq E[[x_0 - \hat{\mathrm{x}}_0][x_0 - \hat{\mathrm{x}}_0]^T],$$

e assumindo conhecidas as estimativas $\hat{\mathrm{x}}_{\mathrm{k}-1}$ e $\hat{\mathrm{p}}_{\mathrm{k}-1}$ do passo anterior, a recursão do algoritmo do Filtro de Kalman Estendido para as estimativas do instante k atual são dadas por:

1. **Predição das estimativas**

$$J_{f,x} = \left.\frac{\partial f(x_{k-1},u_{k-1})}{\partial x_{k-1}}\right|_{x_{k-1},u_{k-1}},$$

$$J_{f,u} = \left.\frac{\partial f(x_{k-1},u_{k-1})}{\partial u_{k-1}}\right|_{x_{k-1},u_{k-1}},$$

$$\bar{x}_{k-1} = f(\hat{x}_{k-1}, \hat{u}_{k-1})$$

$$\bar{P}_k = J_{f,x}\hat{P}_{k-1}J^T{}_{f,x} + J_{f,u}P_{u,k-1}J^T{}_{f,u} + Q_{k-1}$$

em $u_{k-1} \sim N(\bar{u}_{k-1}, P_{u,k-1})$.

2. **Correção das estimativas**

- Se uma medida yk estiver disponível no instante atual, faça

$$J_{h,x}\left.\frac{\partial h(x_k)}{\partial x_k}\right|_{x_k},$$

$$\mathrm{K_k} = \overline{\mathrm{P}}_\mathrm{k}\mathrm{J}_{\mathrm{h,x}}^{\mathrm{T}}(\mathrm{R_k} + \mathrm{J_{h,x}}\overline{\mathrm{P}}_\mathrm{k}\mathrm{J}_{\mathrm{h,x}}^{\mathrm{T}})^{-1},$$

$$\hat{\mathrm{x}}_\mathrm{k} = \bar{\mathrm{x}}_\mathrm{k} + \mathrm{K_k}(\mathrm{y_k} - \mathrm{h}(\bar{\mathrm{x}}_\mathrm{k})),$$

$$\hat{\mathrm{P}}_\mathrm{k} = (\mathrm{I} - \mathrm{K_k}\,\mathrm{J_{h,x}})\overline{\mathrm{P}}_\mathrm{k}(\mathrm{I} - \mathrm{K_k}\,\mathrm{J_{h,x}})^{\mathrm{T}} + \mathrm{K_k}\,\mathrm{R_k}\mathrm{K}_\mathrm{k}^{\mathrm{T}}.$$

- Caso contrário

$$\hat{\mathrm{x}}_\mathrm{k} = \bar{\mathrm{x}}_\mathrm{k},$$
$$\hat{\mathrm{P}}_\mathrm{k} = \overline{\mathrm{P}}_\mathrm{k}.$$

Note que o Filtro de Kalman Estendido preocupa-se em propagar apenas a média e a matriz de covariâncias de (E.19) por meio de linearizações locais, ainda que estas informações sejam suficientes apenas para sistemas lineares (E.5)-(E.6).

2.5 MODELO DO FILTRO DE KALMAN ESTENDIDO PARA SLAM

O algoritmo de SLAM difere do algoritmo do Filtro de Kalman e do Filtro de Kalman estendido pelo simples fato do uso de um estado conjunto que também incluem observações.

2.5.1 VETOR DE ESTADO

O algoritmo de SLAM utiliza um único vetor conjunto de variáveis aleatórias que consiste tanto estimativa da posição do robô e as estimativas das características. O robô atravessa o ambiente, e enquanto faz seu trajeto detecta todas as características do ambiente que são visíveis e perceptíveis. O vetor de estado, não possui um tamanho constante. Em vez disso, o tamanho do vetor de estado mantém incrementando-se com as observações de novas características que não correspondem com as características vistas anteriormente. O estado do sistema no momento k será representada pelo vector de estado $x(k)$, que consiste em variáveis aleatórios escalares v_n para representar posição estimada, e um vetor de variáveis variáveis aleatórios f_n , para cada uma das variável de f_n que representa a posiçaõ estimada de uma única caracteristica:

$$x_k = \begin{bmatrix} x_{v,k} \\ x_{f1,k} \\ \vdots \\ x_{fn-1,k} \\ x_{fn,k} \end{bmatrix} \text{(E.29)}$$

todas as coordenadas $x_{i,k}$ dentro do vetor de estados são denominadas relativas para uma referência global. De acordo com a terminologia de estabelecida por Newman [7], o algoritmo de SLAM em quais todas as estimativas de estados são com respeito a algo em comum, a referencia global se refere ao filtro do mapa absoluto (AMF). É de notar que uma vez que todos os estados são tidos, como no que diz respeito, a uma estrutura de referência global, não foi denotado notação para classificar as variáveis no quadro de referência global.

O vetor de estado cumulativo pode ser dividido em duas partes, o vetor com a posição do robô e um vetor com a posição das características:

$$x_k = \begin{bmatrix} x_{v,k} \\ x_{m,k} \end{bmatrix} \text{(E.30)}$$

onde $x_{v,k}$ é o vetor com a posição do robô e $x_{m,k}$ é o vetor do mapa, o vetor com todas as características do mapa. Um robô atravessando um ambiente de duas dimensões possui três variáveis para representar sua posição. Onde temos:

$$x_{v,k} = \begin{bmatrix} x_{v,k} \\ y_{v,k} \\ \theta_{v,k} \end{bmatrix} \text{(E.31)}$$

onde $x_{v,k}$, $y_{v,k}$ e $\theta_{v,k}$ representam a estimativa média ao longo do eixo x, y e a posição em relação ao quadro de referência global, respectivamente.

As caracteristicas $x_{fn,k}$ podem ser definidas no mapa por uma posição de duas dimensões no espaço

$$x_{fn,k} = \begin{bmatrix} x_{fi,k} \\ y_{fi,k} \end{bmatrix} \text{(E.32)}$$

o vetor do mapa é dado por

$$x_{m,k} = \begin{bmatrix} x_{f1,k} \\ x_{f2,k} \\ \vdots \\ x_{fn-1,k} \\ x_{fn,k} \end{bmatrix} \text{(E.33)}$$

2.5.2 MODELO DE MOVIMENTO DO VEÍCULO

O modelo de movimento do veículo ou simplesmente modelo de movimento, capta a relação fundamental entre o estado passado do robô, $x_{v,k}$ e o estado atual, $x_{v,k+1}$, dada uma entrada u_k

$$x_{v,k} = f_{v,k}(x_{v,k-1}, u_k, v_{u,k}) \text{ (E.34)}$$

Um modelo de movimento preciso é um requisito importante para um esquema de boa navegação. No entanto, como veículos realistas sempre sofrem de ruídos, qualquer modelo de veículo é obrigado a ter erros incluso em função do ruído. Dois modelos de movimento amplamente populares são geralmente adaptados para atender a maioria dos requisitos: o modelo de movimento baseado em odometria e o modelo de movimento baseado em velocidade.

O modelo de movimento baseado em odometria é popular com bases mais comerciais que fornecem informações de odometria usando cinemática. O modelo de movimento baseado em odometria fornece a distância percorrida e o ângulo de giro do robô. Por outro lado, o modelo de movimento baseado em velocidade assume que independentes das velocidades de translação e de rotação estarem especificadas para os motores, o movimento do robô é realizada durante o tempo desejado para chegar ao destino.

Na prática, os modelos baseados em odometria são observados para ser mais preciso do que os modelos baseados em velocidade. Isto pode ser atribuído ao fato de que a maioria dos robôs comerciais não executam comandos de velocidade com base no nível de precisão que pode ser alcançada através da medição da revolução de rodas do robô.

o problema com odometria é, no entanto, que os dados so estão disponíveis após que o comando de movimento for executado. Assim, ele não pode ser usado para os métodos de planeamento de movimento, onde previsão dos efeitos do movimento é necessário para evitar colisões.

Neste trabalho, o modelo de movimento baseado em odometria foi usado. O modelo usa como entradas, a estimativa passada da posição do robô, $\hat{x}_{v,k|k}$ e a entrada de controle u_k, e produz como saída a media da previsão atual da posição do robô.

$$\begin{bmatrix} \hat{x}_{v,k+1|k} \\ \hat{y}_{v,k+1|k} \\ \hat{\theta}_{v,k+1|k} \end{bmatrix} = \begin{bmatrix} \hat{x}_{v,k+1|k} + r_{u,k}\cos(\hat{\theta}_{v,k} + \theta_{u,k}) \\ \hat{y}_{v,k+1|k} + r_{u,k}\sin(\hat{\theta}_{v,k} + \theta_{u,k}) \\ \hat{\theta}_{v,k} + \theta_{u,k} \end{bmatrix} \text{(E.35)}$$

Os escalares $r_{u,k}$ e $\theta_{u,k}$ são parte do comando de controle u_k, emitido para o robô.

$$u_k = \begin{bmatrix} r_{u,k} \\ \theta_{u,k} \end{bmatrix} \text{(E.36)}$$

2.5.3 MODELO DAS CARACTERÍSTICAS

Uma característica na terminologia do SLAM é uma parte do ambiente que pode ser observado de forma confiável através de sensores do robô. Características devem ser descrito de uma forma paramétrica para eles serem incorporados em um modelo de estado. O âmbito do presente trabalho aplica-se apenas para ambientes estáticos, o que implica que as características devem ser consideradas como sendo estacionário. Assim, somente o vetor de estados com a posição do robô que é dinâmico por natureza. Isto é, o modelo da característica é simplesmente.

$$x_{m,k} = x_{m,k-1} \text{ (E.37)}$$

onde $x_{m,k}$ são estimativas das características no instante k.

2.5.4 MODELO DE MEDIÇÃO DO SENSOR

Modelos de medição descrevem o processo de formação em que o sensor mede no mundo físico. Robôs de hoje usam uma variedade de modalidades de sensores diferentes, como sensores táteis, sensores de alcance ou câmeras. A observação do estado pode ser modelado como

$$z_k = \mathrm{h}(x_k) + w_k$$

onde $z_k \in R^{fn}$ é a observação no instante k. e o h é o modelo das observações dos estados do sistema em função do tempo e *w(k)* é um vetor aleatório que descreve o ruído de medição e as incertezas inseridas no modelo.

2.5.5 DESCRIÇÃO DO PROCESSO

No tempo k, o robô tem uma posição estimada $\hat{x}_{v,k|k}$ e sua localização é conhecida com uma incerteza $P_{v,k|k}$. O primeiro subescrito *v* indicativo da parte do vetor de estados ou matriz de covariância associado com a posição do robô e o subescrito k indica o tempo, com a exibição condicional que o estado do robô foi atualizado depois da k^{th} observação. Se o robô tem como entrada u_k, é esperado que ele se mova para a posição $x_{v,k+1|k}$. Esta estimativa local, também chamado de uma estimativa de estado priori, possui um maior grau de incerteza que a incerteza do robô após a última observação ($x_{v,k|k}$). Isto é devido à adição de erro incorrido devido ao movimento do robô.

Uma vez que o robô assume que ele atingiu o destino emitido a ele pela entrada de controle, os sensores do robô adquirem informações sobre o meio ambiente. Especificamente, as características únicas do ambiente visíveis através do sensor são extraídas para atualizar o vetor de estado. As medidas destas características únicas estão disponíveis a partir somente do quadro de referência do robô.

É a partir dessas observações que o filtro corrige a sua estimativa da posição do robô para $x_{v,k+1|k+1}$ que é uma estimativa a posteriori. Simultaneamente, também corrigi a estimativa do mapa $x_{m,k+1|k+1}$ e reduz ambas as incertezas da localização $P_{v,k+1|k+1}$ e do mapa $P_{m,k+1|k+1}$.

2.5.6 A EVOLUÇÃO DO PROCESSO

O movimento do robô e as medições das características do mapa são controlados por um modelo de transição de estados no tempo discreto, dado por:

$$x_k = f(x_k, u_k, v_k), \quad \text{(E.38)}$$

$$z_k = h(x_k) + w_k, \quad \text{(E.39)}$$

A função *f* é em função da entrada de controle, último estado do robô e ruído do processo. No caso do Filtro de Kalman, o modelo de transição de estado é uma transformação linear da entrada de controle e uma transformação linear do estado com o ruído incluso. Como o movimento é feito em duas dimensões, isto acaba induzindo uma não linearidade na transição de estado. O ruído de processo no sistema pode não ser uma Gaussiana na referencia global das coordenadas e por isso não pode ser modelado como uma simples Gaussiana de média zero, em relação a qual estado do robô e do mapa está sendo estimados. Se v_n é o número de dimensões da posição do robô, f_n é o número de característica no mapa e *c* é a dimensão de cada vetor de característica, então

$$x_k \in R^{vn+cfn} \text{ (E.40)}$$

O vetor das entradas $u_k \in R^{un}$ é o vetor de controle do veiculo e $v_k \in R^{un}$ é um vetor Gaussiano aleatório com media zero e matriz de covariância $Q_k \in R^{un \times un}$ onde u_n é a dimensionalidade das entradas de controle. A função $f: R^{un+cfn} \rightarrow R^{un+cfn}$ é a equação que modela a movimentação do robô.

O Valor esperado deste sistema em tempo discreto no instante k+1, é:

$$\hat{x}_{k+1} \triangleq E[x_{k+1}] \text{ (E. 41)}$$

e evolui de acordo com

$$\hat{x}_{k+1} = f(\hat{x}_k, u_k, 0) \text{ (E.42)}$$

Similar a propagação calculada para sistemas lineares, a matriz de covariância do erro para este sistema é calculada aplicando o operador esperança para equação E.43

$$P_{xx,k+1} \triangleq E[[x_{k+1} - \hat{x}_{k+1}][x_{k+1} - \hat{x}_{k+1}]^T] \text{ (E.43)}$$

onde

$$x_{k+1} - \hat{\mathrm{x}}_{k+1} = f(x_k, u_k, v_k) - f(\hat{\mathrm{x}}_k, u_k, 0) \text{ (E.44)}$$

A matriz de covariancia torna-se

$$P_{xx,k+1} \triangleq E[[f(x_k, u_k, v_k) - f(\hat{\mathrm{x}}_k, u_k, 0)][f(x_k, u_k, v_k) - f(\hat{\mathrm{x}}_k, u_k, 0)]^T] \text{ (E.45)}$$

o qual podemos reduzir para

$$P_{xx,k+1} \triangleq E\left[[f(x_k, u_k, v_k) - f(\hat{\mathrm{x}}_k, u_k, 0)][f(x_k, u_k, v_k)^T - f(\hat{\mathrm{x}}_k, u_k, 0)^T] \right] \text{ (E.46)}$$

A equação de estado é aproximadamente

$$f(x_k, u_k, v_k) \approx f(\hat{\mathrm{x}}_k) + \mathrm{F_k}[x_k - \hat{\mathrm{x}}_k] + 0 + \mathrm{G_k} v_k - 0 \text{ (E.47)}$$

simplificando

$$f(x_k, u_k, v_k) \approx f(\hat{\mathrm{x}}_k) + \mathrm{F_k}[x_k - \hat{\mathrm{x}}_k] + \mathrm{G_k} v_k \text{ (E.48)}$$

usando aproximação por série de Taylor, onde $\mathrm{F_k}$ e $\mathrm{G_k}$ são Jacobianos. Substituindo, temos:

$$P_{xx,k+1} = E[[\mathrm{F_k}[x_k - \hat{\mathrm{x}}_k] - \mathrm{G_k} v_k][\mathrm{F_k}[x_k - \hat{\mathrm{x}}_k] - \mathrm{G_k} v_k]^T] \text{ (E.49)}$$

$$P_{xx,k+1} = E\left[[\mathrm{F_k}[x_k - \hat{\mathrm{x}}_k] - \mathrm{G_k} v_k]\left[\mathrm{F_k}\left[{x_k}^T - {\hat{\mathrm{x}}_k}^{\mathrm{T}}\right]{\mathrm{F_k}}^{\mathrm{T}} - {\mathrm{G_k}}^{\mathrm{T}} {v_k}^{\mathrm{T}}\right] \right] \text{ (E.50)}$$

substituindo $\tilde{x}_k = x_k - \hat{\mathrm{x}}_k$, temos:

$$P_{xx,k+1} = E\left[\mathrm{F_k}\tilde{x}_k {\tilde{x}_k}^T {\mathrm{F_k}}^{\mathrm{T}} + \mathrm{F_k}\tilde{x}_k {v_k}^T {\mathrm{G_k}}^{\mathrm{T}} + \mathrm{G_k} v_k {\tilde{x}_k}^T {\mathrm{F_k}}^{\mathrm{T}} + \mathrm{G_k}\tilde{x}_k {\tilde{x}_k}^T {\mathrm{G_k}}^{\mathrm{T}}\right] \text{ (E.51)}$$

A covariância entre o erro do processo e o estado ou estado estimado é zero. Se a variância do ruído de processo é Q_k, podemos escrever agora a propagação da variância como sendo:

$$P_{xx,k+1} = \mathrm{F_k} P_{xx,k+1} {\mathrm{F_k}}^{\mathrm{T}} + \mathrm{G_k} Q_k {\mathrm{G_k}}^{\mathrm{T}} \text{ (E.52)}$$

Isto representa à média e a propagação da variância da posição do robô no EKF-SLAM. Assumindo que o ambiente seja estático, a estimativa da característica pode ser escrita como

$$x_{f,k+1} = x_{f,k} \quad \text{(E.53)}$$

Assim, uma atualização a priori do estado não irá afetar a média ou variância das características.

2.6 ALGORITMO DO EKF-SLAM

O algoritmo do EKF-SLAM é uma extensão do Filtro de Kalman, como explicado anteriormente. O algoritmo é dividido em quatro estágios e cada iteração atravessa os estágios na mesma ordem. Os quatros estágios são:

- Estágio de Predição.
- Estágio de Observação.
- Estágio de Atualização.
- Estágio de Aprimoramento (ou Acréscimo).

Cada iteração do algoritmo EKF-SLAM começa após o término da execução de um comando de controle de entrada emitida para o robô. A iteração do algoritmo é exibida na Figura 4.

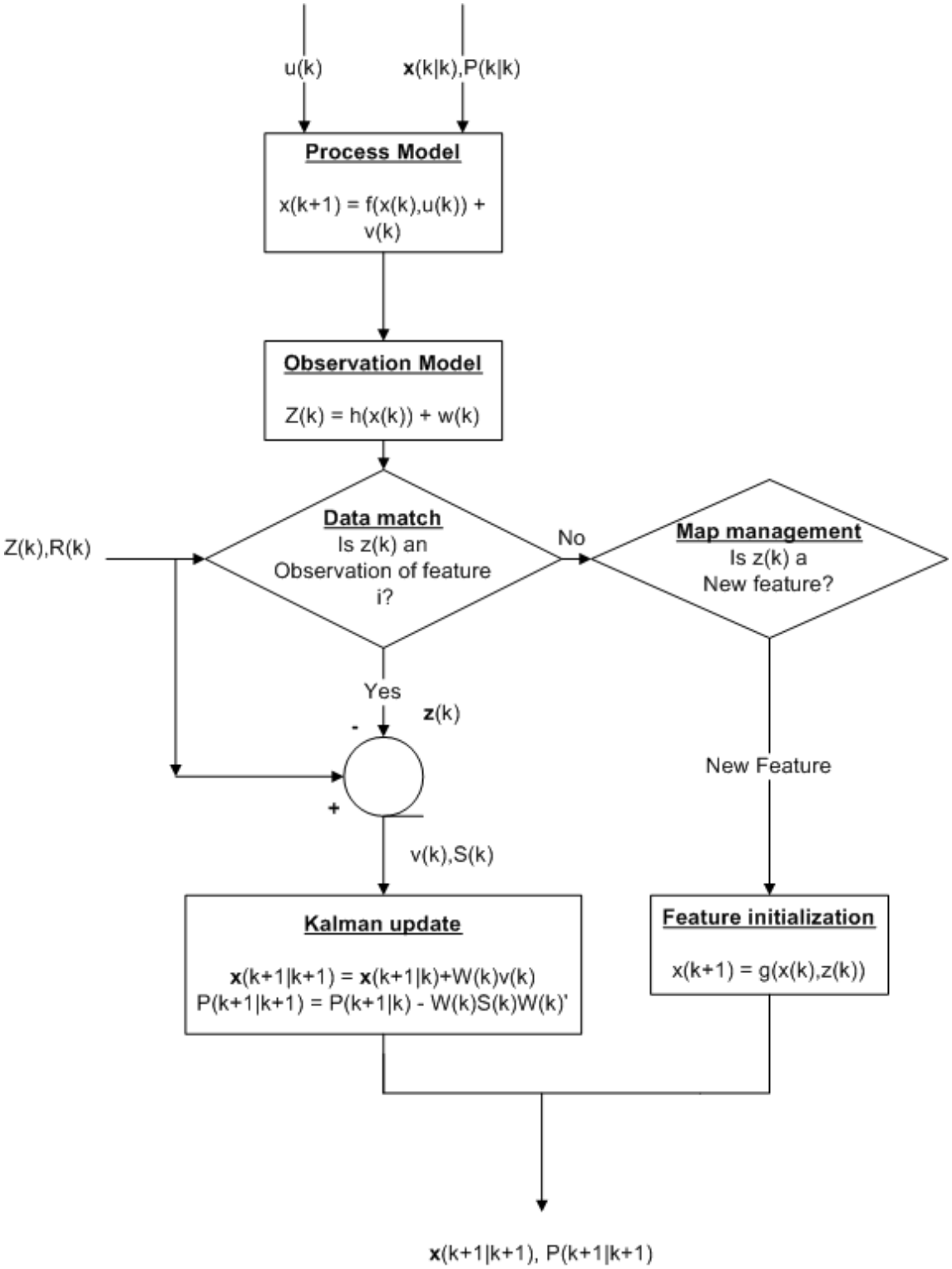

Figura 4 - Algoritmo do EKF-SLAM

2.6.1 ESTÁGIO DE PREDIÇÃO

A etapa de previsão envolve a atualização da média e variância do estado após um movimento. Isto é feito utilizando a informação de controle e a variância do erro do processo. O estado do sistema, após a fase de predição é chamado um estado priori, e este estado ainda não está atualizada com as informações dos sensores.

A predição do estado ou estado a priori $\hat{\mathrm{x}}_{v,k+1|k}$ para o robô é calculado como

$$\hat{\mathrm{x}}_{v,k+1|k} = f(\hat{\mathrm{x}}_k, u_k, 0) \quad \text{(E.54)}$$

A variância do estado do robô pode ser encontrada através da equação de propagação da variância:

$$P_{vv,k+1|k} = \mathrm{F_k} P_{vv,k+1|k} \mathrm{F_k}^{\mathrm{T}} + \mathrm{G_k} Q_k \mathrm{G_k}^{\mathrm{T}} \quad \text{(E.55)}$$

A média e variância das características não mudam durante este estágio como a movimentação do robô não influenciam observações passadas. Assim, as equações conjuntas dos estados podem ser escritas como

$$\begin{bmatrix} \hat{\mathrm{x}}_{v,k+1|k} \\ \hat{\mathrm{x}}_{m,k+1|k} \end{bmatrix} = \begin{bmatrix} f(\hat{\mathrm{x}}_k, u_k, 0) \\ \hat{\mathrm{x}}_{m,k|k} \end{bmatrix} \quad \text{(E.56)}$$

e a variância conjunta, a priori, como:

$$P_{vv,k+1|k} = \begin{bmatrix} \mathrm{F_k} 0 \\ 0\ I \end{bmatrix} P_{k|k} \begin{bmatrix} \mathrm{F_k} 0 \\ 0\ I \end{bmatrix}^T + \begin{bmatrix} \mathrm{G_k} Q_k \mathrm{G_k}^{\mathrm{T}} \\ 0 \end{bmatrix} \quad \text{(E.57)}$$

2.6.2 ESTÁGIO DE OBSERVAÇÃO

O estágio de observação é a fase em que a informação do ambiente é recolhida através dos sensores e é feita a associação com as características. Esta subseção trata da previsão da média e variância das características o qual é necessário combiná-los para observações.

A média pode ser definida como

$$\hat{z}_k = h(\hat{x}_k) \quad \text{(E.58)}$$

Linearizando a equação do modelo de observação, temos:

$$z_k \approx h(\hat{x}_k) + H_k[x_k - \hat{x}_k] + w_k \text{ (E.59)}$$

A variância pode ser encontrada, como:

$$P_{zz,k} = E[[z_k - \hat{z}_k][z_k - \hat{z}_k]^T] \text{ (E.60)}$$

o qual é aproximada para

$$P_{zz,k} \approx E[[H_k[x_k - \hat{x}_k] + w_k][H_k[x_k - \hat{x}_k] + w_k]^T] \text{ (E.61)}$$

Mediante uma simplificação e através do teste não-dependência do erro de observação no estado e assumindo que o erro de observação é R_k, temos:

$$P_{zz,k} = H_k P_{k+1|k} {H_k}^T + R_k \text{ (E.62)}$$

A variância $P_{zz,k}$ é uma necessidade para ser capaz de associar com as características.

2.6.3 ESTÁGIO DE ATUALIZAÇÃO

O estágio de atualização usa as informações adquiridas das associações de características extraídas dos sensores para, simultaneamente, atualizar a posição do robô e o mapa. A equação de atualização para o EKF-SLAM é baseada na atualização do Filtro de Kalman

$$E(\hat{x}_{k+1|k+1}) = \hat{x}_{k+1|k} + P_{xz} {P_{zz}}^{-1}(z_k - \hat{z}_k) \text{ (E.63)}$$

e a covariância é dado por:

$$P_{k+1|k+1} = P_{k+1|k} - P_{xz,k} {P_{zz,k}}^{-1} P_{zx} \text{ (E.64)}$$

O valor de P_{zz} é a variância da observação. O valor de P_{zx} é a covariância entre a posição do robô e as caracteristicas:

$$P_{xz|z} = E[[x - \hat{x}][z_k - \hat{z}_k]^T] \text{ (E.65)}$$

Esta equação pode ser escrita como:

$$P_{xz|z} = E\left[\left[x_{k+1} - \hat{x}_{k+1|k}\right]\left[H_k\left[x_k - \hat{x}_{k+1|k}\right] + w_k\right]^T\right] \text{ (E.66)}$$

o qual pode ser simplificado e removendo termos sem covariância com os ruídos das observações

$$P_{xz|z} = P_{xx|z}{H_k}^T \text{ (E.67)}$$

Similarmente, temos:

$$P_{zx|z} = E[[z_k - \hat{z}_k][x - \hat{x}]^T] \text{ (E.68)}$$

$$P_{zx|z} = H_k P_{xx|z} \text{ (E.69)}$$

A média atualizada pode ser reescrita agora como:

$$E\left(\hat{x}_{k+1|k}\right) = \hat{x}_{k+1|k} + P_{xx|z}{H_k}^T\left(H_k P_{k+1|k}{H_k}^T + R_k\right)^{-1}(z_k - \hat{z}_k) \text{ (E.70)}$$

Substituindo o fator gerado pelo efeito da inovação pelo o valor K, o ganho de Kalman fica:

$$E\left(\hat{x}_{k+1|k}\right) = \hat{x}_{k+1|k} + \mathrm{K}(z_k - \hat{z}_k) \quad \text{(E.71)}$$

A atualização da covariância pode ser reescrita como:

$$P_{k+1|k+1} = P_{k+1|k} - P_{xx|z}{H_k}^T\left(H_k P_{k+1|k}{H_k}^T + R_k\right)^{-1} H_k P_{xx|z} \text{ (E.72)}$$

Simplificando por substituição do ganho de Kalman, temos:

$$P_{k+1|k+1} = P_{k+1|k} - K H_k P_{xx|z} \text{ (E.73)}$$

Finalmente chegamos a está equação:

$$P_{k+1|k+1} = (I - K H_k) P_{k+1|k} \quad \text{(E.74)}$$

2.6.4 ESTÁGIO DE APRIMORAMENTO

O estágio de aprimoramento em cada iteração do algoritmo de SLAM aprimora as observações não comparadas com as novas características no vetor de estado. Uma observação *z* é do tipo (r,θ) da referência do robô . Entretanto, a observação não pode ser adicionada diretamente ao vetor de estados, porque o vetor de estados é representado como uma referência global e uma referência local não é salva ou estimada. Assim, a característica é adicionada ao vetor:

$$\hat{x}_{fj} = m(z, \hat{x}_k) \quad \text{(E.75)}$$

onde $\hat{x}_k$ é a posição estimada do robô depois da atualização na mesma iteração.

A matriz de variância também precisa ser preenchida com os valores correspondentes para a característica adicionais no estado. Para encontrar a variância, podemos utilizar aproximações por series de Taylor das observações:

$$\hat{x}_{fj} = m(z, \hat{x}_k) + \mathrm{M}[z_k - \hat{z}_k] + N[y_k] \quad \text{(E.76)}$$

onde M e N são jacobianos para posição e erros respectivamente. Uma simplificação simples, a variância das características é encontrada como:

$$P_{fj} = MPM^T + NRN^T \quad \text{(E.77)}$$

A nova linha de covariância é adicionada dentro da matriz, como:

$$[MP_{vv}] \quad [MP_{vm}] \quad \left[P_{fj}\right] = [MPM^T + NRN^T] \quad \text{(E.78)}$$

3 SIMULAÇÃO

Inicialmente, foi pensado em utilizar este tipo de algoritmo de SLAM na participação do campeonato mundial de robótica simulada, os estudos iniciais foram feitos utilizando os programas MATLAB e Python, onde pode-se aprender mais profundamente sobre o funcionamento do algoritmo.

Para fazer este estudo, foi necessário a escolha do modelo de robô, que no nosso caso foi o P3Dx, onde iremos utilizar este modelo para desenvolver uma parametrização mais detalhada da sua cinemática e dinâmica. Com estes dados pretendemos fazer os modelos de movimentação do robô e modelo dos sensores. O mundo foi criado, com a implementação de várias características postas em pontos aleatórios, para podermos testar a consistência do nosso algoritmo de SLAM.

3.1 P3DX

Fabricado pelo Mobile Robots Inc, o robô utilizado neste trabalho, Pioneer P3-DX, é uma plataforma de pesquisa popular robótica móvel. Como os mais populares robôs de interiores, o Pioneer P3Dx não se move como um carro. Existem duas rodas principais, cada um dos quais está ligado ao seu próprio motor. Uma terceira roda (rodízio) é colocada na parte traseira para rolar ao longo de forma passiva, impedindo o robô de cair. O scanner SICK LMS 200 foi montado Pioneer P3Dx, que fornece informação sobre o ambiente do sistema.O Pioneer 3 DX pode atingir velocidades de 1,6 metros por segundo e transportar uma carga de até 23 kg. O robô é alimentado por três *hot-swappable* 9Ah baterias seladas.[10] Na figura 5 é exibido o P3Dx.

Figura 5 - O robô P3Dx da Mobile Robots.

3.2 PARAMETRIZAÇÃO

Parametrização de todas as observações é necessário colocá-los em uma estrutura matemática. Assim, a utilização dos dados no âmbito SLAM, todas as informações obtidas a partir do sistema precisam ser modelados em parâmetros que podem representá-la de forma consistente e completa.

O algoritmo EKF-SLAM assume um modelo de movimento para o robô e um modelo de sensor para a observação de acordo com a unidade de robô, a parametrização do movimento e do tipo de sensor. Estes modelos agem como a interface entre o robô e o algoritmo de SLAM.

Nosso sistema utiliza um sensor que retorna um intervalo de valores um plano do ambiente. Estes valores têm que ser parametrizados e aprimorados para os estados do EKF-SLAM. O robô em si, utiliza um sensor de rotações ou sensores baseados em ticks para o cálculo de posição, que é utilizada para estimar o movimento do robô e, consequentemente, a transformação da posição robô a partir de uma representação inicial estimada (antes do movimento) para representar a estimativa atual de posição.

3.2.1 MODELO DE MOVIMENTAÇÃO

O modelo de movimento usa a cinemática do robô para prever seu futuro e representar as estimativas, dada a posição atual do robô e uma entrada de controle. Assim, dado um sistema com a equação

$$\mathrm{x}_{v,k+1|k} = f_{v,k}\left(\mathrm{x}_{v,k}, u_k, v_{u,k}\right) \text{ (E.79)}$$

e um modelo de transição de estados de acordo com a entrada de controle, poderia ser:

$$\hat{\mathrm{x}}_{v,k+1|k} = f_{v,k}\left(\hat{\mathrm{x}}_{v,k}, u_{k+1}, 0\right) \text{ (E.80)}$$

Nós usamos um modelo de movimento baseado em odometria para prever a próxima posição do robô. Um modelo de movimento baseado em odometria tem como entrada a estimativa anterior de posição do robô, a distância percorrida pelo robô ou o ângulo de rotação robô, e retorna a estimativa representndo a nova posição do robô. Vale pena notar que apenas uma das formas de translação ou rotação pode ser dada como entrada em controle de uma única iteração do algoritmo. Isso é essencial porque a ordem da aplicação de entrada de controle poderia não está correta.

O algoritmo utiliza uma entrada de controle baseada em um *joystick*, no caso em que não existe qualquer entrada de controle prévio dada ao robô, mas o robô é conduzido a partir de um ponto para outro. Entretanto, a posição prevista $\hat{\mathrm{x}}_{v,k+1|k}$, neste caso, não precisa ser calculada a partir de uma função da entrada de controle. Em vez disso, a posição prevista logo após o comando de movimento já estará disponível. Entretanto, ainda precisamos de um modelo de $\hat{\mathrm{x}}_{v,k+1|k}$ como uma função do estado anterior $\hat{\mathrm{x}}_{v,k}$ e o controle u_k. A transformação é dada por

$$u_{k+1} = A(\hat{\mathrm{x}}_{v,k+1|k}, \hat{\mathrm{x}}_{v,k}) \text{ (E.81)}$$

A função de transição de estado para a predição desempenha um papel fundamental no modelo EKF-SLAM. A função Jacobiana derivada é utilizado para atualizar a matriz de covariância.

$$P_{k+1|k} = \mathrm{F_k} P_{k|k} \mathrm{F_k}^{\mathrm{T}} + \mathrm{G_k} Q_{k+1} \mathrm{G_k}^{\mathrm{T}} \text{ (E.82)}$$

Utilização a transformação a partir da equação E.81 a covariância Q_{k+1} e os jacobianos $\mathrm{F_k}$ e $\mathrm{G_k}$ serão calculados através da entrada de controle.

A entrada de controle é um vetor com duas quantidades, a distancia percorrida e o ângulo de rotação,

$$u_k = \begin{bmatrix} r_{u,k} \\ \theta_{u,k} \end{bmatrix} \text{ (E.83)}$$

Uma vez que a média do estado foi previsto depois de um comando de movimento, a covariância do Estado, que cresce, devido a erro de movimento, deve ser atualizado. Para calcular os jacobianos utilizaremos a equação E.84 abaixo:

$$\begin{bmatrix} \hat{x}_{v,k+1|k} \\ \hat{y}_{v,k+1|k} \\ \hat{\theta}_{v,k+1|k} \end{bmatrix} = \begin{bmatrix} \hat{x}_{v,k+1|k} + r_{u,k}\cos(\hat{\theta}_{v,k} + \theta_{u,k}) \\ \hat{y}_{v,k+1|k} + r_{u,k}\sin(\hat{\theta}_{v,k} + \theta_{u,k}) \\ \hat{\theta}_{v,k} + \theta_{u,k} \end{bmatrix} \text{ (E.84)}$$

O jacobiano da equação E.84 com respeito ao vetor de posições do robô irá ser

$$F_k = \begin{bmatrix} \frac{\partial \hat{x}_{v,k+1|k}}{\partial \hat{x}_{v,k|k}} & \frac{\hat{y}_{v,k+1|k}}{\partial \hat{y}_{v,k|k}} & \frac{\hat{\theta}_{v,k+1|k}}{\partial \hat{\theta}_{v,k|k}} \\ \frac{\partial \hat{x}_{v,k+1|k}}{\partial \hat{x}_{v,k|k}} & \frac{\hat{y}_{v,k+1|k}}{\partial \hat{y}_{v,k|k}} & \frac{\hat{\theta}_{v,k+1|k}}{\partial \hat{\theta}_{v,k|k}} \\ \frac{\partial \hat{x}_{v,k+1|k}}{\partial \hat{x}_{v,k|k}} & \frac{\hat{y}_{v,k+1|k}}{\partial \hat{y}_{v,k|k}} & \frac{\hat{\theta}_{v,k+1|k}}{\partial \hat{\theta}_{v,k|k}} \end{bmatrix} \text{ (E.85)}$$

resolvendo, temos:

$$F_k = \begin{bmatrix} 1 & 0 & -r_{u,k}\cos(\hat{\theta}_{v,k} + \theta_{u,k}) \\ 0 & 1 & r_{u,k}\sin(\hat{\theta}_{v,k} + \theta_{u,k}) \\ 0 & 0 & 1 \end{bmatrix} \text{(E.86)}$$

Similarmente, o jacobiano da equação de transição de estado com respeito ao comando de controle irá ser

$$G_k = \begin{bmatrix} \frac{\partial \hat{x}_{v,k+1|k}}{\partial r_{u,k}} & \frac{\hat{y}_{v,k+1|k}}{\partial \theta_{u,k}} \\ \frac{\partial \hat{x}_{v,k+1|k}}{\partial r_{u,k}} & \frac{\hat{y}_{v,k+1|k}}{\partial \theta_{u,k}} \\ \frac{\partial \hat{x}_{v,k+1|k}}{\partial r_{u,k}} & \frac{\hat{y}_{v,k+1|k}}{\partial \theta_{u,k}} \end{bmatrix} \text{(E.87)}$$

resolvendo, temos:

$$G_k = \begin{bmatrix} \cos(\hat{\theta}_{v,k} + \theta_{u,k}) & -r_{u,k}\sin(\hat{\theta}_{v,k} + \theta_{u,k}) \\ \sin(\hat{\theta}_{v,k} + \theta_{u,k}) & r_{u,k}\cos(\hat{\theta}_{v,k} + \theta_{u,k}) \\ 0 & 1 \end{bmatrix} \text{(E.88)}$$

3.2.2 MODELO DO SENSOR

O modelo de sensor ajuda a parametrizar os recursos obtidos a partir do *scanner* de alcance, e aprimorar ou associar às características já no algoritmo de SLAM. Como o *scanner,* o laser fornece valores de alcance, esses valores são usados com seu ângulo correspondente e a posição estimada do robô para transformá-los em coordenadas globais para a característica observada. Qualquer característica do ponto observado pelo laser é da forma

$$z_k = \begin{bmatrix} r_k \\ \theta_k \end{bmatrix} \text{(E.89)}$$

onde r_ké a leitura de distância obtida de um ângulo θ_k.

Após o aumento das características no vetor de estado, ele é comparado com as observações mais recentes em cada iteração. A correspondência requer a conversão da funcionalidade armazenado no vetor de estado para o sistema de referência do robô para verificar a sua proximidade com a observação. Este é modelado como

$$\hat{z}_k = \begin{bmatrix} \widehat{r_k} \\ \widehat{\theta_k} \end{bmatrix} \text{(E.90)}$$

Escrevendo $\widehat{r_k}$ e $\widehat{\theta_k}$ em termos da posição do robô e estimativa das características a partir do vetor de estados, a observação prevista é dada como:

$$\hat{z}_k = \begin{bmatrix} \sqrt{(x_k - x_{fi,k})^2 - (y_k - y_{fi,k})^2} \\ \tan^{-1}\frac{(y_k - y_{fi,k})}{(x_k - x_{fi,k})} - \theta_{v,k} \end{bmatrix} \quad \text{(E.91)}$$

3.3 SIMULAÇÃO DO EKF-SLAM

A simulação do sistema foi feito em dois ambientes de programação diferentes: o primeiro é o ambiente MATLAB e o segundo é o ambiente Python. Darei mais ênfase ao sistema simulado em Python, pois a versão mais atualizada do simulado está no mesmo. Inicialmente simulamos o ambiente com todo o algoritmo descrito anteriormente, e simulamos o ambiente com várias características inseridas aleatoriamente no mapa, além de inserirmos ruídos nas medições. É simulado o robô e os sensores em um ambiente com duas dimensões. Quando iniciada a simulação, adquiriram-se os dados dos sensores do robô e geramos de forma automática uma trajetória qualquer que o robô percorre. Esta informação alimenta um algoritmo de extração de característica que confere características identificadas a partir dos dados de digitalização não processados. As características devolvidas, assim, são usadas no algoritmo EKF-SLAM como observações para a atualização do filtro de Kalman. Ao robô é dado um conjunto de instruções para seguir um caminho e explorar o mapa a priori. O movimento do robô é uma atividade distinta, nesse algoritmo, o robô pode mover-se somente a uma distância máxima, ou um ângulo máximo, ambos definidos a priori, em uma única iteração. Isto ajuda a manutenção da variância da posição do robô baixa, o qual é essencial para construir o mapa. A posição do robô é consultada depois da função de movimento para gerar uma estimativa mais precisa a priori. Logo em seguida vem a fase de observação e de atualização seguido por outra nova iteração que consiste em todas as etapas anteriores, isso é feito até que o robô alcance seu último marco. O mapa atualizado na iteração anterior é considerado como o mapa EKF-SLAM final.

4 RESULTADOS

Abaixo é possivel visualizar várias figuras que representam a saída da simulação do EKF-SLAM. A Figura 6 representa o mapa a priori do ambiente e as posições das características colocadas de forma aleatória. A Figura 7 representa a trajetória percorrida pelo nosso robô, onde possuímos a posição real e posição gerada com o auxilio do EKF-SLAM. A Figura 8 representa os comandos de entrada dado ao robô e a Figura 9 representa o erro de medição em todos os eixos de percurso. A Figura 10 exibe o robô no mapa, onde se podem visualizar as características do ambiente, e as elipses em cima das características significam a incerteza da posição dessas características.

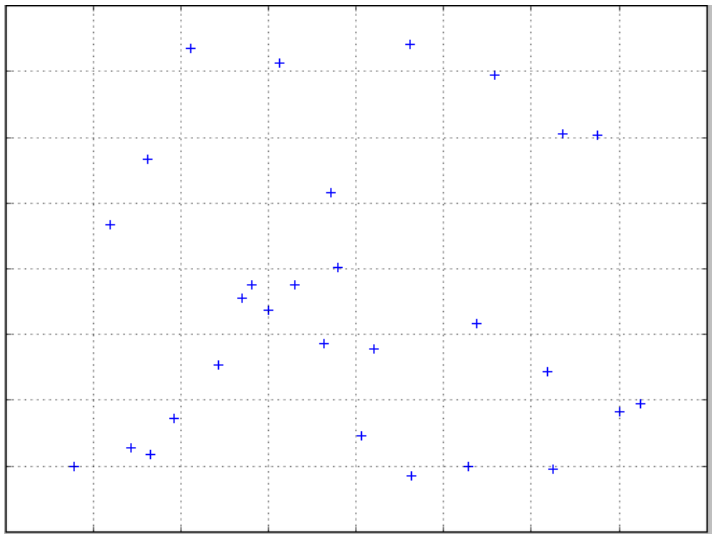

Figura 6 - Mapa do ambiente simulado com as características inseridas.

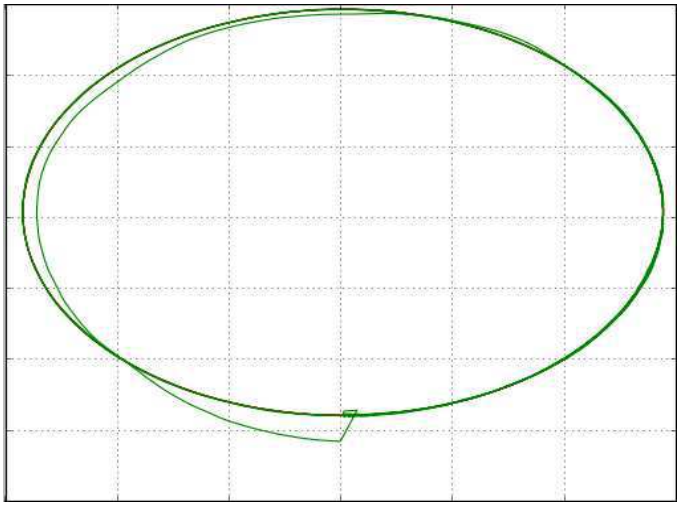

Figura 7 - Trajetória Real previamente definida (verde escuro) e a trajetória gerada pela saída do EFK (verde claro).

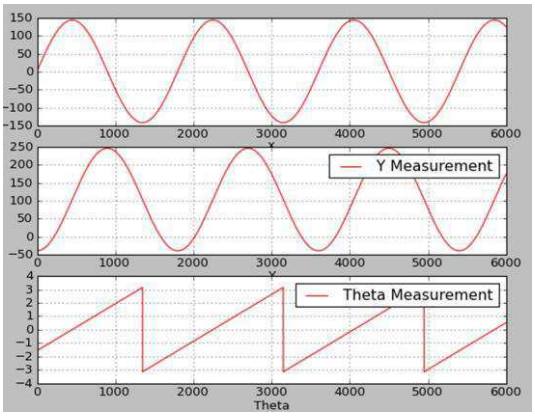

Figura 8 - Comandos de Controle

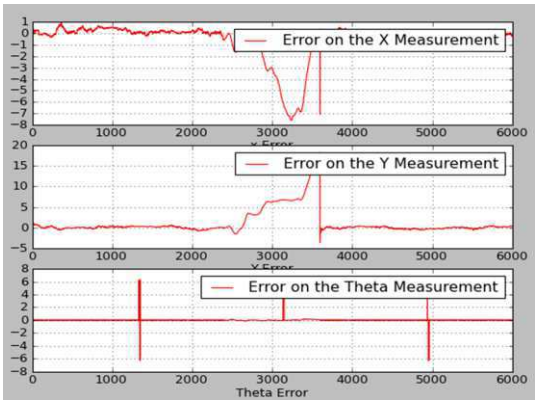

Figura 9 - Erros na Trajetória

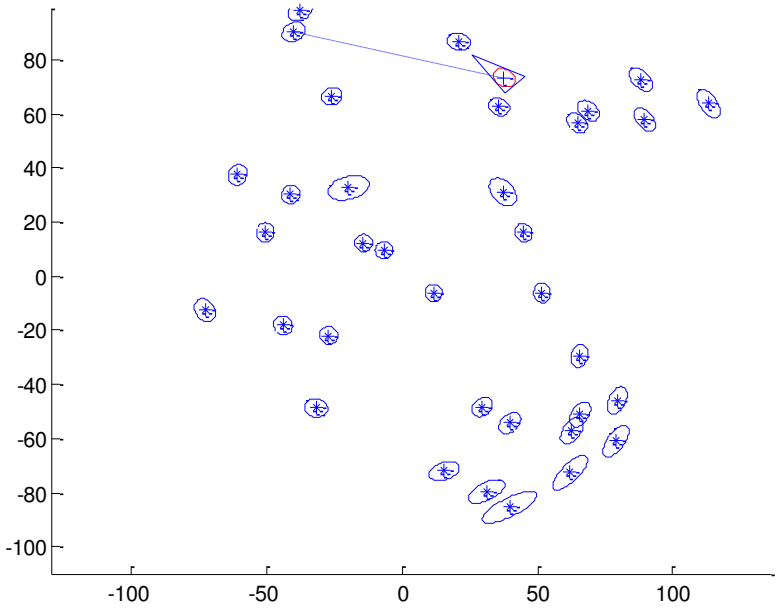

Figura 10 - Mapa com as características e o Robô.

Conclui-se desses dados simulados ao aumento do número de características com incertezas altas, o filtro tende a perde um pouco a sua convergência para a estimativa correta, porem com o passar do tempo, aumenta-se o número de características estimadas e o número de iterações no filtro, fazendo com que essas incertezas diminuem de forma monotônica, fazendo com que o algoritmo do filtro volte a ter uma convergência.

A implementação do EKF é bastante fácil se a compreensão sobre seu funcionamento for entendida, esse trabalho pode ser considerado uma introdução ao estudo de SLAM com o EKF.

5 CONCLUSÃO

Foi discutido o problema de automação de localização e mapeamento simultâneo. Podemos ver alguns pontos introdutórios para começar um estudo de navegação autônoma com robôs móveis. O problema foi resolvido utilizando o que chamamos de Filtro de Kalman Estendido (EKF).

Não podemos considerar este trabalho como um ponto final para este tipo de estudo, pois para uma eficiente navegação em ambientes internos, seria necessário modelar mais sensores, modelar melhor as características e introduzir um conceito de geometria mais avançado. Além do que o ajuste final do mapa criado com EKF junto do mapa que nos é dado a priori não foi discutido nesse trabalho.

O mundo de estudo da robótica é amplo e, muitas vezes, complexo. Tem muita coisa a ser feita e estudada para que possamos algum dia, criar nossa forma perfeita de robô autônomo.

6 BIBLIOGRAFIA

[1] SLAM (en.wikipedia.org/wiki/Simultaneous_localization_and_mapping).

[2] S.B. Williams. Efficient solutions to autonomous mapping and navigation problems. PhD thesis, Australian Centre for Field Robotics, 2001.

[3] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. Journal of Artificial Intelligence Research, 11(3):391–427, 1999.

[4] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. AI magazine, 9(2):61, 1988.

[5] FILTRAGEMESTOCÁSTICA PARA SISTEMAS HÍBRIDOS E SUAS APLICAÇÕES EM ROBÓTICA AÉREA, Universidade de Brasília, dissertação de mestrado de Pedro Henrique.

[6] A. H. Jazwinski, Stochastic Processes and Filtering Theory. New York: Dover Publications, INC, 1970.

[7] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. Robotics and Automation, IEEE Transactions on, 17(3):229–241, 2001.

[8] R. Kalman, “A new approach to linear filtering and prediction problems,” Transactions of the ASME Journal of Basic Engineering, no. 82, pp. 35–45, March 1960.

[9] D. Simon, Optimal State Estimation: Kalman, and Nonlinear Approaches. Wiley-Interscience, 2006.

[10] P3DX (http://www.mobilerobots.com/researchrobots/pioneerp3dx.aspx).

7 ANEXO

Implementação do Algoritmo do Filtro de Kalman Estendido utilizado em SLAM. Código desenvolvido pela equipe eROBOTICA no ambiente de programação Python.

```
Universidade Federal de Campina Grande
Centro de Engenharia Eletrica e Informatica

Dead-Reckoning Localization Simulation

import pylab
import numpy
import time
import math
from numpy import *
from math import *
from scipy import *
import scipy

class EKFLocalisation():
    def __init__(self):

        #inicializando o mapa
        self.Map = mat(140*rand(2,30)-70)

        #numero de passo
        self.nSteps = 6000

        #inicializando posicoes do odometro
        self.LastOdom = []
#         sigmar = 5e0
#         sigmao = 3e-3

        #
        self.UTrue =
mat(scipy.diag([pow(0.01,2),pow(0.01,2),pow(1*pi/180,2)]))
        #         self.RTrue = scipy.diag([sigmar,sigmao])
        self.RTrue = mat(scipy.diag([pow(2.0,2),pow(3*pi/180,2)]))

        #
        self.UEst = self.UTrue
        #
        self.REst = self.RTrue

        self.xTrue = mat([[1],[-40],[-pi/2]])
        self.xOdomLast = self.GetOdometry(1)

    #    initial conditions:
        self.xEst = self.xTrue
        self.PEst = scipy.diag([1,1,pow((1*pi/180),2)])

    #      storage
        self.InnovStore = []
```

```
        self.SStore = []
        self.PStore = []
        #Estimated Values
        self.XStore = []
        self.YStore = []

#     Ground truth sensor position
        self.YGround = []
        self.XGround = []
        self.OGround = []

#     store the feature position
        self.MapXStore = []
        self.MapYStore = []

#    store the robot position
        self.XErrStore = []
        self.YErrStore = []
        self.OErrStore = []

#================================================================================
====
# Functions for the Jacobian calculations
#================================================================================
====

    def J1(self, x1, x2):

        s1 = sin(x1[2,0])
        c1 = cos(x1[2,0])
        Jac  = mat([[1, 0, -x2[0,0]*s1-x2[1,0]*c1], [0, 1, x2[0,0]*c1-
x2[1,0]*s1],[0, 0, 1]])

        return Jac

    def J2(self,x1,x2):

        s1 = sin(x1[2,0])
        c1 = cos(x1[2,0])
        Jac  = mat([[c1,-s1,0],[s1,c1,0],[0,0,1]])

        return Jac

#================================================================================
====
# Inverse Composition functions
#================================================================================
====

    def tinv(self,tab):

        tba = mat([[0],[0],[0]])
        tba = self.tinv1(tab)

        return tba

    def tinv1(self,tab):

        s = sin(tab[2,0])
```

```
        c = cos(tab[2,0])
        aux = -tab[0,0]*c - tab[1,0]*s
        aux2 = tab[0,0]*s - tab[1,0]*c
        aux3 = -tab[2,0]
        tba = mat([[aux],[aux2],[aux3]])

        return tba

#================================================================================
====
# Composition function
#================================================================================
====
    def tcomp(self, tab, tbc):

        if (len(tab) != 3):
            print 'TCOMP: tab is not a transformation!!!'

        if (len(tbc) != 3):
            print 'TCOMP: tbc is not a transformation!!!'

        result = tab[2,0]+tbc[2,0]

        if (result > pi or result <= -pi):
            result = self.AngleWrap(result)

        s = sin(tab[2,0])
        c = cos(tab[2,0])
        aux = mat([[c, -s], [s, c]])
        aux2 = tab[0:2,0]+ aux*tbc[0:2,0]
        tac = mat([[aux2[0,0]], [aux2[1,0]],[result]])

        return tac

#================================================================================
====
# Function for fixing angle problems
#================================================================================
====
    def AngleWrap(self,a):

        if(a > pi):
            a = a - 2 * pi

        elif(a < -pi):
            a = a + 2 * pi

        return a

#================================================================================
====
# Dead Reckoning main calculations
#================================================================================
====
    def EKFSLAM(self):

        for i in range(2,self.nSteps):
        #    do world iteration
            self.SimulateWorld(i)
```

```
        #    figure out control
            xOdomNow = self.GetOdometry(i)
            u = self.tcomp(self.tinv(self.xOdomLast),xOdomNow)
            self.xOdomLast = xOdomNow

        #    do prediction
            xPred = self.tcomp(self.xEst,u)
            xPred[2,0] = self.AngleWrap(xPred[2,0])
            PPred =  self.J1(self.xEst,u)* self.PEst
*transpose(self.J1(self.xEst,u)) +self.J2(self.xEst,u)* self.UEst *
transpose(self.J2(self.xEst,u))

        #    observe a randomn feature
            [z,iFeature] = self.GetObservation(i)

            if(len(z) != 0):
        #        predict observation
                zPred = self.DoObservationModel(xPred,iFeature,self.Map)

        #        get observation Jacobian
                jH = self.GetObsJac(xPred,iFeature,self.Map)

        #        do Kalman update:
                Innov = z - zPred
                Innov[1,0] = self.AngleWrap(Innov[1,0])

                S = jH*PPred*transpose(jH)+self.REst
                W = PPred*transpose(jH)*numpy.linalg.inv(S)
                self.xEst = xPred + W*Innov
                self.xEst[2,0] = self.AngleWrap(self.xEst[2,0])

        #       note use of 'Joseph' form which is numerically stable
                I = eye(3)
                self.PEst = (I-W*jH)*PPred*transpose((I-W*jH))+
W*self.REst*transpose(W)
                self.PEst = 0.5*(self.PEst+transpose(self.PEst))

            else:
        #        There was no observation available
                self.xEst = xPred
                self.PEst = PPred
                Innov = mat([[0],[0]])
                S = eye(2)
            #    store results:
            self.InnovStore.append(Innov)
            self.PStore.append(sqrt(diag(self.PEst)))
            self.SStore.append(sqrt(diag(S)))
            #Estimated Values
            self.XStore.append(self.xEst[0,0])
            self.YStore.append(self.xEst[1,0])
            #        Groundtruth sensor position
            self.XGround.append(self.xTrue[0,0])
            self.YGround.append(self.xTrue[1,0])
            self.OGround.append(self.xTrue[2,0])
            # store the feature position
            self.MapXStore.append(self.Map[0,iFeature])
            self.MapYStore.append(self.Map[1,iFeature])
            #    store the robot position
            self.XErrStore.append(self.xTrue[0,0]-self.xEst[0,0])
```

```
            self.YErrStore.append(self.xTrue[1,0]-self.xEst[1,0])
            self.OErrStore.append(self.xTrue[2,0]-self.xEst[2,0])

#==========================================================================
====
# Observation Model
#==========================================================================
====
    def GetObservation(self, k):
        # fake sensor failure here
        if(abs(k-self.nSteps/2)<0.1*self.nSteps):
            z = []
            iFeature = -1
        else:
            iFeature = ceil((size(self.Map[1])-1)*rand(1))
            iFeature = (int)(iFeature)
            z = self.DoObservationModel(self.xTrue,
iFeature,self.Map)+sqrt(self.RTrue)*mat(randn(2,1))
            z[1,0] = self.AngleWrap(z[1,0])

        return z, iFeature


    def DoObservationModel(self,xVeh,iFeature, Map):

        riE = math.sqrt(pow((xVeh[0,0] - Map[0,iFeature]),2) + pow((xVeh[1,0]
- Map[1,iFeature]),2))
        oiE = atan2((Map[1,iFeature]-xVeh[1,0]),(Map[0,iFeature]-xVeh[0,0]))
- xVeh[2,0]

        z = mat([[riE],[oiE]])

        return z

    def GetObsJac(self,xPred, iFeature,Map):

        jH = mat(zeros((2,3)))
        Delta = (Map[0:2,iFeature]-xPred[0:2])
        r = numpy.linalg.norm(Delta)
        jH[0,0] = -Delta[0,0] / r
        jH[0,1] = -Delta[1,0] / r
        jH[1,0] = Delta[1,0] / pow(r,2)
        jH[1,1] = -Delta[0,0] / pow(r,2)
        jH[1,2] = -1

        return jH

#==========================================================================
====
# Odometry Sensor readings for feeding the dead reckoning model
#==========================================================================
====
    def GetOdometry(self,k):

        if(len(self.LastOdom)==0):
            self.LastOdom = self.xTrue

        u = self.GetRobotControl(k)
        xnow =self.tcomp(self.LastOdom,u)
```

```
        uNoise = sqrt(self.UTrue)*mat(0.5*randn(3,1))
        xnow = self.tcomp(xnow,uNoise)
        self.LastOdom = xnow

        return xnow

    def SimulateWorld(self,k):

        u = self.GetRobotControl(k)
        self.xTrue = self.tcomp(self.xTrue,u)
        self.xTrue[2,0]=self.AngleWrap(self.xTrue[2,0])

    def GetRobotControl(self,k):

        Vx = 0
        Vy = 0.5
        phi = 0.2*pi/180
        #phi = 0.1*pi/180*sin(3*pi*k/self.nSteps)
        u = mat([[Vx], [Vy] ,[phi]])

        return u

    def plot(self):

#Errors on the robot pose

        pylab.figure()

        pylab.subplot(311)
        pylab.plot(self.XErrStore, 'r-',label='Error on the X Measurement')
        pylab.xlabel('x Error' )
        pylab.ylabel('')
        pylab.legend()
        pylab.grid(True)

        pylab.subplot(312)
        pylab.plot(self.YErrStore, 'r-',label='Error on the Y Measurement')
        pylab.xlabel('Y Error')
        pylab.ylabel('')
        pylab.legend()
        pylab.grid(True)

        pylab.subplot(313)
        pylab.plot(self.OErrStore, 'r-',label='Error on the Theta
Measurement')
        pylab.xlabel('Theta Error')
        pylab.ylabel('')
        pylab.legend()
        pylab.grid(True)

__#    Comparison between the estimated and real trajectory of the vehicle
        pylab.figure()

        pylab.plot(self.XGround, self.YGround, 'r-',label='Analysis of the
Real Value')
        pylab.plot(self.XStore, self.YStore,'g-',label='Analysis of the
Estimated Value')
        pylab.xlabel('x Error' )
        pylab.ylabel('')
```

```
        pylab.grid(True)

__#    Measurements

        pylab.figure()
        pylab.subplot(311)
        pylab.plot(self.XGround,   'r-',label='x Measurement')
        pylab.xlabel('x' )
        pylab.ylabel('')
        pylab.grid(True)

        pylab.subplot(312)
        pylab.plot(self.YGround, 'r-',label='Y Measurement')
        pylab.xlabel('Y')
        pylab.ylabel('')
        pylab.legend()
        pylab.grid(True)

        pylab.subplot(313)
        pylab.plot(self.OGround, 'r-',label='Theta Measurement')
        pylab.xlabel('Theta')
        pylab.ylabel('')
        pylab.legend()
        pylab.grid(True)

# Feature Estimation
        pylab.figure()

        pylab.plot(self.MapXStore,self.MapYStore,  'b+',label='Feature
Position')
        pylab.xlabel('X' )
        pylab.ylabel('Y')
        pylab.grid(True)

# Show graphics
        pylab.show()

teste = EKFLocalisation()
teste.EKFSLAM()
teste.plot()
```