Universidade Federal de Juiz de Fora

Engenharia Elétrica

Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial

**Gustavo Hofstätter**

# Filtro de Kalman aplicado à localização de robôs móveis

Juiz de Fora

2016

**Gustavo Hofstätter**

# Filtro de Kalman aplicado à localização de robôs móveis

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial da Universidade Federal de Juiz de Fora, na área de concentração em Localização de Robôs Móveis , como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Guilherme Márcio Soares, M. Eng.

Juiz de Fora

2016

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Hofstätter, Gustavo.
Filtro de Kalman aplicado à localização de robôs móveis / Gustavo Hofstätter. – 2016.
68 f. : il.

Orientador: Prof. Guilherme Márcio Soares, M. Eng.
Trabalho de Conclusão de Curso (Graduação) – Universidade Federal de Juiz de Fora, Engenharia Elétrica. Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial, 2016.

1. Filtro de Kalman. 2. Localização. 3. Robótica Móvel. I. Márcio Soares, Guilherme, orient. II. Me. Eng.

**Gustavo Hofstätter**

**Filtro de Kalman aplicado à localização de robôs móveis**

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Engenharia Elétrica com habilitação nas áreas de Robótica e Automação Industrial da Universidade Federal de Juiz de Fora, na área de concentração em Localização de Robôs Móveis , como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

---

Prof. Guilherme Márcio Soares, M. Eng. - Orientador
Universidade Federal de Juiz de Fora

---

Prof. Ana Sophia Cavalcanti Alves Vilas Boas, D. Eng.
Universidade Federal de Juiz de Fora

---

Prof. Exuperry Barros Costa, M. Eng.
Universidade Federal de Juiz de Fora

## AGRADECIMENTOS

Sem dúvidas as primeiras pessoas que eu tenho que agradecer são meus pais. Realmente agradeço por todo o apoio que sempre me deram durante minha vida inteira. Eu não teria me formado tão cedo e de forma tão completa se não fosse pela ajuda deles, e por isso eles merecem um lugar especial aqui nos meus agradecimentos.

Agradeço ao PET Elétrica por ter contribuindo tanto no meu crescimento pessoal, acadêmico e profissional. Não teve lugar melhor na engenharia para desenvolver boas amizades e habilidades transversais. Equilíbrio perfeito entre ambiente de trabalho e ambiente familiar. Com certeza vai deixar saudades.

Agradeço aos meus ~~colegas~~ companheiros Frederick Tavares e Pedro Alcântara, que me acompanharam nessa luta que foi se formar em Engenharia Elétrica. Juntos, ajudando um ao outro, formamos uma equipe que foi capaz de superar de forma invicta os desafios desse curso que sempre foi recordista em reprovações.

Agradeço também a equipe Rinobot por todas as experiências que tive na equipe. Sempre tive uma paixão pela robótica e ter tido a oportunidade de participar de uma competição latino-americana de robótica foi realizador.

Agradeço ao meu orientador Guilherme Márcio por todo o esforço, dedicação e paciência que teve em me ajudar a fazer este trabalho de conclusão de curso.

“Faça as coisas o mais simples que puder, porém não as mais simples.”
(Albert Einstein)

**RESUMO**

Este trabalho aborda o estudo e implementação do filtro de Kalman na localização de robôs móveis. A localização do robô é obtida, primeiramente, com processamento das imagens obtidas por uma câmera posicionada sobre o ambiente de ação do robô. Para deixar a localização do robô mais confiável, essas informações obtidas pela câmera são combinadas, por meio do filtro de Kalman, com as informações retornadas por um modelo cinemático discreto que é atualizado com as medidas dos encoders das rodas do robô. Para tal, foi utilizado o mesmo robô, campo e câmera utilizados em futebol de robôs da categoria IEEE Very Small Size Soccer, concedidos pela equipe de futebol de robôs autônomos da UFJF, **Rinobot Team**, e a implementação deste trabalho deve ser futuramente utilizada por esta equipe.

Palavras-chave: Filtro de Kalman. Localização. Robótica Móvel.

**ABSTRACT**

This work addresses the study and implementation of the Kalman filter in the location of mobile robots. The location of the robot is obtained, first, with the processing of the images obtained by a camera positioned on the environment of action of the robot. To make the location of the robot more reliable, this information obtained by the camera is combined through the Kalman filter with the information returned by a discrete kinematic model that is updated with the measurements of the robot's wheel encoders. For that, the same robot, field and camera used in robots of the category IEEE Very Small Size Soccer were used, granted by the autonomous robots soccer team of the UFJF, Rinobot Team, and the implementation of this work should be used in the future by this team.

Key-words: Kalman filter. Location. Mobile Robotics.

## LISTA DE ILUSTRAÇÕES


Figura 1 – Robô diferencial [3]. . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
Figura 2 – Exemplo de movimentação do modelo cinemático [3]. . . . . . . . . . . 16
Figura 3 – Gráfico da distribuição de probabilidade de uma gaussiana [7]. . . . . . 19
Figura 4 – Gráfico da distribuição de probabilidade com 1 dado. . . . . . . . . . . 19
Figura 5 – Gráfico da distribuição de probabilidade com 2 dados. . . . . . . . . . 20
Figura 6 – Gráfico da distribuição de probabilidade com 4 dados. . . . . . . . . . 20
Figura 7 – Fluxograma do EKF. . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
Figura 8 – Fluxograma do código principal. . . . . . . . . . . . . . . . . . . . . . 29
Figura 9 – Robô com tag. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
Figura 10 – Coordenadas do robô . . . . . . . . . . . . . . . . . . . . . . . . . . 32
Figura 11 – Vista frontal e superior do robô. . . . . . . . . . . . . . . . . . . . . . 33
Figura 12 – Fluxograma da obtenção da leitura dos encoders do robô. . . . . . . . 34
Figura 13 – Coordenadas do robô . . . . . . . . . . . . . . . . . . . . . . . . . . 36
Figura 14 – Teste com o robô parado. . . . . . . . . . . . . . . . . . . . . . . . . 37
Figura 15 – Gráficos das variáveis de estado do teste do robô parado. . . . . . . . . 38
Figura 16 – Frames do teste com o robô andando em linha reta. . . . . . . . . . . . 39
Figura 17 – Gráficos das variáveis de estado do teste do robô andando em linha reta. 39
Figura 18 – Frames do teste com o robô andando em círculos. . . . . . . . . . . . . 40
Figura 19 – Gráficos das variáveis de estado do teste do robô andando em círculos. 41
Figura 20 – Frames obtidos durante o teste de robustez. . . . . . . . . . . . . . . . 42
Figura 21 – Resultados com o teste de robustez . . . . . . . . . . . . . . . . . . . . 42

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| UFJF | Universidade Federal de Juiz de Fora |
| FDP | Função de Distribuição de Probabilidade |
| IEEE | Instituto de Engenheiros Eletricistas e Eletrônicos |
| EKF | Filtro de Kalman Estendido |
| $\bar{x}$ | : Estado do robô |
| $x$ | : Posição x do robô no plano cartesiano |
| $y$ | : Posição y do robô no plano cartesiano |
| $\theta$ | : Orientação do robô |
| $v$ | : Velocidade linear |
| $\omega$ | : Velociade rotacional (também chamada de velocidade angular) |
| $v_r$ | : Velocidade da roda direita |
| $v_l$ | : Velocidade da roda esquerda |
| $b$ | : Distância entre as rodas item[$\Delta s$] : Deslocamento linear |
| $\Delta\theta$ | : Deslocamento angular |
| $\Delta t$ | : Tempo decorrido com essas velocidades |
| $\bar{x}^-$ | : Estado a *Priori* |
| $A$ | : Matriz de Transição de Estado |
| $\bar{x}^+$ | : Estado a *Posteriori* |
| $P^-$ | : Covariância de Estado a *Priori* |
| $P^+$ | : Covariância de Estado a *Posteriori* |
| $C$ | : Matriz de mapeamento de $Q$ para $\bar{x}^+$ |
| $Q$ | : Covariância de Transição de Estado |
| $K$ | : Constante de Kalman |
| $R$ | : Covariância do Sensor |
| $H$ | : Matriz de mapeamento do estado real para $z$ |

$z$ : Leitura do Sensor

$F_p$ : Matriz de transição de estados linearizada

$F_{\Delta_{rl}}$ : Matriz de mapeamento de Q linearizada

$k_r$ : Constante de multiplicação do deslocamento da roda direita

$k_l$ : Constante de multiplicação do deslocamento da roda esquerda

$\Delta s_r$ : Deslocamento da roda direita do robô

$\Delta s_l$ : Deslocamento da roda esquerda do robô

# SUMÁRIO


**1 Introdução . . . . . 12**
1.1 Futebol de robôs . . . . . 12
1.2 Objetivo . . . . . 12
1.3 Organização . . . . . 12

**2 Modelo Cinemático de Robô Diferencial . . . . . 14**

**3 Filtro de Kalman . . . . . 18**
3.1 Variáveis Aleatórias Gaussianas . . . . . 18
3.2 Filtro de Kalman unidimencional . . . . . 21
3.3 Localização utilizando o Filtro de Kalman . . . . . 21
3.3.1 Etapa de predição . . . . . 22
3.3.2 Etapa de atualização . . . . . 23
3.4 Filtro de Kalman Estendido . . . . . 24
3.4.1 Cálculo do estado a *Priori* . . . . . 25
3.4.2 Cálculo da covariância do estado a *Priori* . . . . . 26
3.4.3 Cálculo do ganho de Kalman e estado a *Posteriori* . . . . . 28

**4 Implementação . . . . . 29**
4.1 Visão computacional . . . . . 30
4.2 Implementação Hardware/Software robô . . . . . 32
4.3 EKF . . . . . 34
4.4 Ajuste dos parâmetros do EKF . . . . . 35
4.5 Resultados . . . . . 36
4.6 Problemas de implementação . . . . . 43

**5 Conclusão . . . . . 44**
5.1 Comentários adicionais . . . . . 44
5.2 Trabalhos futuros . . . . . 44

**REFERÊNCIAS . . . . . 45**

**APÊNDICE A – Código para encontrar a faixa de valores de cada cor . . . . . 46**

**APÊNDICE B – Código para encontrar extremidades do campo na imagem . . . . . 50**

**APÊNDICE C – Código completo** . . . . . . . . . . . . . . . . . **52**

**APÊNDICE D – Código do arduino do robô** . . . . . . . . . . . **57**

**APÊNDICE E – Código para medir covariância da câmera** . **64**

# 1 Introdução

A robótica móvel é uma área promissora da engenharia que está em constante crescimento e evolução. Fica cada vez mais comum o uso de drones e de carros autônomos, por exemplo.

Todavia, para que um robô móvel consiga executar suas tarefas de maneira bem sucedida, ele precisa saber sua localização no ambiente.

A localização de robôs móveis é uma das mais desafiadoras competências da robótica. São necessários métodos de localização eficientes para interpretar as informações dos sensores extraindo dados de localização mais próximos possíveis do estado real do robô.

Uma plataforma para estudos acerca de técnicas de localização de robôs móveis é o futebol de robôs.

## 1.1 Futebol de robôs

A UFJF possui uma equipe de competição de robótica chamada Rinobot. Uma das categorias em que a Rinobot compete é a de futebol de robôs IEEE Very Small Size Soccer. Na categoria de futebol de robôs IEEE Very Small Size Soccer duas equipes de 3 robôs de até 7,5×7,5×7,5 cm realizam um jogo de futebol. Os robôs são controlados por um computador, mas sem intervenção humana. O computador processa a imagem de uma câmera de vídeo colocada acima do campo e comanda os robôs.

Conhecer a posição e orientação dos robôs é essencial para executar qualquer estrategia de jogo. Deste modo, através da coleta e do processamento das imagens da câmera é possível determinar a posição dos robôs e da bola. Entretanto, a câmera, como qualquer outro sensor, realiza medidas com imperfeições.

Para contornar esses erros de medida e obter a localização do robô com mais precisão, é possível utilizar o mesmo método utilizado pela NASA no Programa Apollo: filtrar os erros usando o filtro de Kalman [18].

## 1.2 Objetivo

O objetivo geral desse trabalho foi estudar e implementar o Filtro de Kalman e deixar uma contribuição para a Equipe de Futebol de Robôs da UFJF - Rinobot.

## 1.3 Organização

A organização do restante deste documento é descrita a seguir. O capítulo 2 aborda a modelagem matemática de um robô diferencial. Por outro lado, o capítulo 3 apresenta

uma breve revisão sobre variáveis aleatórias gaussianas e explica o funcionamento do filtro de Kalman, bem como sua implementação na localização de robôs móveis.

O capítulo 4 descreve a implementação realizada e os resultados obtidos com e sem o filtro de Kalman.

Por fim, as conclusões do trabalho são apresentadas no capítulo 5.

## 2 Modelo Cinemático de Robô Diferencial

O estado de um robô diferencial depende de 3 parâmetros: sua posição $x$, sua posição $y$ e sua orientação $\theta$ no plano cartesiano.

$$\bar{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \tag{2.1}$$

$\bar{x}$ : Estado do robô

$x$ : Posição x do robô no plano cartesiano

$y$ : Posição y do robô no plano cartesiano

$\theta$ : Orientação do robô

A modelagem descrita neste documento considera a implementação discreta do sistema. Neste contexto, o subescrito $k$ representa o k-ésimo estado da variável e, por conseguinte, $k-1$ representa o estado anterior.

Esta notação será utilizada no restante do documento. Também foi considerado que o cálculo dos estado do robô foi feito de maneira recursiva, isto é, o estado calculado no momento atual será utilizado como estado inicial na próxima interação.

O giro das rodas do robô gera uma velocidade linear e rotacional conforme observado na Figura 1 [2].

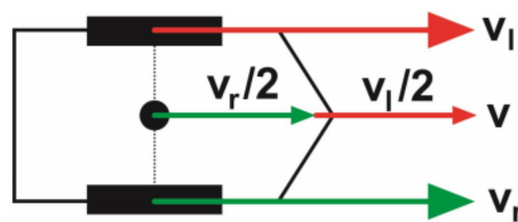


Figura 1 – Robô diferencial [3].

$$v = \frac{(v_r + v_l)}{2} \tag{2.2}$$

$$\omega = \frac{(v_r - v_l)}{b}, \tag{2.3}$$

em que:

$v$ : Velocidade linear

$\omega$ : Velociade rotacional (também chamada de velocidade angular)

$v_r$ : Velocidade da roda direita

$v_l$ : Velocidade da roda esquerda

$b$ : Distância entre as rodas

A velocidade linear é igual a média da velocidade das rodas e a velocidade rotacional é igual a diferença de velocidade das rodas dividida pela distância entre elas.

Observe que se essas velocidades forem mantidas constantes por uma quantidade de tempo $\Delta t$, geram um deslocamento linear e rotacional.

$$\Delta s = v\Delta t \tag{2.4}$$

$$\Delta\theta = \omega\Delta t, \tag{2.5}$$

em que:

$\Delta s$ : Deslocamento linear

$\Delta\theta$ : Deslocamento angular

$v$ : Velocidade linear

$\omega$ : Velociade angular

$\Delta t$ : Tempo decorrido com essas velocidades

A Figura 2 mostra o estado do robô antes e após a movimentação.

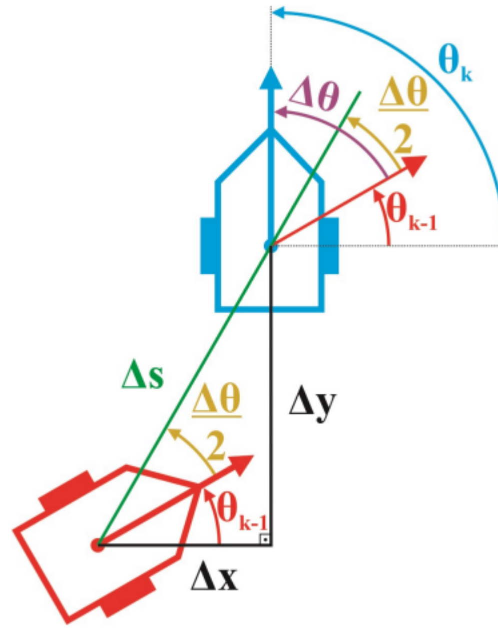


Figura 2 – Exemplo de movimentação do modelo cinemático [3].

O Estado do robô após essa movimentação pode ser calculado conforme na figura 2.6.

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}, \tag{2.6}$$

em que:

$x$ : Posição x do robô no plano cartesiano

$y$ : Posição y do robô no plano cartesiano

$\theta$ : Orientação do robô

$\Delta x$ : Deslocamento $x$ do robô no plano cartesiano

$\Delta y$ : Deslocamento $y$ do robô no plano cartesiano

$\Delta\theta$ : Deslocamento rotacional do robô

Substituindo os valores de $\Delta x$, $\Delta y$ e $\Delta\theta$ de acordo com a Figura 2, temos a equação 2.7.

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \Delta s.cos(\theta_{k-1} + \frac{\Delta\theta}{2}) \\ \Delta s.sen(\theta_{k-1} + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix} \tag{2.7}$$

$$\bar{x}_k = \bar{x}_{k-1} + \begin{bmatrix} \Delta s.cos(\theta_{k-1} + \frac{\Delta\theta}{2}) \\ \Delta s.sen(\theta_{k-1} + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix}, \tag{2.8}$$

em que:

$\bar{x}$ : Estado do robô

$x$ : Posição x do robô no plano cartesiano

$y$ : Posição y do robô no plano cartesiano

$\theta$ : Orientação do robô

$\Delta x$ : Deslocamento x do robô no plano cartesiano

$\Delta y$ : Deslocamento y do robô no plano cartesiano

$\Delta\theta$ : Deslocamento rotacional do robô

$\Delta s$ : Deslocamento linear do robô

Com essas informações, é possível calcular a transição de estados do robô, se tiver em mãos o deslocamento linear e angular do robô.

Encoders acoplados às rodas do robô retornam o valor da velocidade de cada roda e, através do cálculo das equações 2.4 e 2.5, é obtido a velocidade linear e angular do robô.

# 3 Filtro de Kalman

Esse capítulo foi baseado em [1] e [2].

Em 1950, Rudolph Emil Kalman inventou o filtro de Kalman, técnica de predição e filtragem de sistemas lineares e contínuos.

O filtro de Kalman basicamente realiza uma fusão matemática entre dois ou mais sensores para obter uma medida mais precisa, tentando mitigar os erros aleatórios e incertezas de leitura dos sensores.

Dentre as aplicações, uma das formas mais usadas é na localização de veículos e sistemas móveis. Tem sido vital nos sistemas de navegação e guiamento de submarinos, mísseis de cruzeiro, ônibus espaciais (com destaque para o Projeto Apollo da NASA) e para a Estação Espacial Internacional [13].

A incerteza de alguns sensores pode ser modelada como uma gaussiana e isso é fundamental para o filtro de Kalman, pois o mesmo admite que as entradas do sistema são gaussianas e que o sistema é linear, portanto a saída do filtro também vai ser gaussiana.

Portanto, para entender como o filtro de Kalman funciona, primeiro é necessário entender como se comporta uma variável aleatória gaussiana.

## 3.1 Variáveis Aleatórias Gaussianas

De acordo com o Teorema do Limite Central a função de distribuição de probabilidade da soma um grande número de variáveis aleatórias aproxima-se de uma gaussiana [4]. Uma Função de Distribuição de Probabilidade (FDP) descreve o comportamento de uma variável aleatória, portanto, uma variável aleatória gaussiana tem sua FDP descrita pela função [4]:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\bar{x})^2}{2\sigma^2}}, \tag{3.1}$$

em que:

$\bar{x}$ : média

$\sigma^2$ : variância

$f(x)$ : probabilidade da variável aleatória gaussiana retornar o valor x

A FDP de uma gaussiana é graficamente representada na Figura 3 [7].

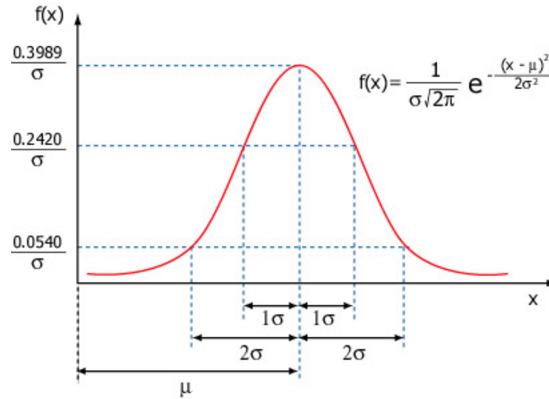


Figura 3 – Gráfico da distribuição de probabilidade de uma gaussiana [7].

Variáveis aleatórias gaussianas aparecem em praticamente todas as áreas da ciência e engenharia. Muitos fenômenos aleatórios se comportam próximos a essa distribuição, tais fenômenos como altura, pressão sanguínea e peso de uma população por exemplo.

Um modo de observar o Teorema do Limite Central na prática é pelo exemplo do lançamento de dados não viciados.

Imagine dados não viciados sendo jogados em uma mesa. O valor de cada dado é uma variável aleatória e o valor da soma desses dados é uma variável aleatória também, e é resultante da soma dessas outras variáveis aleatórias (o valor dos dados). Jogando apenas 1 dado, a FDP do valor do dado na mesa é representado na Figura 4.

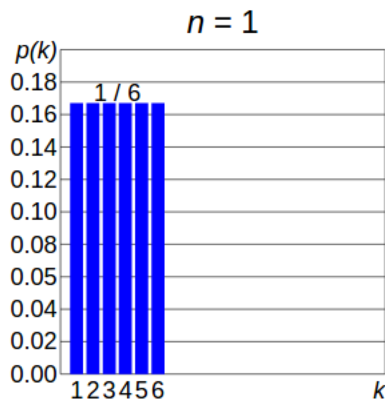


Figura 4 – Gráfico da distribuição de probabilidade com 1 dado.

A FDP é uniforme pois todos os valores tem a mesma probabilidade de aparecer. Entretanto, se forem jogados 2 dados, a FDP da soma dos dados da mesa fica conforme mostrado na Figura 5.

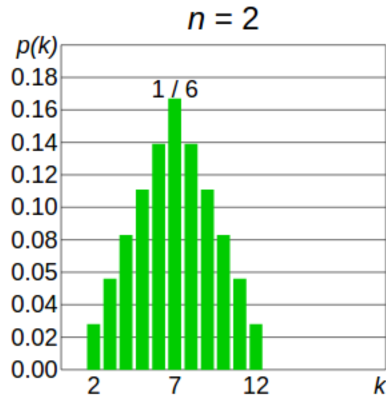


Figura 5 – Gráfico da distribuição de probabilidade com 2 dados.

Observe que mesmo que a FDP de cada dado não se pareça em nada com uma gaussiana, a PDF da soma desses 2 dados já começa a tomar forma similar a FDP gaussiana. E quanto maior o número de variáveis aleatórias somadas, mais o comportamento fica parecido com o de uma gaussiana, como é possível observar na FDP do lançamento de 4 dados na Figura 6.

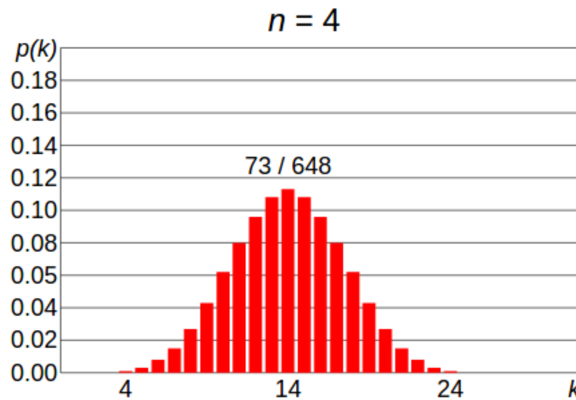


Figura 6 – Gráfico da distribuição de probabilidade com 4 dados.

Isso acontece com a soma de variáveis aleatórias não uniformes também, por isso podemos aproximar os erros de leitura de alguns sensores como uma variável aleatória gaussiana [4].

É possível caracterizar uma gaussiana a partir de dois parâmetros: média e variância, os quais são utilizados pelo filtro de Kalman para a estimação dos estado. É importante ressaltar que o estado estimado pelo filtro de Kalman é representado por uma gaussiana. Deste modo o valor do filtro não é um valor absoluto, mas sim os parâmetros média e variância. Por isso o filtro de Kalman é um filtro paramétrico.

É importante ressaltar também que o filtro de Kalman foi desenvolvido para sistemas lineares [1], uma vez em tais sistemas, se a entrada for uma gaussiana, a saída também será. Tal característica não se aplica a sistemas não lineares.

## 3.2 Filtro de Kalman unidimensional

Suponha um sistema real e linear. O estado atual deste sistema é $x_R$.

Medindo este estado com um sensor, é obtido um valor muitas vezes impreciso. Sensores mais confiáveis são geralmente mais caros e vice versa. Portando a medida do sensor será modelada como uma gaussiana com média $\bar{x_s}$ e variância $\sigma_s^2$ já conhecida.

É feito então um modelo do sistema real e com esse modelo é feita uma previsão $\bar{x_m}$ de qual seria o estado real do sistema. Esse modelo também não é perfeito e possui um variância $\sigma_m^2$.

O filtro de Kalman faz uma média ponderada entre o sensor e o modelo, obtendo uma média mais próxima do valor real e uma variância menor.

$$\bar{x_k} = \frac{\sigma_s^2}{\sigma_m^2 + \sigma_s^2}\bar{x_m} + \frac{\sigma_m^2}{\sigma_m^2 + \sigma_s^2}\bar{x_s} \tag{3.2}$$

$$\sigma_k^2 = \frac{\sigma_m^2 \sigma_s^2}{\sigma_m^2 + \sigma_s^2}, \tag{3.3}$$

em que:

$\bar{x_k}$ : Média do filtro de Kalman

$\bar{x_m}$ : Média do modelo

$\bar{x_s}$ : Média do sensor

$\sigma_k^2$ : Variância do filtro de Kalman

$\sigma_m^2$ : Variância do modelo

$\sigma_s^2$ : Variância do sensor

Observe que a variância do filtro de Kalman sempre vai ser menor que a variância do modelo e do sensor, garantindo sempre resultados mais precisos.

Este é o filtro de Kalman. Neste trabalho o filtro de Kalman será implementado na localização de um robô móvel. Por isso será trabalhado com múltiplas variáveis e com covariância, que nada mais é que uma matriz de variâncias correlacionadas.

## 3.3 Localização utilizando o Filtro de Kalman

A localização por filtro de Kalman tem duas etapas: predição e atualização. Na etapa de predição é utilizado o modelo para realizar uma previsão de qual seria o estado real do robô. Por outro lado, na etapa de atualização, é comparado a confiabilidade do modelo e do sensor, em seguida é atualizado o estado do filtro utilizando a leitura do sensor principal e o valor da previsão retornada pelo modelo.

### 3.3.1 Etapa de predição

Na primeira etapa é feita a predição do estado.

$$\bar{x}_k^- = A.\bar{x}_{k-1}^+ \tag{3.4}$$

$$P_k^- = A.P_{k-1}^+.A^T + C.Q.C^T, \tag{3.5}$$

em que:

$\bar{x}^-$ : Estado a *Priori*

$A$ : Matriz de Transição de Estado

$\bar{x}^+$ : Estado a *Posteriori*

$P^-$ : Covariância de Estado a *Priori*

$P^+$ : Covariância de Estado a *Posteriori*

$C$ : Matriz de mapeamento de $Q$ para $\bar{x}^+$

$Q$ : Covariância de Transição de Estado

O estado a *Posteriori* faz parte do resultado anterior do próprio filtro de Kalman. Lembrando que o filtro de Kalman é recursivo e trabalha com os parâmetros média e variância. A gaussiana resultante da etapa de predição é caracterizada pelas variáveis $\bar{x}^+$ e $P^+$.

A matriz $A$ é a matriz de transição de estado. Geralmente é gerada a partir da leitura de outros sensores para fazer uma previsão de qual é o estado atual.

O resultado dessa multiplicação matricial é chamado de Estado a *Priori* ($\bar{x}^-$). Representa a média da gaussiana que define a previsão do Estado Atual.

Da mesma forma $P^-$, chamada de covariância de estado a *Priori*, representa a covariância da gaussiana que define a previsão do estado atual.

Entretanto essa previsão não é perfeita e geralmente é feita com base em outros sensores que também podem conter erros aleatórios gaussianos. Por isso para calcular a transição da covariância é levado em consideração a matriz $Q$, que representa a covariância de transição de estado. $Q$ é o erro aleatório do modelo, dado pelo deslizamento das rodas, como será mostrado a seguir.

A matriz $C$ serve para deixar a matriz $Q$ no formato certo para realizar a soma, pois muitas vezes a matriz $A$ utiliza uma quantidade de sensores diferente da quantidade de sensores usados para fazer a leitura de Estado Atual. Por exemplo, no caso do robô diferencial, para realizar a transição de estados é necessário realizar a leitura de 2 encoders,

tendo então uma matriz de Covariância com dimensão $2 \times 2$ que é impossível de somar com uma matriz de dimensão $3 \times 3$. Para resolver este exemplo basta uma operação matricial para mapear os valores de $Q$ com uma matriz $C$ de dimensão $3 \times 2$.

$$\underbrace{CQC^T}_{3\times3} = \underbrace{C}_{3\times2} \cdot \underbrace{Q}_{2\times2} \cdot \underbrace{C^T}_{2\times3} \tag{3.6}$$

Assim, ao final desses cálculos, é obtido uma previsão da média e covariância do estado atual. Podendo então passar para a segunda etapa do filtro de Kalman.

### 3.3.2 Etapa de atualização

Na segunda etapa é feita a atualização do estado.

$$K_k = P_k^-.H^T.(H.P_k^-.H^T + R)^{-1} \tag{3.7}$$

$$P_k^+ = P_k^- - K_k.H.P_k^- \tag{3.8}$$

$$\bar{x}_k^+ = \bar{x}_k^- + K_k.[z_k - H.\bar{x}_k^-], \tag{3.9}$$

em que:

$\bar{x}^-$ : Estado a *Priori*

$\bar{x}^+$ : Estado a *Posteriori*

$P^-$ : Covariância de Estado a *Priori*

$P^+$ : Covariância de Estado a *Posteriori*

$K$ : Constante de Kalman

$R$ : Covariância do Sensor

$H$ : Matriz de mapeamento do estado real para $z$

$z$ : Leitura do Sensor

A matriz $R$ representa a covariância do sensor. Quanto maior seus valores, menos preciso é o sensor.

A constante de Kalman avalia a importância do sensor na saída do filtro de Kalman, ponderando sobre a covariância de estado a *Priori* e a covariância do sensor. Basicamente mensura a capacidade do Sensor Principal de produzir estados mais confiáveis do que os da etapa de predição. Observe na equação 3.7 que quanto maior forem os valores de $R$, menor vai ser o valor de $K$.

A leitura do sensor é o estado real multiplicado pela matriz $H$ mais um erro aleatório gaussiano.

$$z_k = H.x_k + ruido_gaussiano \tag{3.10}$$

Observe então que $H$ realiza o mapeamento do estado real para $z$. Isso é útil pois muitas vezes o valor retornado pelo sensor não representa o estado. Por exemplo, se a localização do robô for baseada em sensor do tipo *laserscan*, $z$ retorna o valor das medidas dos lasers e seria necessário uma matriz $H$ para mapear os valores do estado real para as medidas $z$.

Entretanto, no caso do futebol de robôs, o sensor principal (a câmera), depois de passar por um algorítimo de visão computacional, retorna diretamente os estado $x$, $y$ e $\theta$, por isso a matriz $H$ é apenas uma matriz identidade neste caso.

A covariância de estado a *Posteriori* é atualizada utilizando a covariância de estado a *Priori* e a constante de Kalman. Por outro lado o estado a *Posteriori* é atualizado comparando a leitura do sensor com o estado a *Priori*, multiplicando essa comparação pela constante de Kalman e somando com o estado a *Priori*.

Este é o método de localização utilizando o filtro de Kalman. Observe que se $R$ possuir valores extremamente altos, $K$ vai ter um valor próximo a zero e a saída do filtro de Kalman vai ser igual a $x^-$, ignorando o sensor. Por outro lado, se $R$ possuir valores próximos a zero, $K$ vai ter um valor próximo a 1 e o filtro de Kalman vai praticamente ignorar a previsão e confiar apenas no sensor. O filtro de Kalman está a cada interação atualizando tanto a média quanto a variância da gaussiana que representa seu estado atual, garantindo uma medida mais precisa do que apenas o sensor principal sozinho ou mesmo o modelo de transição.

Para aplicar o filtro de Kalman ao futebol de robôs, é utilizado os encoders das rodas para atualizar a matriz de transição de estado e a câmera como sensor principal.

O único prolema é que o filtro de Kalman é feito para sistemas lineares e o modelo cinemático do robô diferencial não é linear conforme mostrado no capítulo 2. E para contornar esse problema, é preciso linearizar a cinemática, derivando em cada ponto.

O filtro de Kalman Estendido (EKF) propõe a linearização do modelo de transição de estados e também do modelo do sensor, possibilitando a utilização do filtro de Kalman para a localização de robôs diferenciais.

## 3.4 Filtro de Kalman Estendido

A diferença entre o EKF e o Filtro de Kalman convencional está na forma de calcular $\bar{x}^-$, $P^-$ e $H$. No caso deste trabalho, a matriz $H$ já é linear.

A Figura 7 mostra o fluxograma do algorítimo EKF.

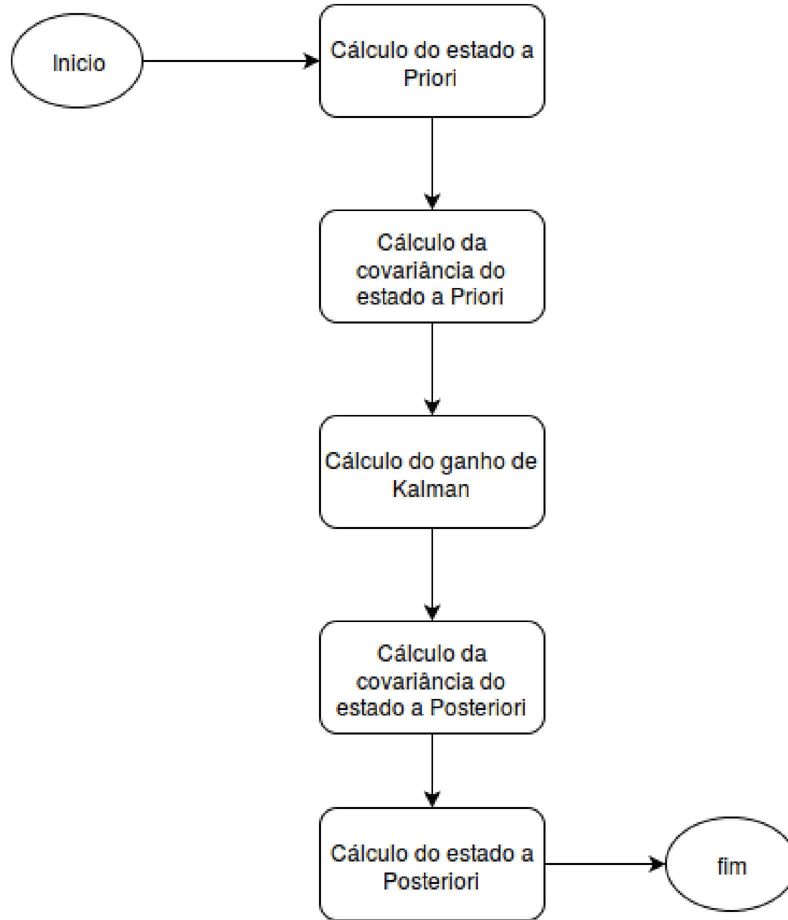


Figura 7 – Fluxograma do EKF.

### 3.4.1 Cálculo do estado a *Priori*

Na primeira etapa do EKF, a predição do Estado é feita levando em consideração o modelo cinemático do robô.

Para tal, é necessário saber o estado anterior $\bar{x}_{k-1}^{+}$ e o deslocamento linear $\Delta s_k$ e angular $\Delta\theta_k$ do robô.

Conforme mostrado no capítulo 2 a equação de transição de estados pode ser descrita por 3.11 [2], que pode ser simplificada, gerando 3.12.

$$\bar{x}_k^{-} = f(x, y, \theta, \Delta s, \Delta\theta) = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \Delta s_k cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ \Delta s_k sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ \Delta\theta_k \end{bmatrix} \tag{3.11}$$

$$\bar{x}_k^{-} = f(x, y, \theta, \Delta s, \Delta\theta) = \begin{bmatrix} x_{k-1} + \Delta s_k cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ y_{k-1} + \Delta s_k sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ \theta_{k-1} + \Delta\theta_k \end{bmatrix}, \tag{3.12}$$

em que:

$\bar{x}^-$ : Estado a *Priori*

$\bar{x}^+$ : Estado a *Posteriori*

$\Delta s$ : Deslocamento linear do robô adquirido com a leitura dos encoders

$\Delta\theta$ : Deslocamento rotacional do robô adquirido com a leitura dos encoders

$x_{k-1}$ : Coordenada cartesiana x do robô antes da etapa de predição

$y_{k-1}$ : Coordenada cartesiana y do robô antes da etapa de predição

$\theta_{k-1}$ : Orientação do robô antes da etapa de predição

### 3.4.2 Cálculo da covariância do estado a *Priori*

O cálculo da covariância do estado a *Priori* é feito conforme a equação 3.13 [2].

$$P_k^- = F_p.P_{k-1}^+.F_p^T + F_{\Delta_{rl}}.Q.F_{\Delta_{rl}}^T, \tag{3.13}$$

em que:

$P^-$ : Covariância de estado a *Priori*

$P^+$ : Covariância de estado a *Posteriori*

$F_p$ : Matriz de transição de estados linearizada

$Q$ : Covariância de Transição de Estado

$F_{\Delta_{rl}}$ : Matriz de mapeamento de Q linearizada

Como já mencionado anteriormente, o filtro de Kalman só trabalha com sistemas lineares. Para adaptar o filtro de Kalman para este sistema não linear, é necessário linearizar o mesmo. Isso pode ser feito derivando em cada ponto.

A fim de se calcular a matriz $F_p$, que representa a linearização da matriz de transição de estados, basta aplicar a Jacobiana na matriz de transição de estados. É importante ressaltar que a operação jacobiana é composta pelas derivadas parciais dos estados.

A equação 3.14 mostra o resultado desta manipulação matemática.

$$F_p = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s_k sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ 0 & 1 & \Delta s_k cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

O modelo de transição de estados pode não representar o verdadeiro movimento do robô por conta de deslizamentos das rodas no piso. O erro do modelo é então proporcional

a distância percorrida por cada roda [2]. Por isso a Covariância de Transição de Estado pode ser calculada conforme demonstrado na equação 3.15.

$$Q = \begin{bmatrix} k_r|\Delta s_{rk}| & 0 \\ 0 & k_l|\Delta s_{lk}| \end{bmatrix}, \tag{3.15}$$

em que:

$Q$ : Covariância de Transição de Estado

$k_r$ : Constante de multiplicação do deslocamento da roda direita

$k_l$ : Constante de multiplicação do deslocamento da roda esquerda

$\Delta s_r$ : Deslocamento da roda direita do robô

$\Delta s_l$ : Deslocamento da roda esquerda do robô

Observe que quanto maior for a distância percorrida por cada roda, mais o erro de leitura do encoder se acumula por conta de possíveis deslizamentos das rodas e maior vai ser a covariância de transição de estado.

A matriz $Q$ precisa ser mapeada conforme já demonstrado e este mapeamento precisa ser linearizado também.

Lembrando que:

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{3.16}$$

$$\Delta \theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{3.17}$$

Então a Jácobiana fica conforme a equação 3.19:

$$F_{\Delta_{rl}} = \begin{bmatrix} \frac{\partial f}{\partial \Delta s_r} & \frac{\partial f}{\partial \Delta s_l} \end{bmatrix} \tag{3.18}$$

$$F_{\Delta_{rl}} = \begin{bmatrix} \frac{1}{2}cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) - \frac{\Delta s_k}{2b}sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) & \frac{1}{2}cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) + \frac{\Delta s_k}{2b}sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ \frac{1}{2}sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) + \frac{\Delta s_k}{2b}cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) & \frac{1}{2}sen(\theta_{k-1} + \frac{\Delta\theta_k}{2}) - \frac{\Delta s_k}{2b}cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix} \tag{3.19}$$

Tendo em mãos a matriz de transição de estados linearizada, a covariância de transição de estados $Q$ e a matriz de mapeamento de $Q$ linearizada, é calculada a covariância de estado a *Priori*.

### 3.4.3 Cálculo do ganho de Kalman e estado a *Posteriori*

A etapa de atualização do EKF não possui alterações significativas em relação à implementação clássica do método de localização utilizado o filtro de Kalman.

$$K_k = P_k^-.H^T.(H.P_k^-.H^T + R)^{-1} \tag{3.20}$$

$$P_k^+ = P_k^- - K_k.H.P_k^- \tag{3.21}$$

$$\bar{x}_k^+ = \bar{x}_k^- + K_k.[z_k - H.\bar{x}_k^-] \tag{3.22}$$

$\bar{x}^-$ : Estado a *Priori*

$\bar{x}^+$ : Estado a *Posteriori*

$P^-$ : Covariância de Estado a *Priori*

$P^+$ : Covariância de Estado a *Posteriori*

$K$ : Constante de Kalman

$R$ : Covariância do Sensor

$H$ : Matriz de mapeamento de $\bar{x}^+$ para $z$

$z$ : Leitura do Sensor

Por fim, o algorítimo do EKF pode ser sintetizado através do pseudo-código mostrado [1].

**Algorithm 1** EKF

1: **function** EKF($\bar{x}_{k-1}^+$,$P_{k-1}^+$,$\Delta s_k$,$\Delta\theta_k$,$z_k$)
2: $\quad \bar{x}_k^- = f(\bar{x}_{k-1}^+, \Delta s, \Delta\theta)$
3: $\quad P_k^- = F_p.P_{k-1}^+.F_p^T + F_{\Delta_{rl}}.Q.F_{\Delta_{rl}}^T$
4: $\quad K_k = P_k^-.H^T.(H.P_k^-.H^T + R)^{-1}$
5: $\quad P_k^+ = P_k^- - K_k.H.P_k^-$
6: $\quad \bar{x}_k^+ = \bar{x}_k^- + K_k.[z_k - H.\bar{x}_k^-]$
7: **end function**

# 4 Implementação

Utilizando uma câmera, um robô diferencial e um computador, foi possível implementar o EKF para melhorar de forma considerável a precisão da localização do robô. Todo o programa do computador foi desenvolvido em Python, pois uma linguagem fácil de aprender, fácil de programar e extremamente funcional, atendendo às necessidades desse trabalho. Várias empresas usam essa linguagem de programação atualmente, com destaque para Google e YouTube [10].

A parte principal do código ficou de acordo com fluxograma mostrado na Figura 8.

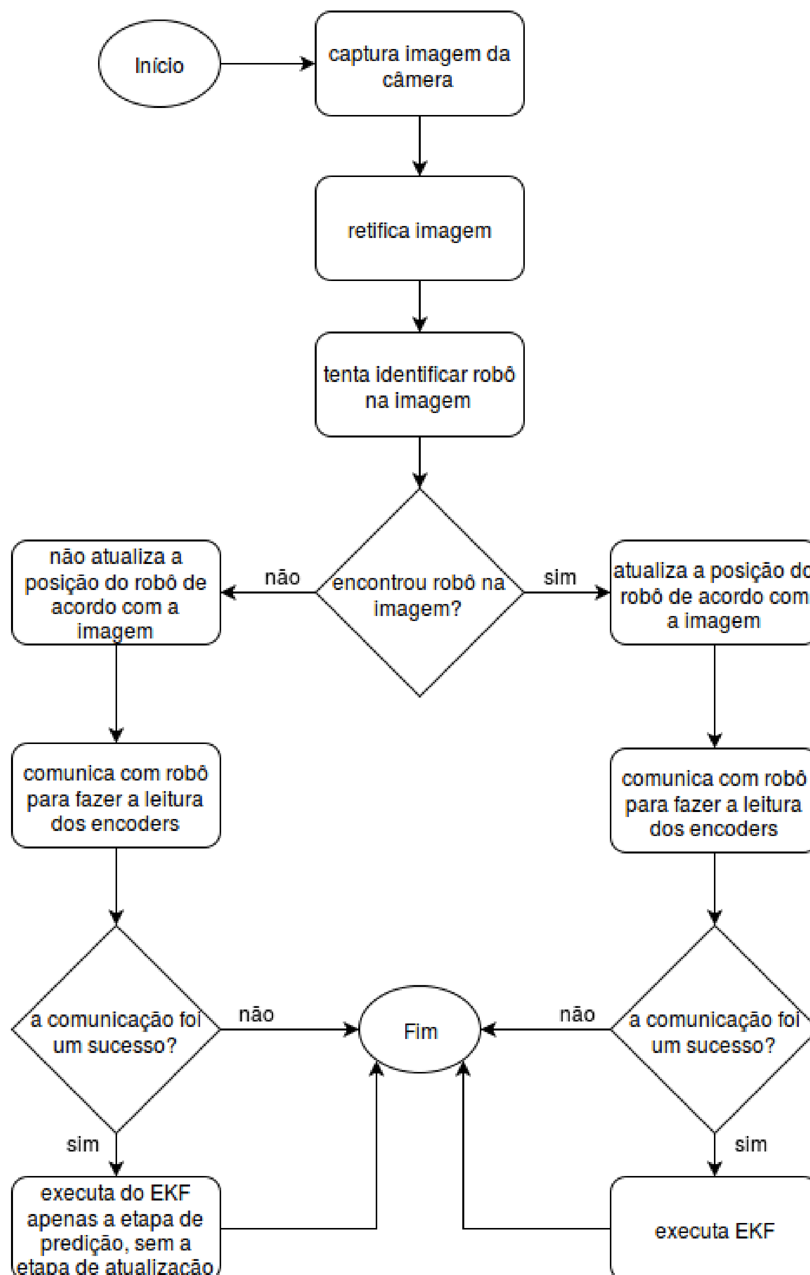


Figura 8 – Fluxograma do código principal.

O código principal pode ser dividido em visão computacional, comunicação com robô e EKF.

## 4.1 Visão computacional

Para utilizar a câmera como sensor, foi utilizado uma biblioteca chamada OpenCV, que é uma biblioteca *opensource* de processamento de imagens e visão computacional [14].

Para identificar o robô, foi colocada uma *tag* em cima dele com duas cores.

Figura 9 – Robô com tag.

Uma cor sinaliza a frente do robô (escolhido roxo para testes), e outra cor sinaliza a parte de trás do robô (escolhido amarelo para teste).

O código com OpenCV deve realizar 3 tarefas a cada iteração: capturar a imagem da câmera, cortar e retificar a imagem para desprezar os pixels que não serão utilizados, analisando apenas a área de interesse [3] e processar a imagem para encontrar a posição do robô.

Para capturar a imagem da câmera, basta uma linha de código.

```
_, frame = cap.read()
```

Onde *cap* é um objeto gerado pela biblioteca OpenCV, $frame$ é a matriz imagem e _ faz parte da sintaxe do Python para descartar informações desnecessárias.

Para retificar a imagem é necessário primeiro configurar as dimensões da imagem resultante. No caso deste trabalho a proporção entre altura/largura da imagem (em pixels) foi adaptada para ficar na mesma relação altura/largura do campo (em metros).

Como o campo é a região de interesse, foram utilizadas as coordenadas dos pixels onde se encontram as extremidades do campo para delimitar a área de interesse na imagem da câmera. Em seguida, é gerada a matriz de transformação com uma função pronta do OpenCV.

Com a imagem, a matriz de transformação e as dimensões, é então retificada a imagem com outra função pronta do OpenCV [8, 14]. A função que corta e retifica a imagem está no apêndice. Em que $cv2$ é a biblioteca OpenCV, a função $getPerspectiveTransform$ gera uma matriz de transformação de perspectiva e a função $warpPerspective$ aplica essa matriz na imagem.

Para encontrar quais são as coordenadas certas das extremidades do campo, foi necessário fazer outro programa. Este está no apêndice.

O processamento da imagem segue os passos:

1. Configura valores mínimos e máximos da cor desejada (valores no padrão HSV de cores). Necessário para encontrar a cor desejada. Por exemplo de um amarelo mais claro até um amarelo mais escuro. Para encontrar quais são os parâmetros certos da faixa de cor de cada cor, foi necessário fazer outro programa. Este também está no apêndice.

2. Muda o padrão de cores da imagem do RGB para o HSV para facilitar a busca das cores na imagem.

3. Gera uma mascara em que os pixels valem 1 se tiverem cor dentro da faixa estipulada e 0 caso contrário, criando uma imagem binária.

4. Filtra a mascara pelo método de erosão [3] [8] [14] para eliminar ruídos.

5. Encontra contorno do resultado da máscara. A coordenada das cores é igual a coordenada do centro da área do contorno encontrado.

6. Guarda a coordenada das cores em pixels.

7. Converte as coordenadas de pixels para metros.

8. Calcula a posição e orientação do robô de acordo com a posição das cores.

Para converter as coordenadas em pixels para coordenas em metros, basta multiplicar por uma constante.

$$constante_pixel_metro = \frac{valor_equivalente_em_metros}{valor_equivalente_em_pixels} \tag{4.1}$$

Como a base do campo tem 1,5m e a base da imagem retificada com a base do campo tem 552 pixels, então pode-se utilizar estes dados para calcular a constante descrita na equação 4.1.

As coordenadas $x$ e $y$ do robô são iguais à média das coordenadas das cores. A orientação do robô é encontrada com um simples cálculo trigonométrico, as coordenadas da cor da frente menos as coordenadas da cor de trás é igual a um vetor e a orientação desse vetor é igual ao $\theta$ do robô.

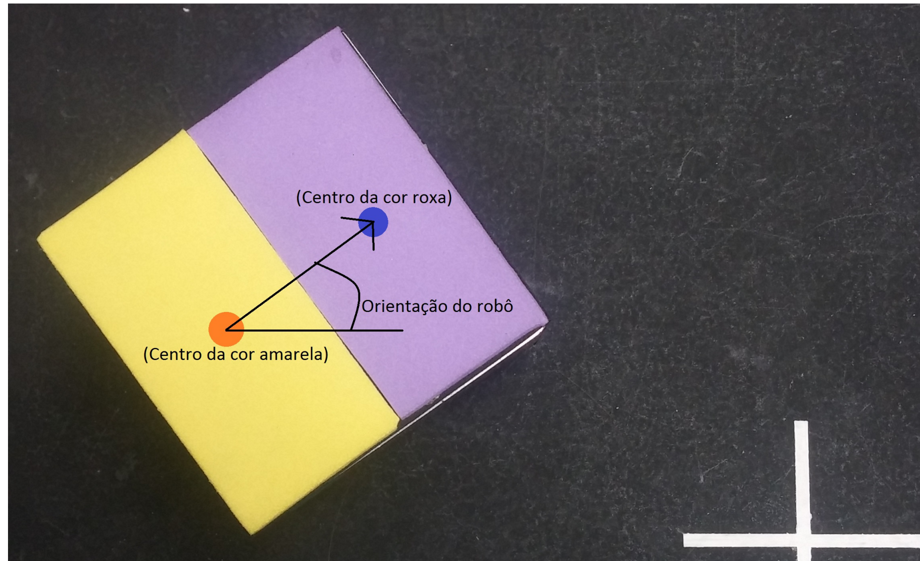


Figura 10 – Coordenadas do robô

Seguindo esses passos é possível criar uma função que detecta a posição do robô usando a câmera. Esta esta no apêndice.

## 4.2 Implementação Hardware/Software robô

O robô utilizado foi desenvolvido pela equipe Rinobot da UFJF. É um robô diferencial composto por um Arduino Nano, uma ponte H, uma bateria, um módulo Xbee e dois motores com encoders.

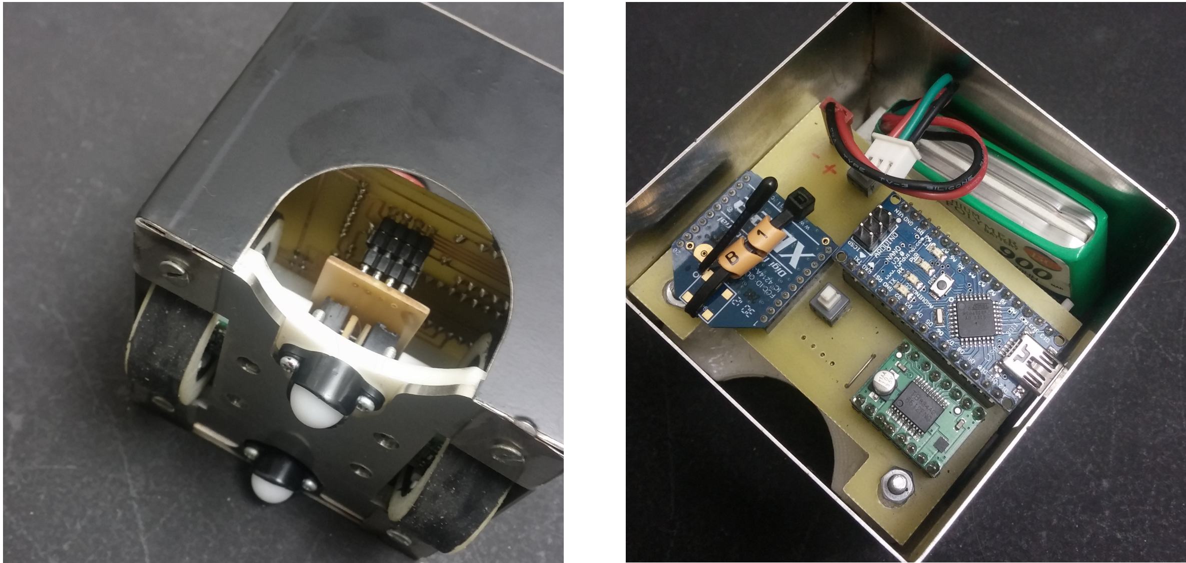

Figura 11 – Vista frontal e superior do robô.

O canal de comunicação entre o computador e o robô é sem fio. Através de um módulo Xbee [15], foi emulado um canal de comunicação serial entre o robô (controlado por um arduino nano [16]) e o computador, onde o algorítimo estava sendo executado em tempo real através do interpretador de Python.

O robô foi programado (pelo Arduino) para ler os comandos recebidos pelo Xbee e executar o controle das rodas. Foi programado também para ler as medidas dos encoders, calcular a velocidade linear e rotacional através das equações 2.2 e 2.3 e enviar essas informações pelo Xbee sempre que for requisitado.

No programa do computador foi feita uma função que envia o comando de requisição de dados pela porta usb do módulo Xbee e, depois disso, lê os dados de velocidade linear e rotacional enviados pelo robô.

O controle de baixo nível do robô não é o objetivo deste trabalho, mas o código do arduino está no apêndice.

O fluxograma da comunicação do computador com o robô pode ser observado na figura 12.

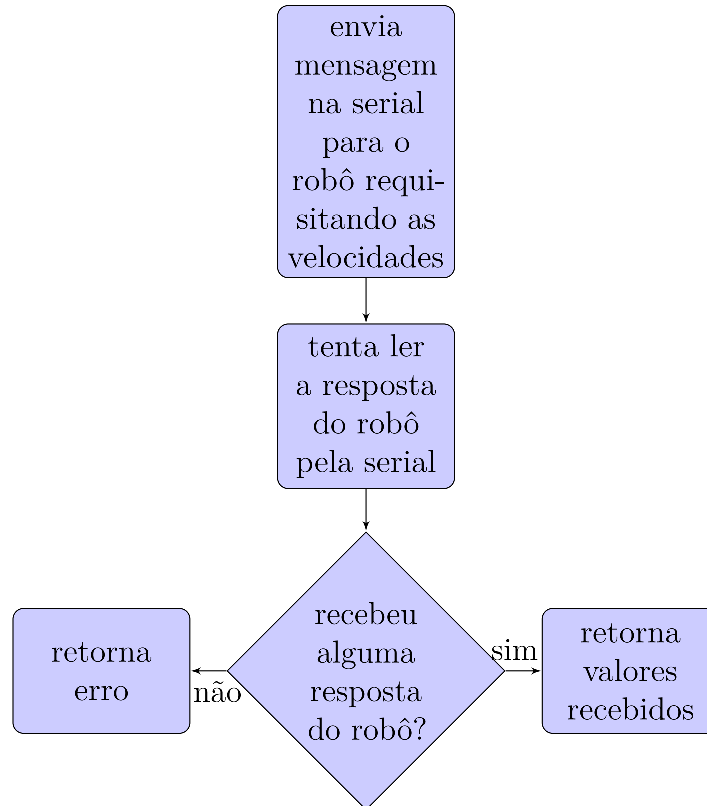


Figura 12 – Fluxograma da obtenção da leitura dos encoders do robô.

A velocidade linear e rotacional do robô é armazenada numa variável e será utilizada na função EKF. O código da função de comunicação pode ser encontrada no apêndice.

## 4.3 EKF

Para realizar todos os cálculos matriciais em Python, foi utilizada a biblioteca Numpy [17].

A função EKF tem como entrada a posição do robô medida pela câmera $z_k$, a velocidade linear $v_k$ e angular $\omega_k$ do robô medidas pelos encoders, o estado a *Posteriori* anterior $x^+_{k-1}$, o intervalo de tempo decorrido $\Delta t_k$, e a covariância de estado a *Posteriori* anterior $P^+_{k-1}$. A EKF retorna no final de todos os cálculos o estado a *Posteriori* $x^+_k$ e a covariância de estado a *Posteriori* $P^+_k$.

A primeira coisa feita na função é converter os vetores recebidos para a forma matricial padrão da biblioteca Numpy.

É configurado algumas constantes e matrizes necessárias, tais como distância entre as rodas $b$, as constantes de derrapamento $k_r$ e $k_l$, a matriz de mapeamento do sensor $H$ e a covariância do sensor $R$. Observe que $H$ é uma matrix identidade nesta situação, pois é extraída a informação da câmera já no mesmo formato de $x^+$ conforme já explicado. Os valores de $R$ foram estimados através de uma bateria de testes. Foi necessário criar outro

código para obter esses valores e este código está no apêndice.

Em seguida é calculado o valor de algumas variáveis necessárias.

$$\Delta s_k = v_k \Delta t_k \tag{4.2}$$

$$\Delta \theta_k = \omega_k \Delta t_k \tag{4.3}$$

$$\Delta s_{rk} = \Delta s_k + \frac{b}{2}\Delta \theta_k \tag{4.4}$$

$$\Delta s_{lk} = \Delta s_k - \frac{b}{2}\Delta \theta_k, \tag{4.5}$$

em que:

$\Delta s$ : Deslocamento linear

$v$ : Velocidade linear

$\Delta \theta$ : Deslocamento angular

$\omega$ : Velocidade angular

$\Delta t$ : Tempo decorrido com essas velocidades

$\Delta s_r$ : Deslocamento da roda direita

$\Delta s_l$ : Deslocamento da roda esquerda

$b$ : Distância entre as rodas

Com tudo em mãos, $x_k^+$ e $P_k^+$ são calculados seguindo os passos já demonstrados neste trabalho. O código está no apêndice.

É comum ocorrer falhas na identificação dos robôs, seja por problemas de iluminação ou obstrução de imagem como por exemplo por ter uma pessoa colocando a mão em cima do robô para reposicioná-lo. Por isso foi importante para a robustez do programa criar exceções que tratarão esses problemas. Caso não for possível encontrar a posição do robô com a câmera, o EKF vai ser executado mas epenas com a parte preditiva, sem a parte de atualização.

## 4.4 Ajuste dos parâmetros do EKF

Para que um bom desempenho do filtro de Kalman Extendido possa ser alcançado, é necessário saber qual é a covariância da câmera e o valor dos parâmetros $k_r$ e $k_l$, pertencentes a matriz $Q$.

A covariância da câmera foi medida da seguinte forma: o robô foi colocado numa determinada posição conhecida, foram realizadas 100 medições da posição do robô com a

câmera e, em seguida, foi calculada média e variância das medições. Esse procedimento foi repetido em 56 posições diferentes, totalizando 5600 medições.

Observou-se que a covariância da câmera varia com a posição do robô ao longo do campo, isso pode ter sido causado pela iluminação não uniforme do campo ou devido à indefinição das cores dos pixels em determinadas posições. Deste modo, neste trabalho a matriz $R$ foi calculada para o pior caso, ou seja, para a posição onde a variância das medidas era a maior. Foi necessário criar outro código para obter esses valores e este código está no apêndice.

Para calibrar $k_r$ e $k_l$, foi enviado um comando para o robô andar por um determinado trajeto e foi registrado todas as medidas da câmera e encoders durante este trajeto. A posição inicial foi medida como a média de 1000 medições da câmera. O mesmo foi feito para determinar a posição final. A Figura 13 ilustra a coleta dessas medidas.

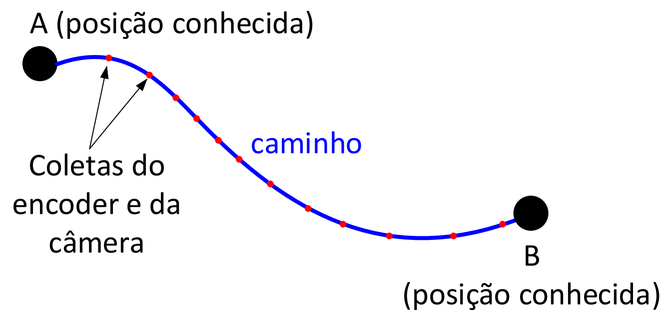


Figura 13 – Coordenadas do robô

Foi desenvolvido um programa em MATLAB para otimizar os valores de $k_r$ e $k_l$. Como o piso é o mesmo para as duas rodas, então a derrapagem é teoricamente a mesma também e, portanto, $k_r = k_l$.

A função de otimização pega o ponto inicial e todos os valores adquiridos durante o trajeto, esses valores são introduzidos na EKF (reescrita em MATLAB) e retorna o erro quadrático da diferença entre a posição final retornada pela EKF e a posição final adquirida. Os valores de $k_r$ e $k_l$ serão aqueles que farão com que a Função Objetivo (FOB) seja mínima.

A função foi otimizada com a função *fmincon()* do MATLAB [9].

## 4.5 Resultados

Para observar os resultados foram feitos testes com o robô parado, se movimentando em linha reta e se movimentando em círculos. Em todos os casos foram registrados os valores de posição e orientação de acordo com a câmera e também de acordo com a estimativa gerada pelo filtro de Kalman. Esses valores registrados foram plotados utilizando o MATLAB.

A Figura 14 mostra uma imagem obtida da câmera com o robô em repouso. O círculo vermelho mostra a estimativa do filtro de Kalman para a posição e orientação e o círculo verde mostra a medição destes parâmetros obtida através da câmera.

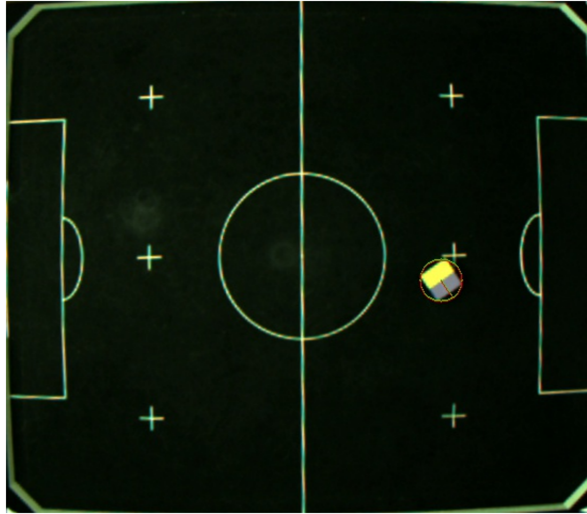

Figura 14 – Teste com o robô parado.

Com o robô parado foi possível uma excelente filtragem de ruídos, como observado no gráfico da Figura 15. Nas coordenadas $x$ e $y$ a posição que a câmera retornava tinha uma varição de cerca de 1mm, demonstrando ser um sensor muito bom para esta finalidade, entretanto mesmo essas pequenas variações são filtradas pelo filtro de Kalman. Por outro lado a orientação do robô fica variando de forma brusca, oscilando entre dois valores com 4° de diferença entre eles. Neste caso o filtro de Kalman apresentou um resultado muito mais condizente com a realidade, mantendo um valor mais constante.

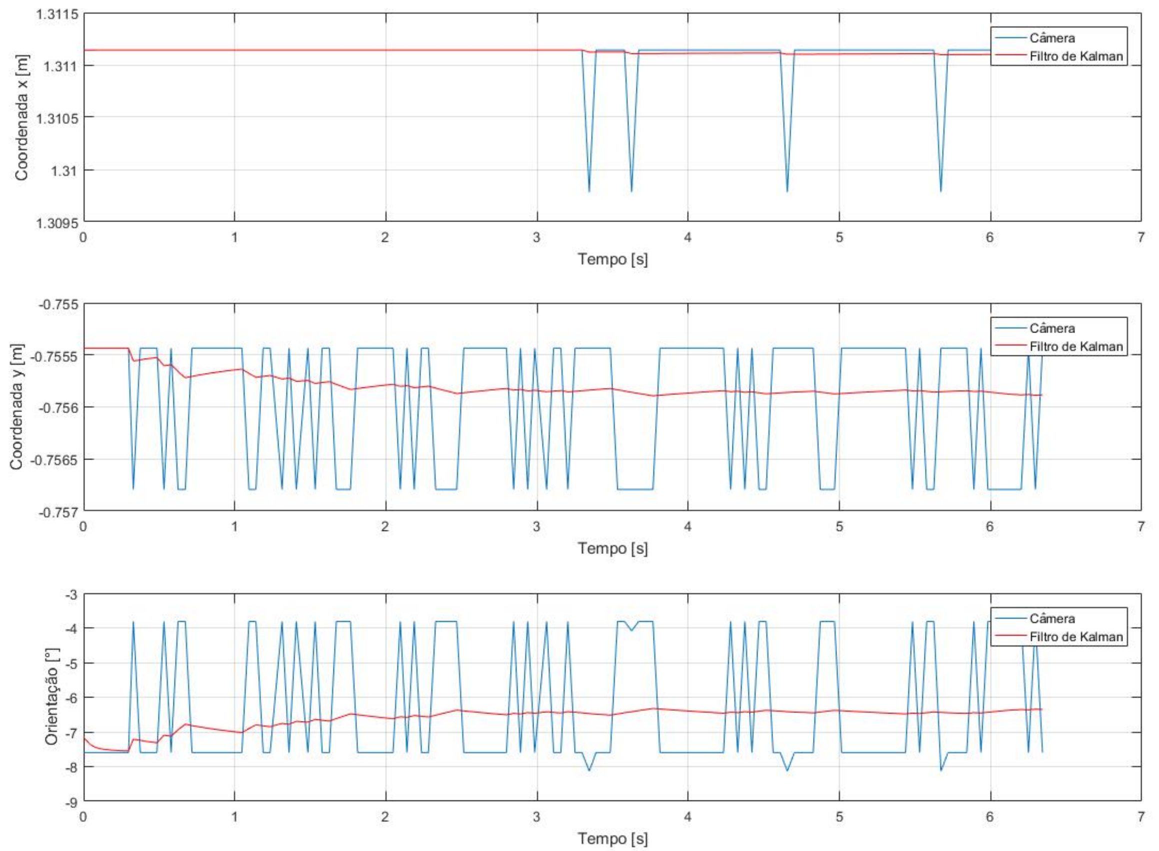


Figura 15 – Gráficos das variáveis de estado do teste do robô parado.

No teste com o robô andando em linha reta, foi observado que o robô não executou um movimento perfeitamente reto devido às suas características construtivas e controle de baixo nível. A Figura 16 mostra alguns frames da câmera coletados durante o movimento em linha reta do robô. Através da Figura 17 é possível observar que a câmera manteve uma boa precisão das coordenadas $x$ e $y$, mas mediu a orientação de forma bem ruidosa, já o filtro de Kalman acompanhou a precisão da câmera com as coordenadas $x$ e $y$ e filtrou parte do ruído da leitura de orientação da câmera.

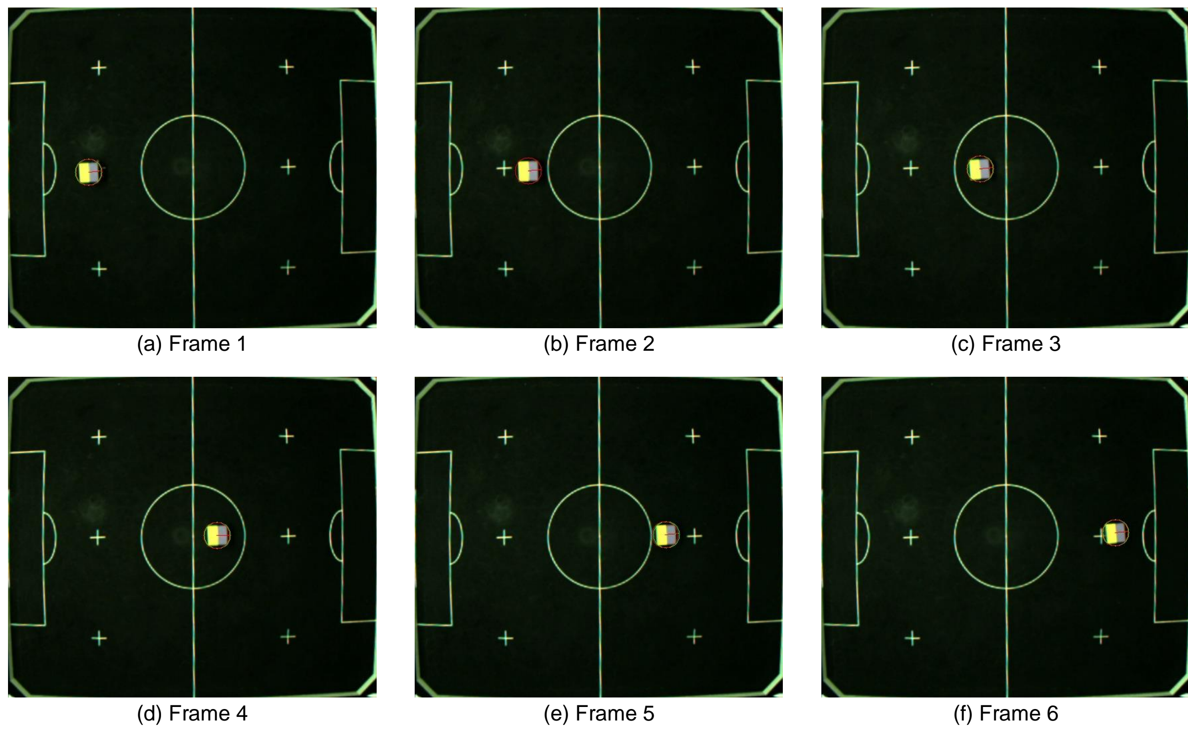


Figura 16 – Frames do teste com o robô andando em linha reta.

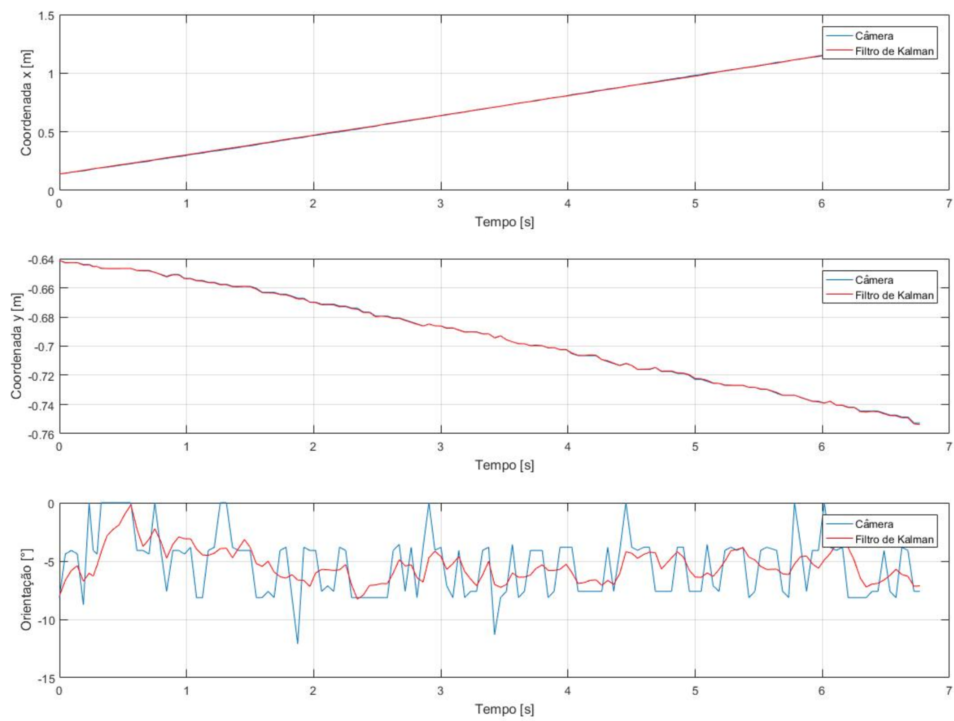


Figura 17 – Gráficos das variáveis de estado do teste do robô andando em linha reta.

A Figura 18 mostra alguns frames coletados durante o teste com o robô percorrendo uma trajetória circular. Neste caso, o gráfico das coordenas $x$ e $y$ do filtro de Kalman fica sobreposto ao mesmo gráfico da câmera, pois a câmera realmente faz medições bem precisas e com pouco ruído para esses parâmetros. Já a orientação, novamente, a câmera mede de forma ruidosa e o filtro de Kalman retorna um valor menos ruidoso. Estes resultados podem ser verificados através da Figura 19.

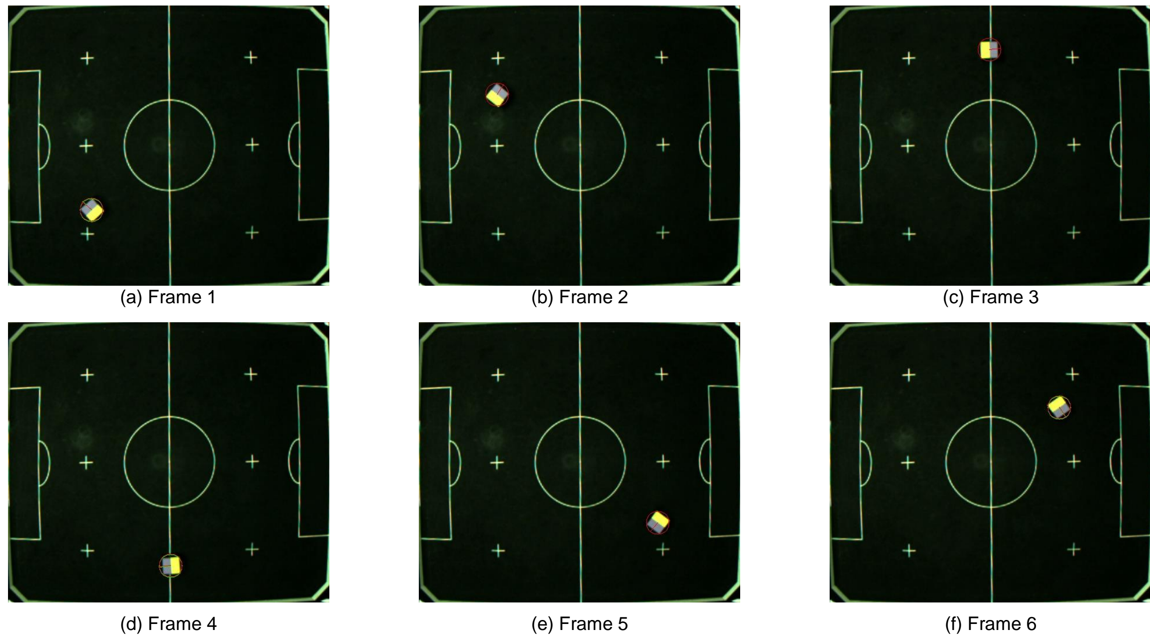


Figura 18 – Frames do teste com o robô andando em círculos.

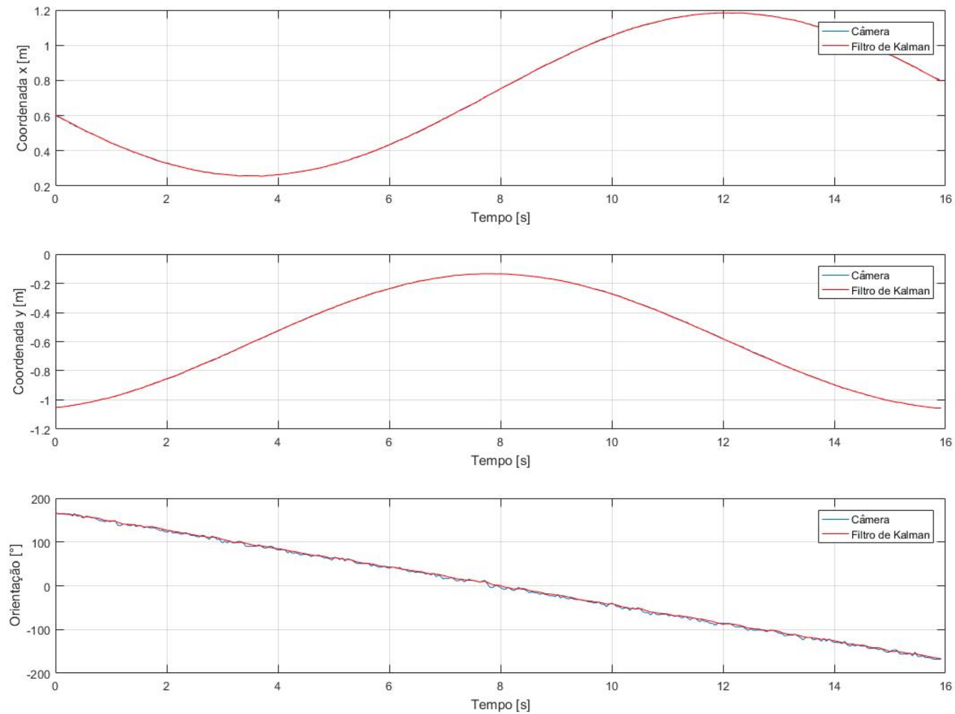


Figura 19 – Gráficos das variáveis de estado do teste do robô andando em círculos.

Esses testes demonstram que o filtro de Kalman consegue filtrar ruídos da câmera na leitura da orientação do robô.

A fim de testar a robustez do sistema de localização proporcionada pelo filtro de Kalman, a visão da câmera foi obstruída por um curto período de tempo durante o percurso do robô. A Figura 20 mostra alguns frames coletados durante o referido teste.

Como observado nos gráficos da Figura 21, mesmo que a câmera perca a posição real do robô, o filtro de Kalman (trabalhando apenas com sua parte preditiva) consegue retornar uma posição mais fidedigna com a realidade.

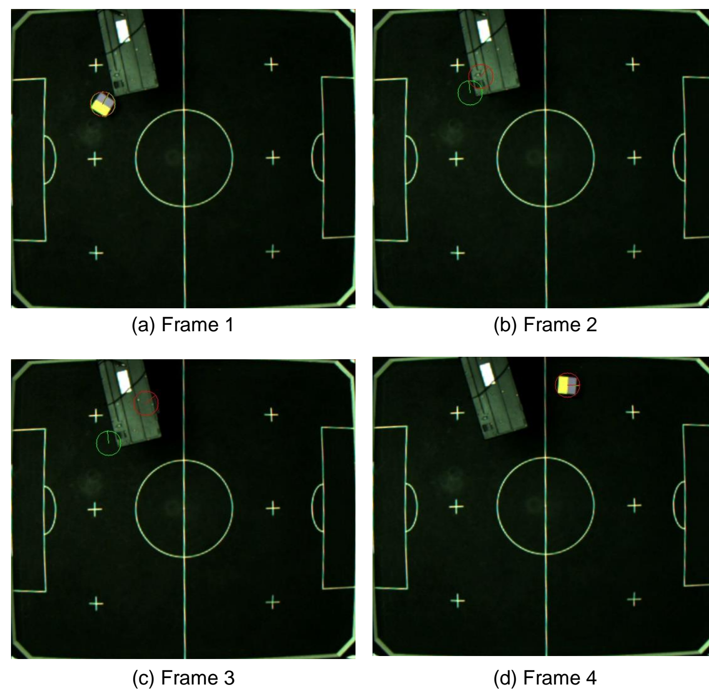


Figura 20 – Frames obtidos durante o teste de robustez.

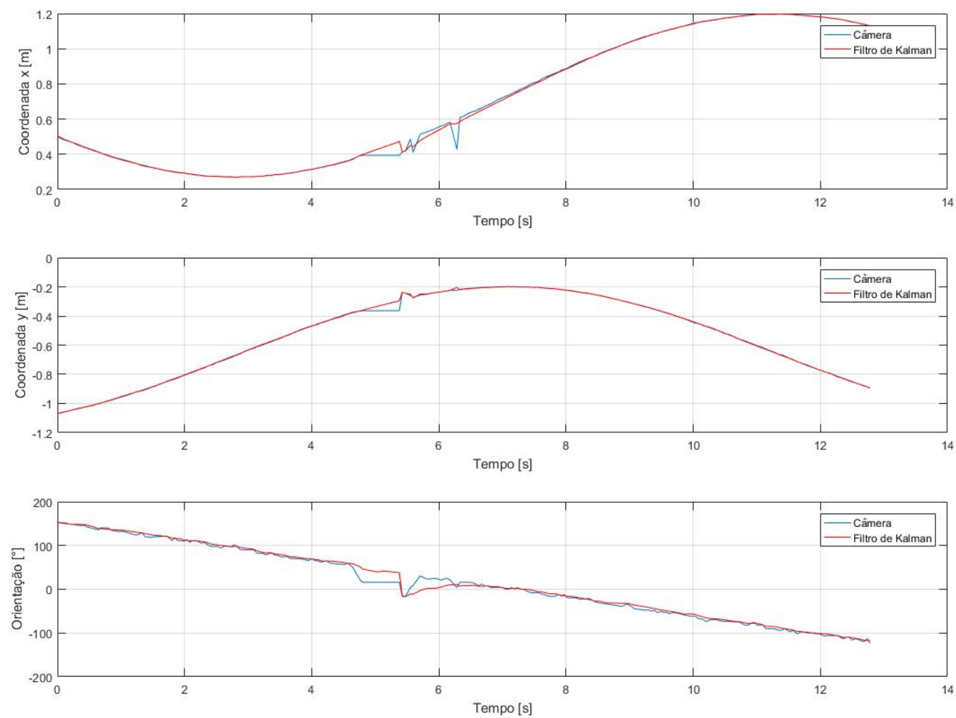


Figura 21 – Resultados com o teste de robustez

## 4.6 Problemas de implementação

Alguns problemas foram identificados durante a implementação. O maior problema de todos foi que o filtro de Kalman não acompanhava a posição do robô de forma satisfatória.

De forma empírica foi corrigido esse problema multiplicando (dentro da função EKF) a velocidade recebida por uma constante para compensar a diferença entre as medidas da câmera e as medidas do encoder.

Esse ajuste era esperado, uma vez que erros de medição da distância entre as rodas e tamanho das mesmas ou ainda erros de medição do encoder poderiam levar a estimativas incorretas das velocidades angular e linear do robô.

Apesar dos resultados terem sido bons, foram realizados testes de normalidade das medidas da câmera com as funções hist() e kstest() do MATLAB [11] [12], que apontaram que as medições do dispositivo não resultavam em distribuições normais. Todavia, conforme já mencionado, as medições provenientes da câmera foram aproximadas por variáveis gaussianas para que os testes pudessem ser realizados.

# 5 Conclusão

Este trabalho foi feito visando implementar a localização usando filtro de Kalman para uso futuro pela equipe Rinobot da UFJF. Para tal foi feito um estudo sobre o filtro de Kalman e o EKF e foi programado robô e computador para possibilitar a execução deste objetivo.

Aproximando os erros de leitura da câmera e dos encoders como gaussianas e linearizando o sistema (robô diferencial), foi possível implementar o Filtro de Kalman Estendido (EKF).

Foram feitos testes para encontrar a covariância da câmera e do modelo, testes para calibrar o EKF e testes para adquirir os resultados finais deste trabalho.

O resultado final da implementação do filtro de Kalman foi satisfatório, obtendo resultados mais precisos e acurados que usando apenas a câmera para localizar o robô. O filtro de Kalman também trouxe uma robustez muito maior, já que acompanha o movimento do robô mesmo se a imagem da câmera for obstruída por um determinado intervalo de tempo.

## 5.1 Comentários adicionais

O trabalho foi muito bom para integrar conhecimentos adquiridos durante o curso. Tais como: microcontroladores, robótica móvel, informática industrial e métodos estocásticos.

E foi interessante ter a oportunidade de poder trazer/implementar algo que é historicamente usado na engenharia aeroespacial para a equipe de futebol de robôs da UFJF - Rinobot.

## 5.2 Trabalhos futuros

Um problema previsível é que, durante o futebol de robôs, é comum haver contato físico entre os robôs de forma que as rodas girem em falso por muito tempo, atrapalhando bastante o EKF. É possível melhorar este método implementado estudando técnicas que superem esse problema apresentado, aumentando ainda mais a robustez do EKF.

Como trabalho futuro nessa área, é possível, também, incrementar o filtro de Kalman utilizando outro sensor para obter a posição e orientação do robô, tal como um acelerômetro e giroscópio, que não teria problemas com o deslizamento das rodas. Seria necessário fazer estudos sobre um novo modelo cinemático que se adeque ao novo sensor.

Por fim, uma outra sugestão seria o estudo de outros métodos de localização que possam lidar com variáveis não gaussianas, como o Método de Localização Monte-Carlo.

## REFERÊNCIAS


[1] Thrun, S. *Probabilistics Robotics* Communications of the ACM.

[2] SIEGWART, R. and NOURBAKHSH, I. R. *Introduction to Autonomous Mobile Robots.* The MIT Press, Massachusetts, 2004.

[3] CORK, P. *Robotics, Vision and Control: Fundamental Algorithms in Matlab.* Springer, 2011.

[4] J. P. A. Albuquerque, J. M. P. Fortes, W. A. Finamore. *Probabilidade, variáveis aleatórias e processos estocásticos* Editora Interciência, 2008.

[5] P. Z. Peebles, Jr. *Random variables and random signal principles* 3a Edição McGraw-Hill, 1993.

[6] J. A. Gubner. *Probability and Random Processes for Electrical and Computer Engineers* Cambridge University Press, 1a Edição, 2006.

[7] $http://www.cbpf.br/cat/pdsi/gauss.html$

[8] $http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html$

[9] $https://www.mathworks.com/help/optim/ug/fmincon.html?requestedDomain=www.mathworks.com$

[10] $https://www.python.org/$

[11] $https://www.mathworks.com/help/matlab/ref/hist.html$

[12] $https://www.mathworks.com/help/stats/kstest.html$

[13] Gaylor, David E.; Lightsey, E. Glenn. *GPS/INS Kalman filter design for spacecraft operating in the proximity of the international space station.* AIAA Guidance, Navigation, and Control Conference and Exhibit. 2003.

[14] BRADSKI, Gary; KAEHLER, Adrian. *Learning OpenCV: Computer vision with the OpenCV library.* "O'Reilly Media, Inc.", 2008.

[15] $https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf$

[16] $https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf$

[17] $http://www.numpy.org/$

[18] $https://www.nasa.gov/centers/ames/news/releases/2004/moon/apollo_ames_atmos.html$

# APÊNDICE A – Código para encontrar a faixa de valores de cada cor

Este código foi feito para encontrar a faixa de valores das cores que englobam as cores da tag do robô.

```
import cv2
import numpy as np
import pickle

def nothing(x):
    pass

cap = cv2.VideoCapture(0)

# GET BACK THE OBJECTS
with open('save.txt') as fileObject:
    H,S,V,H2,S2,V2 = pickle.load(fileObject)

# CREATE TRACK BARS FOR COLOR CHANGE
cv2.namedWindow('main', cv2.WINDOW_AUTOSIZE)
cv2.createTrackbar('Lower\nH','main',H,179, nothing)
cv2.createTrackbar('S','main',S,255, nothing)
cv2.createTrackbar('V','main',V,255, nothing)
cv2.createTrackbar('Upper\nH2','main',H2,179, nothing)
cv2.createTrackbar('S2','main',S2,255, nothing)
cv2.createTrackbar('V2','main',V2,255, nothing)

#while(cap.isOpened()):         ta sem camera
while(1):

    # CAPTURE FRAME-BY-FRAME
    grabbed, frame = cap.read(0)
    #frame=cv2.imread('messigray4.png')
    cv2.imshow('frame',frame)

    # IF WE ARE VIEWING A VIDEO AND WE DID NOT GRAB A FRAME
    #if grabbed is True:

    # CONVERT BGR TO HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

```
# GET CURRENT POSITIONS OF THE TRACKBARS
H, S, V = cv2.getTrackbarPos('Lower\nH','main'),
          cv2.getTrackbarPos('S','main'),
          cv2.getTrackbarPos('V','main')
H2, S2, V2 = cv2.getTrackbarPos('Upper\nH2','main'),
             cv2.getTrackbarPos('S2','main'),
             cv2.getTrackbarPos('V2','main')

# DEFINE RANGE OF COLOR IN HSV
lower_green = np.array([29, 86, 6], dtype=np.uint8)
upper_green = np.array([80, 255, 255], dtype=np.uint8)
lower = np.array([H, S, V], np.uint8)
upper = np.array([H2, S2, V2], np.uint8)

# THRESHOLD THE HSV IMAGE TO GET ONLY CHOOSEN COLORS
mask = cv2.inRange(hsv, lower, upper)
kernel = np.ones((5,5),np.uint8)
mask = cv2.erode(mask,kernel,iterations = 1)

# MORPHOLOGICAL TRANSFORMATIONS
kernel = np.ones((10,10), dtype=np.uint8)
erosion = cv2.erode(mask,kernel,iterations = 1)
dilation = cv2.dilate(erosion,kernel,iterations = 2)
thresh = cv2.dilate(erosion,kernel,iterations = 2)

#cimg = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
#opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

# CONTOUR FEATURES
#ret,thresh = cv2.threshold(dilation,127,255,0)
_,contours, hierarchy = cv2.findContours(thresh,
                        cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

#area = moments['m00']
if len(contours) > 0: #colored object detected

    max_area = 0
    best_cnt = 0
```

```
        for cnt in contours:
            area = cv2.contourArea(cnt)
            if area > max_area:
                max_area = area
                best_cnt = cnt

        if max_area > 1000:
            moments = cv2.moments(best_cnt)
            cx = int(moments['m10']/moments['m00'])
            cy = int(moments['m01']/moments['m00'])
            cv2.drawContours(frame, contours[0], -1, (80, 255, 255), 2)

            font = cv2.FONT_HERSHEY_SIMPLEX
            text = [str(cx), str(cy)]
            text = ','.join(text)
            cv2.putText(frame, text, (cx,cy), font, 0.5,
                                            (80,255,255), 2, 8)
            cv2.putText(frame, str(area), (cx+150,cy-150), font,
                                        0.5, (80,255,255), 2, 8)


    # BITWISE-AND MASK AND ORIGINAL IMAGE
    res = cv2.bitwise_and(frame,frame, mask= mask)

    # DISPLAY THE RESULTING FRAME
    #cv2.imshow('frame',frame)
    #cv2.imshow('mask',mask)
    cv2.imshow('res',res)
    #cv2.imshow('dilation', dilation)
    #cv2.imshow('erosion', erosion)

    k = cv2.waitKey(25) & 0xFF
    if k == 27:
        break

# SAVE THE OBJECS
with open('save.txt', 'w') as fileObject:
    pickle.dump([H,S,V,H2,S2,V2], fileObject, protocol=0)
```

```
#cap.release()
cv2.destroyAllWindows()
```

Para utilizar este código, basta alterar os valores mostrados até aparecer na imagem da câmera apenas a cor desejada.

# APÊNDICE B – Código para encontrar extremidades do campo na imagem

Este código foi feito para encontrar os pixels das extremidades do campo na imagem. A coordenada desses pixels são importantes para poder cortar e retificar a imagem da manira correta.

```
import cv2
import numpy as np

vetor=[]
cap = cv2.VideoCapture(0)

# mouse callback function
def armazena(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDBLCLK:
        print x,y
        vetor.append([x,y])

# Create a black image, a window and bind the function to window
#img = np.zeros((512,512,3), np.uint8)
_, frame = cap.read()
cv2.namedWindow('image')
cv2.setMouseCallback('image',armazena)

while(len(vetor)<4):
    _, frame = cap.read()
    cv2.imshow('image',frame)
    if cv2.waitKey(20) & 0xFF == 27:
        break
cv2.destroyAllWindows()

rows,cols,ch = frame.shape

cols=552

pts1 = np.float32([vetor[0],vetor[1],vetor[2],vetor[3]])
pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])
M = cv2.getPerspectiveTransform(pts1,pts2)
```

```
dst = cv2.warpPerspective(frame,M,(cols,rows))

print 'vetor',vetor,'rows',rows,'cols',cols

while(1):
    cv2.imshow('resultado',dst)
    if cv2.waitKey(20) & 0xFF == 27:
        break
cv2.destroyAllWindows()
```

Para utilizar este código basta clicar duas vezes em cada canto do campo na ordem: canto superior esquerdo, canto superior direito, canto inferior esquerdo, canto inferior direito.

# APÊNDICE C – Código completo

Todos as partes da implementação estão neste código.

```
'''Codigo que detecta a posicao de um robo usando sistema de cor e
camera, captura a leitura dos encoders do robo e junta as
informacoes com um Filtro de Kalman.

2016. UFJF. Gustavo Hofstatter. hofstatter.gustavo@engenharia.ufjf.br'''

import cv2
import numpy as np
from math import cos,sin,pi,atan2
import serial   # importando a biblioteca
import time

def retifica_imagem(imagem):
    rows,cols= 480,552
    vetor= [[123, 3], [603, 31], [101, 415], [574, 447]]
    pts1 = np.float32([vetor[0],vetor[1],vetor[2],vetor[3]])
    pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])
    M = cv2.getPerspectiveTransform(pts1,pts2)
    dst = cv2.warpPerspective(imagem,M,(cols,rows))
    return dst

def captura_posicao_do_robo_com_camera(frame):
    def Encontra_cor(imagem,corMinima,corMaxima,corReconhecida):
        # Convert BGR to HSV
        hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)
        # Threshold the HSV image to get only blue colors
        mask = cv2.inRange(hsv, corMinima, corMaxima)
        #Filtra a mascara
        kernel = np.ones((5,5),np.uint8)
        mask = cv2.erode(mask,kernel,iterations = 1)
        #Encontra e desenha posicao da cor
        _,contours,hierarchy = cv2.findContours(mask, 1, 2)
        if contours!=[]:
            cnt = contours[0]
            x,y,w,h = cv2.boundingRect(cnt)
            return [x+w/2,-1*(y+h/2)]
```

```
    lower_amarelo = np.array([24,52,255])
    upper_amarelo = np.array([50,248,255])
    lower_roxo = np.array([71,0,0])
    upper_roxo = np.array([179,110,227])
    amarelo=Encontra_cor(frame,lower_amarelo,upper_amarelo,(0,255,255))
    roxo=Encontra_cor(frame,lower_roxo,upper_roxo,(255,0,255))
    constante_pixel_centimetro=1.5/552
    #Converte coordenadas em cm
    roxo=[roxo[0]*constante_pixel_centimetro,
                roxo[1]*constante_pixel_centimetro]
    amarelo=[amarelo[0]*constante_pixel_centimetro,
                amarelo[1]*constante_pixel_centimetro]
    posicao_do_robo=[(roxo[0]+amarelo[0])/2,(roxo[1]+amarelo[1])/2]
    teta=atan2(roxo[1]-amarelo[1],roxo[0]-amarelo[0])*180/3.1415
    #return [posicao_do_robo[0],posicao_do_robo[1],teta]
    return posicao_do_robo+[teta]

def captura_leitura_dos_encoders(nome_do_robo):
    def requisita_dados_do_robo(robo_selecionado):
        #OBS: essa funcao muda muito de acordo com o protocolo usado.
        if robo_selecionado==1:
            ser.write('v000w000')
    def le_e_interpreta_dados_do_robo():
        #OBS: essa funcao muda muito de acordo com o protocolo usado.
        if ser.inWaiting()>0:
            dados_lidos_do_robo= ser.read()
            while dados_lidos_do_robo[-1]!='\n':
                dados_lidos_do_robo+= ser.read()
            v=float(dados_lidos_do_robo)
            dados_lidos_do_robo= ser.read()
            while dados_lidos_do_robo[-1]!='\n':
                dados_lidos_do_robo+= ser.read()
            w=float(dados_lidos_do_robo)
            return [v/1000,w]
        else:
            return 'erro'
    if nome_do_robo=='robo666':
        requisita_dados_do_robo(1)
        leitura= le_e_interpreta_dados_do_robo()
```

```
        return leitura

def leitura_ta_correta(leitura_a_ser_avaliada):
    if leitura_a_ser_avaliada=='erro': return False
    else: return True

def filtro_de_kalman(pose_pela_camera,v_w,ultima_posicao,dt,
                          matriz_P_anterior,deu_ruim_na_camera=0):
    pose_pela_camera= np.matrix(pose_pela_camera).T
    v,w= v_w[0],v_w[1]
    v,w= v*1.14,w*1.14
    ds,dw= v*dt,w*dt
    ultima_posicao= np.matrix(ultima_posicao).T
    P= matriz_P_anterior
    b=.055
    dsr=ds+b/2*dw
    dsl=ds-b/2*dw
    kr=9
    kl=9
    C= np.matrix([[1,0,0],[0,1,0],[0,0,1]])
    R= np.matrix([[9.23765359168e-07,0,0],
                  [0,7.93803166352e-07,0],
                  [0,0,6]])
    I= np.matrix([[1,0,0],[0,1,0],[0,0,1]])
    # Kalman Extendido
    x_predito= ultima_posicao+
               np.matrix([[ds*cos((ultima_posicao[2,0]+dw/2)*pi/180)],
                          [ds*sin((ultima_posicao[2,0]+dw/2)*pi/180)],
                          [dw]])
    covar= np.matrix([[kr*abs(dsr),0],
                      [0,kl*abs(dsl)]])
    Fp= np.matrix([[1,0,-ds*sin((ultima_posicao[2,0]+dw/2)*pi/180)],
                   [0,1,ds*cos((ultima_posicao[2,0]+dw/2)*pi/180)],
                   [0,0,1]]);
    Fdrl= np.matrix([[1/2*cos((ultima_posicao[2,0]+dw/2)*pi/180)
                     -ds/(2*b)*sin((ultima_posicao[2,0]+dw/2)*pi/180),
                     1/2*cos((ultima_posicao[2,0]+dw/2)*pi/180)
                     +ds/(2*b)*sin((ultima_posicao[2,0]+dw/2)*pi/180)],
                     [1/2*sin((ultima_posicao[2,0]+dw/2)*pi/180)
```

```
                              +ds/(2*b)*cos((ultima_posicao[2,0]+dw/2)*pi/180),
                               1/2*sin((ultima_posicao[2,0]+dw/2)*pi/180)
                              -ds/(2*b)*cos((ultima_posicao[2,0]+dw/2)*pi/180)],
                              [1/b,-1/b]])
    P_predito= Fp*P*Fp.T+Fdrl*covar*Fdrl.T
    if deu_ruim_na_camera==0:
        K= P_predito*C.T*(C*P_predito*C.T+R).I
        posicao_atual= x_predito+K*(pose_pela_camera-C*x_predito)
        P= (I-K*C)*P_predito;
        return [posicao_atual[0,0],posicao_atual[1,0],
                                            posicao_atual[2,0]],P
    else:
        return [x_predito[0,0],x_predito[1,0],x_predito[2,0]],P_predito

posicao_do_robo= [0,0,0]
ser = serial.Serial('COM5', 57600)
time.sleep(2)#Tempo necessario pro arduino estabilizar
ser.write('a')#OBS: linha necessaria de acordo com o protocolo usado.
ser.write('f150r010')#Pra testar mando andar em circulo
matriz_P= np.matrix([[1,0,0],[0,1,0],[0,0,100]])
cap = cv2.VideoCapture(0)
t= time.time()
while(1):
    _, frame = cap.read()
    frame= retifica_imagem(frame)
    try:
        posicao_do_robo_de_acordo_com_a_camera=
                        captura_posicao_do_robo_com_camera(frame)
        flag_deu_ruim_na_camera=0
    except:
        flag_deu_ruim_na_camera=1
        print 'Problema na leitura da camera'
    leitura_dos_encoders= captura_leitura_dos_encoders('robo666')
    if leitura_ta_correta(leitura_dos_encoders):
        tempo_decorrido_desde_a_ultima_leitura= time.time()-t
        t= time.time()
        posicao_do_robo,matriz_P=
              filtro_de_kalman(posicao_do_robo_de_acordo_com_a_camera,
                          leitura_dos_encoders,posicao_do_robo,
```

```
                                        tempo_decorrido_desde_a_ultima_leitura ,
                                        matriz_P , flag_deu_ruim_na_camera)
            print posicao_do_robo_de_acordo_com_a_camera,posicao_do_robo,[t]
        else:
            ser.flushInput()
            print 'Problema na comunicacao'
            pass
        #Plot
        pose_x_pixelado_camera=
                    int(posicao_do_robo_de_acordo_com_a_camera[0]*552/1.5)
        pose_y_pixelado_camera=
                int(-1*posicao_do_robo_de_acordo_com_a_camera[1]*552/1.5)
        pose_th_pixelado_camera=
                    posicao_do_robo_de_acordo_com_a_camera[2]*pi/180
        cv2.circle(frame,(pose_x_pixelado_camera,pose_y_pixelado_camera),
                                                            20,(0,250,0))
        cv2.line(frame,(pose_x_pixelado_camera,pose_y_pixelado_camera),
            (int(pose_x_pixelado_camera+20*cos(pose_th_pixelado_camera)),
            int(pose_y_pixelado_camera-20*sin(pose_th_pixelado_camera))),
                                                            (0,250,0))
        pose_x_pixelado_kalman= int(posicao_do_robo[0]*552/1.5)
        pose_y_pixelado_kalman= int(-1*posicao_do_robo[1]*552/1.5)
        pose_th_pixelado_kalman= posicao_do_robo[2]*pi/180
        cv2.circle(frame,(pose_x_pixelado_kalman,pose_y_pixelado_kalman),
                                                            20,(0,0,250))
        cv2.line(frame,(pose_x_pixelado_kalman,pose_y_pixelado_kalman),
            (int(pose_x_pixelado_kalman+20*cos(pose_th_pixelado_kalman)),
            int(pose_y_pixelado_kalman-20*sin(pose_th_pixelado_kalman))),
                                                            (0,0,250))

        cv2.imshow('resultado',frame)
        if cv2.waitKey(20) & 0xFF == 27:
            break
    ser.write('f000r000')
    time.sleep(.5)
    ser.write('f000r000')
```

# APÊNDICE D – Código do arduino do robô

Este é o código utilizado no arduino do robô.

```
//Elisa Rossi , Frederick Tavares, Gustavo Hofstatter, Pedro Alcantara

#include <PololuWheelEncoders.h> //biblioteca do encoder
PololuWheelEncoders encoders; //objeto da biblioteca encoder

long tempo = 0; //contador de tempo
float dT = 0.1; // tempo de leitura do encoder
float vel_ref_e = 0; // velocidade de referencia esquerdo
float vel_ref_d = 0; // velocidade de referencia direito
float linVel=0; //velocidade linear do robo
float rotVel=0; //velocidade angular do robo

//Variaveis de posicao
float x=0;
float y=0;
float th=0;

//Distancia entre rodas
float l=0.055; //5,5cm = 0,055m

//Variaveis pro controle
float erro;
float proporcional;
float integral_e=0;
float integral_d=0;
float kp_e=50;
float ki_e=1000;
float kp_d=50;
float ki_d=1000;
int u;//Acao de controle

//vaiaveis de armazenamento da leitura dos encoders
float esquerdo = 0;
float direito = 0;
float vel_d=0;
float vel_e=0;
```

```
float pV = 48; //Numero de pulsos por volta encoder
float D = 0.039; //diametro da roda (m)
//Distancia percorrida por leitura do pulso:
float dS = D*3.1416/pV;

//pinos ponte H
int PWMA = 5;
int AIN1 = 7;
int AIN2 = 6;
int STBY = 8;
int BIN2 = 9;
int BIN1 = 10;
int PWMB = 11;

//variaveis globais de armazenamento dos
//valores PWM de alimentacao do motor A e B
int MTA = 0;
int MTB = 0;

int dado='a';
float valor=0;
int aux=0;

void setup() {
  // put your setup code here, to run once:
  pinMode(PWMA,OUTPUT);
  pinMode(PWMB,OUTPUT);
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(STBY,OUTPUT);
  pinMode(BIN1,OUTPUT);
  pinMode(BIN2,OUTPUT);

  digitalWrite(STBY,HIGH);//permite o funcionamento da ponte h

  encoders.init(16,17,18,19); //configuracao dos pinos do encoder

  Serial.begin(57600); //taxa de comunicacao baud-rate
```

```
  delay(2000);//delay paa comunicacao serial, MATLAB-ARDUINO

  Serial.println('a');// caractere para liberacao da
                      //comunicacao serial ARDUINO - MATLAB
  char data = 'b';
  while (data != 'a')
  {
    data = Serial.read();
  }
}

void loop() {
  //Estrutura para aquisicao de dados do encoder
  if(millis()-tempo >= dT*1000)
  {
    tempo = millis();

    //Atualiza posicao
    x+=linVel*dT*cos(th+rotVel*dT);
    y+=linVel*dT*sin(th+rotVel*dT);
    th+=rotVel*dT;
    if(th>3.1416){th=-3.1416*2+th;}
    if(th<-3.1416){th=3.1416*2-th;}

    //leituras do encoder:
    esquerdo = encoders.getCountsAndResetM1();
    direito = encoders.getCountsAndResetM2();

    //calcula a velocidade real do robo
    vel_e=abs(esquerdo*dS/dT);
    vel_d=abs(direito*dS/dT);
    /*
    //printa velocidade de referencia
    Serial.print(vel_ref_e);//leitura do encoder esquerdo em pulsos
    Serial.print(" ");
    Serial.print(vel_ref_d);//leitura do encoder esquerdo em pulsos
    Serial.print(" ");
```

```
    //printa leituras realizadas
    Serial.print(vel_e);//leitura do encoder esquerdo em pulsos
    Serial.print(" ");
    Serial.println(vel_d);//leitura do encoder direito em pulsos
  */
    //Controlador esquerdo
    erro=vel_ref_e-vel_e;
    proporcional=erro*kp_e;
    integral_e+=erro*ki_e*dT;
    u=int(proporcional+integral_e);
    if(u>255){u=255;} //saturador
    if(u<0){u=0;}
    MTA = u;

    //Controlador direito
    erro=vel_ref_d-vel_d;
    proporcional=erro*kp_d;
    integral_d+=erro*ki_d*dT;
    u=int(proporcional+integral_d);
    if(u>255){u=255;} //saturador
    if(u<0){u=0;}
    MTB = u;

    //zera parametro integrativo quando robo esta parado
    if(aux==0)
    {
      integral_e=0;
      integral_d=0;
      u=0;
    }
  }

  analogWrite(PWMA,MTA);//passa para o motor1 sua respectiva taxa de pwm
  analogWrite(PWMB,MTB);//passa para o motor2 sua respectiva taxa de pwm

  if(Serial.available()>3)
  {
   dado = Serial.read(); // dado recebe o valor da serial, dado pelas
                                          //teclas de movimento do robo
```

```
    //f=pra frente, b=pra tras, l=esquerda, r=direita, s=para
    //v=retorna a velocidade linear, w=retorna a velocidade angular,
    //x=retorna posicao x, y=retorna posicao y, t=retorna posicao th
    //i=seta valor positivo de x, u=seta valor negativo de x,
    //j=seta valor positivo de y, p=seta valor negativo de y,
    //k=seta valor positivo de th, h=seta valor negativo de th

    //le digito, converte ASCII e ordena quanto a grandeza
    valor= (int(Serial.read())-48)*100;
    valor+= (int(Serial.read())-48)*10;
    valor+= (int(Serial.read())-48)*1;
/*
    //printa comando recebido pela serial
    Serial.print(char(dado));
    Serial.print("   ");
    Serial.println(int(valor));
*/
    //seta acoes para cada comando enviado
    if(dado == 'f'){linVel=valor/1000.0;}
    if(dado == 'b'){linVel=-valor/1000.0;}
    if(dado == 'r'){rotVel=-valor*3.1416/180;}
    if(dado == 'l'){rotVel=valor*3.1416/180;}
    if(dado == 's')//Stop 32
    {
     digitalWrite(AIN1,LOW);
     digitalWrite(AIN2,LOW);
     digitalWrite(BIN1,LOW);
     digitalWrite(BIN2,LOW);
     vel_ref_d=0;
     vel_ref_e=0;
     linVel=0;
     rotVel=0;
     aux=0;
    }
    //responde a requisicao de dados
    if(dado=='v'){
        if(aux==0){Serial.println((vel_d+vel_e)/2.*1000);}
        if(aux==1){Serial.println((vel_d+vel_e)/2.*1000);}
        if(aux==2){Serial.println((-vel_d-vel_e)/2.*1000);}
```

```
    if(aux==3){Serial.println((-vel_d+vel_e)/2.*1000);}
    if(aux==4){Serial.println((vel_d-vel_e)/2.*1000);}
  }
  if(dado=='w'){
    if(aux==0){Serial.println((vel_d-vel_e)/2./l*180/3.1416);}
    if(aux==1){Serial.println((vel_d-vel_e)/2./l*180/3.1416);}
    if(aux==2){Serial.println((-vel_d+vel_e)/2./l*180/3.1416);}
    if(aux==3){Serial.println((-vel_d-vel_e)/2./l*180/3.1416);}
    if(aux==4){Serial.println((vel_d+vel_e)/2./l*180/3.1416);}
  }
  if(dado=='x'){Serial.println(x);}
  if(dado=='y'){Serial.println(y);}
  if(dado=='t'){Serial.println(th);}
  //seta posicao
  if(dado=='i'){x=valor/1000;}
  if(dado=='u'){x=-valor/1000;}
  if(dado=='j'){y=valor/1000;}
  if(dado=='p'){y=-valor/1000;}
  if(dado=='k'){th=valor*3.1416/180;}
  if(dado=='h'){th=-valor*3.1416/180;}

//calcula velocidade referencia para cada roda
vel_ref_d=linVel+l*rotVel;
vel_ref_e=linVel-l*rotVel;

//ajusta direcao das rodas
if(vel_ref_d>=0&&vel_ref_e>=0)//Forward 30
{
  digitalWrite(AIN1,HIGH);
  digitalWrite(AIN2,LOW);
  digitalWrite(BIN1,LOW);
  digitalWrite(BIN2,HIGH);
  aux=1;
}
else if(vel_ref_d<0&&vel_ref_e<0)//Backwad 31
{
  digitalWrite(AIN1,LOW);
  digitalWrite(AIN2,HIGH);
  digitalWrite(BIN1,HIGH);
```

```
      digitalWrite(BIN2,LOW);
      vel_ref_d=-1*vel_ref_d;
      vel_ref_e=-1*vel_ref_e;
      aux=2;
    }
    else if(vel_ref_d<0&&vel_ref_e>0)//Right 29
    {
        digitalWrite(AIN1,HIGH);
        digitalWrite(AIN2,LOW);
        digitalWrite(BIN1,HIGH);
        digitalWrite(BIN2,LOW);
        vel_ref_d=-1*vel_ref_d;
        aux=3;
    }
      else if(vel_ref_d>0&&vel_ref_e<0)//Left 28
    {
        digitalWrite(AIN1,LOW);
        digitalWrite(AIN2,HIGH);
        digitalWrite(BIN1,LOW);
        digitalWrite(BIN2,HIGH);
        vel_ref_e=-1*vel_ref_e;
        aux=4;
    }
  }
}

//fim
```

# APÊNDICE E – Código para medir covariância da câmera

Este código foi usado para medir a covariância da câmera.

```
'''Codigo que detecta a posicao de um robo usando sistema de cor e
camera, captura a leitura dos encoders do robo e junta as informacoes
com um Filtro de Kalman.

2016. UFJF. Gustavo Hofstatter. hofstatter.gustavo@engenharia.ufjf.br'''

import cv2
import numpy as np
from math import cos,sin,pi,atan2
import serial   # importando a biblioteca
import time

def retifica_imagem(imagem):
    rows,cols= 480,552
    vetor= [[62, 35], [548, 35], [61, 441], [530, 464]]
    pts1 = np.float32([vetor[0],vetor[1],vetor[2],vetor[3]])
    pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])
    M = cv2.getPerspectiveTransform(pts1,pts2)
    dst = cv2.warpPerspective(imagem,M,(cols,rows))
    return dst

def captura_posicao_do_robo_com_camera(frame):
    def Encontra_cor(imagem,corMinima,corMaxima,corReconhecida):
        # Convert BGR to HSV
        hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)
        # Threshold the HSV image to get only blue colors
        mask = cv2.inRange(hsv, corMinima, corMaxima)
        #Filtra a mascara
        kernel = np.ones((5,5),np.uint8)
        mask = cv2.erode(mask,kernel,iterations = 1)
        #Encontra e desenha posicao da cor
        _,contours,hierarchy = cv2.findContours(mask, 1, 2)
        if contours!=[]:
            cnt = contours[0]
            x,y,w,h = cv2.boundingRect(cnt)
            return [x+w/2,-1*(y+h/2)]
```

```
    lower_amarelo = np.array([24,52,255])
    upper_amarelo = np.array([50,248,255])
    lower_roxo = np.array([71,0,0])
    upper_roxo = np.array([179,110,227])
    amarelo=Encontra_cor(frame,lower_amarelo,upper_amarelo,(0,255,255))
    roxo=Encontra_cor(frame,lower_roxo,upper_roxo,(255,0,255))
    constante_pixel_centimetro=1.5/552
    #Converte coordenadas em cm
    roxo=[roxo[0]*constante_pixel_centimetro,
                roxo[1]*constante_pixel_centimetro]
    amarelo=[amarelo[0]*constante_pixel_centimetro,
                amarelo[1]*constante_pixel_centimetro]
    posicao_do_robo=[(roxo[0]+amarelo[0])/2,(roxo[1]+amarelo[1])/2]
    teta=atan2(roxo[1]-amarelo[1],roxo[0]-amarelo[0])*180/3.1415
    return posicao_do_robo+[teta]

def captura_leitura_dos_encoders(nome_do_robo):
    def requisita_dados_do_robo(robo_selecionado):
        #OBS: essa funcao muda muito de acordo com o protocolo usado.
        if robo_selecionado==1:
            ser.write('v000w000')
    def le_e_interpreta_dados_do_robo():
        #OBS: essa funcao muda muito de acordo com o protocolo usado.
        if ser.inWaiting()>0:
            dados_lidos_do_robo= ser.read()
            while dados_lidos_do_robo[-1]!='\n':
                dados_lidos_do_robo+= ser.read()
            v=float(dados_lidos_do_robo)
            dados_lidos_do_robo= ser.read()
            while dados_lidos_do_robo[-1]!='\n':
                dados_lidos_do_robo+= ser.read()
            w=float(dados_lidos_do_robo)
            return [v/1000,w]
        else:
            return 'erro'
    if nome_do_robo=='robo666':
        requisita_dados_do_robo(1)
        leitura= le_e_interpreta_dados_do_robo()
        return leitura
```

```
def leitura_ta_correta(leitura_a_ser_avaliada):
    if leitura_a_ser_avaliada=='erro': return False
    else: return True

def filtro_de_kalman(pose_pela_camera,v_w,ultima_posicao,dt,
                                                matriz_P_anterior):
    pose_pela_camera= np.matrix(pose_pela_camera).T
    v,w= v_w[0],v_w[1]
    ds,dw= v*dt,w*dt
    ultima_posicao= np.matrix(ultima_posicao).T
    P= matriz_P_anterior
    b=.055
    dsr=ds+b/2*dw
    dsl=ds-b/2*dw
    kr=1e-1
    kl=1e-1
    C= np.matrix([[1,0,0],[0,1,0],[0,0,1]])
    R= np.matrix([[1^2,0,0],
                  [0,1^2,0],
                  [0,0,1^2]])
    I= np.matrix([[1,0,0],[0,1,0],[0,0,1]])
    # Kalman Extendido
    x_predito= ultima_posicao+
                np.matrix([[ds*cos((ultima_posicao[2,0]+dw/2)*pi/180)],
                           [ds*sin((ultima_posicao[2,0]+dw/2)*pi/180)],
                           [dw]])
    covar= np.matrix([[kr*abs(dsr),0],
                      [0,kl*abs(dsl)]])
    Fp= np.matrix([[1,0,-ds*sin((ultima_posicao[2,0]+dw/2)*pi/180)],
                   [0,1,ds*cos((ultima_posicao[2,0]+dw/2)*pi/180)],
                   [0,0,1]]);
    Fdrl= np.matrix([[1/2*cos((ultima_posicao[2,0]+dw/2)*pi/180)
                -ds/(2*b)*sin((ultima_posicao[2,0]+dw/2)*pi/180),
                1/2*cos((ultima_posicao[2,0]+dw/2)*pi/180)
                +ds/(2*b)*sin((ultima_posicao[2,0]+dw/2)*pi/180)],
                   [1/2*sin((ultima_posicao[2,0]+dw/2)*pi/180)
                +ds/(2*b)*cos((ultima_posicao[2,0]+dw/2)*pi/180),
                1/2*sin((ultima_posicao[2,0]+dw/2)*pi/180)
```

```
                    -ds/(2*b)*cos((ultima_posicao[2,0]+dw/2)*pi/180)],
                        [1/b,-1/b]])
    P_predito= Fp*P*Fp.T+Fdrl*covar*Fdrl.T
    K= P_predito*C.T*(C*P_predito*C.T+R).I
    posicao_atual= x_predito+K*(pose_pela_camera-C*x_predito)
    P= (I-K*C)*P_predito;
    return [posicao_atual[0,0],posicao_atual[1,0],posicao_atual[2,0]],P

posicao_do_robo= [0,0,0]
ser = serial.Serial('COM6', 57600)
time.sleep(2)#Tempo necessario pro arduino estabilizar
ser.write('a')#OBS: linha necessaria de acordo com o protocolo usado.
#ser.write('f050r010')#Pra testar mando andar em circulo
matriz_P= np.matrix([[1,0,0],[0,1,0],[0,0,1]])
cap = cv2.VideoCapture(0)
t= time.time()
cont=0
N=100
medidas=[0]*N
while(cont<N):
    _, frame = cap.read()
    frame= retifica_imagem(frame)
    try:
        posicao_do_robo_de_acordo_com_a_camera=
                        captura_posicao_do_robo_com_camera(frame)
        medidas[cont]= posicao_do_robo_de_acordo_com_a_camera
        cont += 1
    except:
        print 'Problema na leitura da camera'
    leitura_dos_encoders= captura_leitura_dos_encoders('robo666')
    if leitura_ta_correta(leitura_dos_encoders):
        tempo_decorrido_desde_a_ultima_leitura= time.time()-t
        t= time.time()
        posicao_do_robo,matriz_P=
                filtro_de_kalman(posicao_do_robo_de_acordo_com_a_camera,
                                leitura_dos_encoders,posicao_do_robo,
                        tempo_decorrido_desde_a_ultima_leitura,matriz_P)
    else:
        ser.flushInput()
```

```
            print 'Problema␣na␣comunicacao'
            pass

        cv2.imshow('resultado',frame)
        if cv2.waitKey(20) & 0xFF == 27:
            break

x,y,th= [],[],[]
for i in range(N):
    x.append(medidas[i][0])
    y.append(medidas[i][1])
    th.append(medidas[i][2])
media_x,media_y,media_th= sum(x)/N,sum(y)/N,sum(th)/N
print media_x,media_y,media_th
sigma_x,sigma_y,sigma_th= 0,0,0
for i in range(N):
    sigma_x += (x[i]-media_x)**2/N
    sigma_y += (y[i]-media_y)**2/N
    sigma_th += (th[i]-media_th)**2/N
print sigma_x,sigma_y,sigma_th
```

Para usar este código, basta posicionar o robô em algum lugar do campo, executar o código e aguardar as informações aparecerem na tela.