
UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

MARACATRONICS

# *Free Angle Implementation*

Relatório

Thayná Cavalcante
Victor Araujo

Treinee de Controle

Recife
2019

# 1 Introdução

O algoritmo Free Angles foi desenvolvido dentro do *Warthog Robotics* com o objetivo inicial específico de permitir o cálculo dos "ângulos livres" para chute ao gol. Ao longo do tempo, as aplicações do Free Angles foram expandidas e o algoritmo foi aperfeiçoado para permitir um cálculo genérico de ângulos livres. O Free Angles pode ser definido formalmente com um algoritmo que calcula os ângulos livres a partir de um ponto de observação e uma lista de obstáculos.

# 2 Desenvolvimento

- **Estruturas e Métodos**

  A implementação realizada pela equipe faz uso de um modelo iterativo que, com o ferramentas do *SFML* e técnicas de geometria analítica, faz aparecer numa janela, em tempo real, as regiões que não podem ser utilizadas pelo robô para a realização de alguma atividade.

  Para isso, fizemos um programa que, enquanto a janela estiver aberta, acompanha a movimentação tanto dos adversários quanto do robô em analise e guarda suas coordenadas x, y em uma matriz, fazendo os seguintes cálculos:

  1. Encontra o módulo do segmento (A) situado entre o centro do robô e o centro do adversário através da função *distância*:

```
float distancia(int rX, int rY, int advX, int advY){

    float p;
    p = sqrt(pow(float(rX-advX),2) + pow(float(rY-advY),2);

    return p;
}
```

  2. Através do Teorema de Pitágoras, calcula o módulo do segmento (B) situado entre o centro do robô e um dos pontos de tangência do segmento em relação à circunferência formada em torno do robô adversário
  3. Depois, checamos se os robôs possuem a mesma abscissa. Caso não possuam, obtemos a tangente do ângulo formado entre os segmentos A e B da seguinte forma:

```
if(adv[i][0] - jogx != 0)
    m = float(adv[i][1]-jogy)/float(adv[i][0]-jogx);

float alfa = asin(18/dist);
float beta = atan(m);
```

  Caso possuam, a variável m terá um valor indeterminado e, consequentemente, a variável beta também. Constatando que possuem a mesma abscissa, verificamos se o adversário está acima ou abaixo do jogador em análise, para que o sinal do arco-tangente de 90 graus seja atribuído corretamente.

```
if(adv[i][0] - jogx == 0){
    if (adv[i][1] > jogy)
        beta = 1.5708;
```

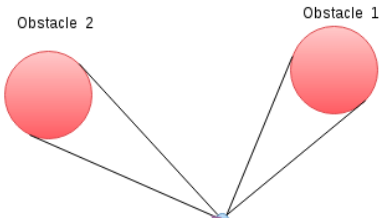
```
        else
            beta = -1.5708;
    }
```

4. Após tais cálculos, somos capazes de determinar com precisão os dois pontos onde as retas que passam pelo centro do robô em analise tangenciam a circunferência com raio de 18 u.c. formada pelo robô adversário, bastando apenas unificar da seguinte forma as variáveis encontradas anteriormente:

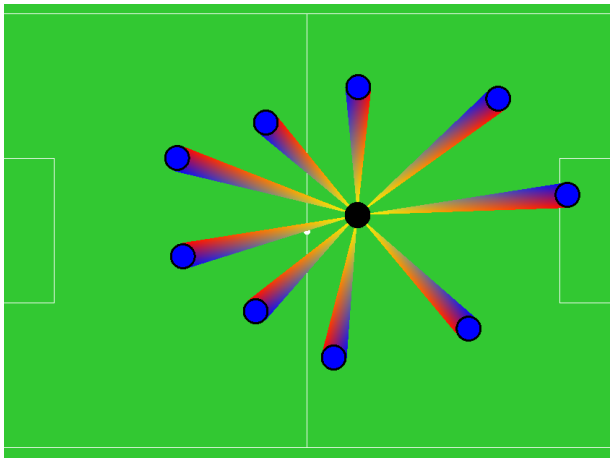
```
ponto[i][0] = jogx + int(batata*dist2*cos(beta+alfa));
ponto[i][1] = jogy + int(batata*dist2*sin(beta+alfa));

ponto[i][2] = jogx + int(batata*dist2*cos(beta-alfa));
ponto[i][3] = jogy + int(batata*dist2*sin(beta-alfa));
```

   Onde *i* refere-se ao adversário em análise.
   Assim, temos como resultador algo similar ao exibido abaixo.




5. Por fim, em um *loop* onde cada adversário é considerado, construímos, com ferramentas do *SFML*, um triângulo preenchido cujos vértices são o centro do robô, os pontos de tangência superior e o inferior à circunferência formada em torno do adversário analisado, como mostrado na imagem abaixo.



   Como a velocidade com que o código é processado é muito alta, todos os triângulos aparecem na tela de forma simultânea, como se fossem previamente construídos.
   A cada iteração de execução do código, todas as variáveis são recalculadas, permitindo a movimentação dos *bottons* na tela pelo interlocutor.