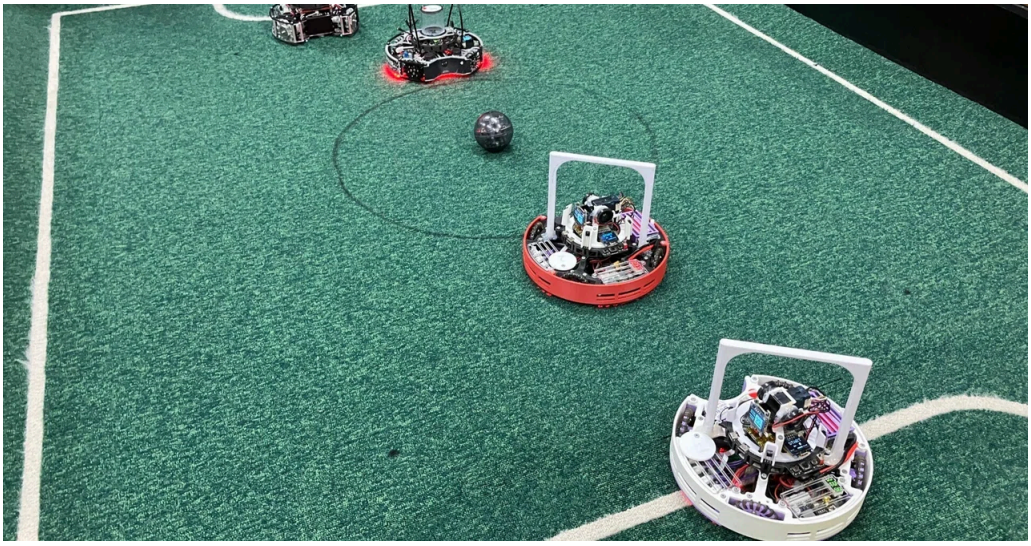


Edgeのディフェンス機のソフト紹介

しおから
2024年12月14日 21:02

まえがき

こんにちは。11月進研模試が上振れて嬉しいしおからです。今回は、Edgeのディフェンスのソフトの全てを紹介していこうと思います。
いよいよ今シーズンもRCJ始まりましたね。OBになってから見る高度な情報戦、めっちゃ面白いです。自分も感化されてちょっとプログラムの構造をいじったりもしてます。ところで、今のライトウェイトでは、ゴール前の白線に留まるタイプのディフェンス機を採用しているチームはかなり少ないように感じます。確かに何をやっているかわからんという意味で敬遠したくなる気持ちはわかるのですが、「ゴールの前の白線でボールを追い続ける」ということは実はかなり簡単に達成できます。
自分はディフェンスを書く能力しかないので、これが自分の実質全てみたいなところはあります。是非最後まで読んでください。ちなみに画像はリツモリカップで宗高アマテラスとやった時の写真です。3月のリツモリカップきてね！

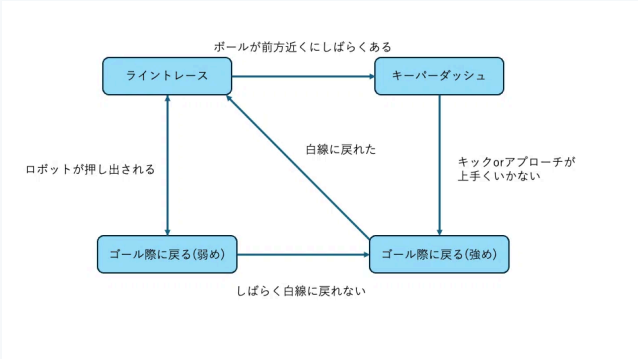
目次

ソフトの構造

まずはソフトの構造を紹介します。自分はソフトの構造にかなりこだわっていて、ステートマシンを使っています。これが使いやすくて、ロボットが今どの動きをしているかすごく分かりやすく、例えばライントレースをしているときおかしい動きをしたら、そのステートを治せばいい！みたいな感じで簡単にデバッグができます。

ステートマシン図

これがステートマシン図です。もうちょっと複雑なものになるかと思っていたのですが、細かくて例外的なときにしか使わないステートとかを抜いてみると意外と簡単でした。まあディフェンスは条件付けが単純なもので解決するので、言われてみれば妥当ですね。（まあ細かいバグを抜いていくことがディフェンスの強さそのものになるのですが…）



ステートマシン図

各ステートの紹介

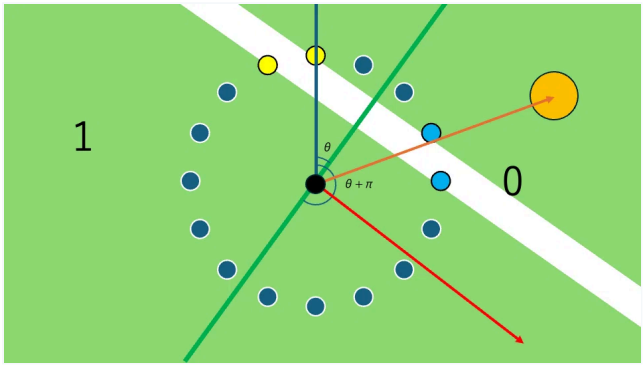
ということで各ステートの内容の解説をしていきます。
ちなみに、「ゴール前に戻る」が強めと弱めで分かれているのは、例えば押し合いでペナルティエリア内に入ったのと、キーパーダッシュでゴール前から離れたとでは意味が全く違ってくるからです。前者が弱めで、後者が強めの処理をしてます。

ライントレース

ここがメインでもあります。一応前の記事でもちょろっと紹介はしているのですが、同級生からわからんって言われたのでもう少し丁寧に書いてみます。前提条件として、「ラインとレースをする」というのは「ロボットの中心を白線上にとどめつつ、白線と垂直に移動する」と言い換えられること、そしてラインセンサーの記事で上げた「白線に対するベクトル」を出しているものとします。また、このベクトルを考えるとき、円形ラインセンサーの半径は1とします。

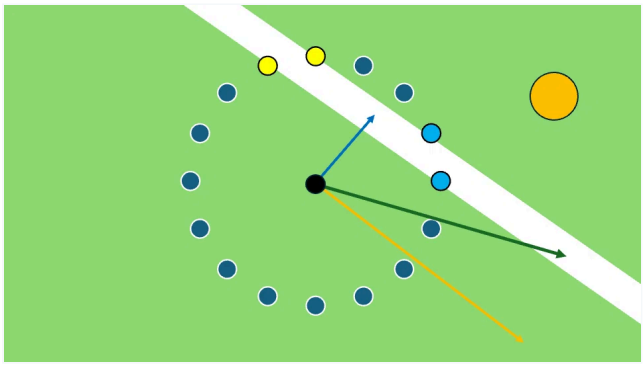
まず白線に対するベクトルを、x成分とy成分に分解し、$\vec{l} = (l_x, l_y)$ と置きます。これは、「ロボットの中心を白線上に留めるためのベクトル」です。また、これとロボット正面方向の偏角を θ とおきます。次に、$\vec{l}$ と垂直なベクトルを考えます。これは「白線と垂直に移動するためのベクトル」です。

また、図のように考えると、ボールの位置を元に左右のどちらかに動けばいいかを判定できます。書き下したプログラムでそこら辺はわかりやすくなっているので、そこを参照してください。



ボールの位置によって進行方向を決定する

これを $\vec{n}$ とおくと、$\vec{n} = (\cos(\theta + \pi/2), \sin(\theta + \pi/2))$ となります。そして、求めるベクトル $\vec{x}$ は、$\vec{x} = \vec{l} + k\vec{n}$ となるので、成分を計算して終了です。コードを書き下すとこんな感じになります。



ベクトルの合成によってラインとレースを行う

```
float go_x,go_y;                //進むベクトルの成分
float go_angle;                 //進む角度
int go_flag;                    //左右どちらに進むかのフラグ

float l_x,l_y;                  //白線に対するベクトルの成分
float line_coeffident = 1.0;    //白線に留まる成分の強さ
float theta = atan2(l_y,l_x);   //白線に対するベクトルとロボット正面方向の偏角
```

```
float n_x,n_y;                  //白線と並行移動するベクトルの成分

float ball_angle;               //ボールへのベクトルとロボット正面方向の偏角

//ラインに並行なベクトルのうち、左右を決定するところ。ここでは区分けに使う角度を決定する

if(theta < 0){  //角度を-π~πに収めるための場合分け
  border_angle[0] = theta;
  border_angle[1] = theta + pi;
}
else{
  border_angle[0] = theta - pi;
  border_angle[1] = theta;
}

//ここでball_angleをgo_border[0]~go_border[1]+πに収める処理をする。本筋とは関係ないので割愛

if(border_angle[0] < ball_angle && ball_angle < border_angle[1]){  //ボールの角度を区分
  go_flag = 0;
}
else{
  go_flag = 1;
}

go_angle = go_border[go_flag] + pi/2;  //進む角度決定


n_x = cos(go_angle);                  //ラインと並行移動するベクトル決定
n_y = sin(go_angle);

go_x = n_x + l_x * line_coeffident;    //ラインと並行移動するベクトルと白線に留まるベクトルを
go_y = n_y + l_y * line_coeffident;
```

キーパーダッシュ

これは本当にシンプルです。まず、ボールとの距離が閾値より近く、方位角の絶対値が閾値より小さければ、ボ

ゴール際に戻る（弱め）

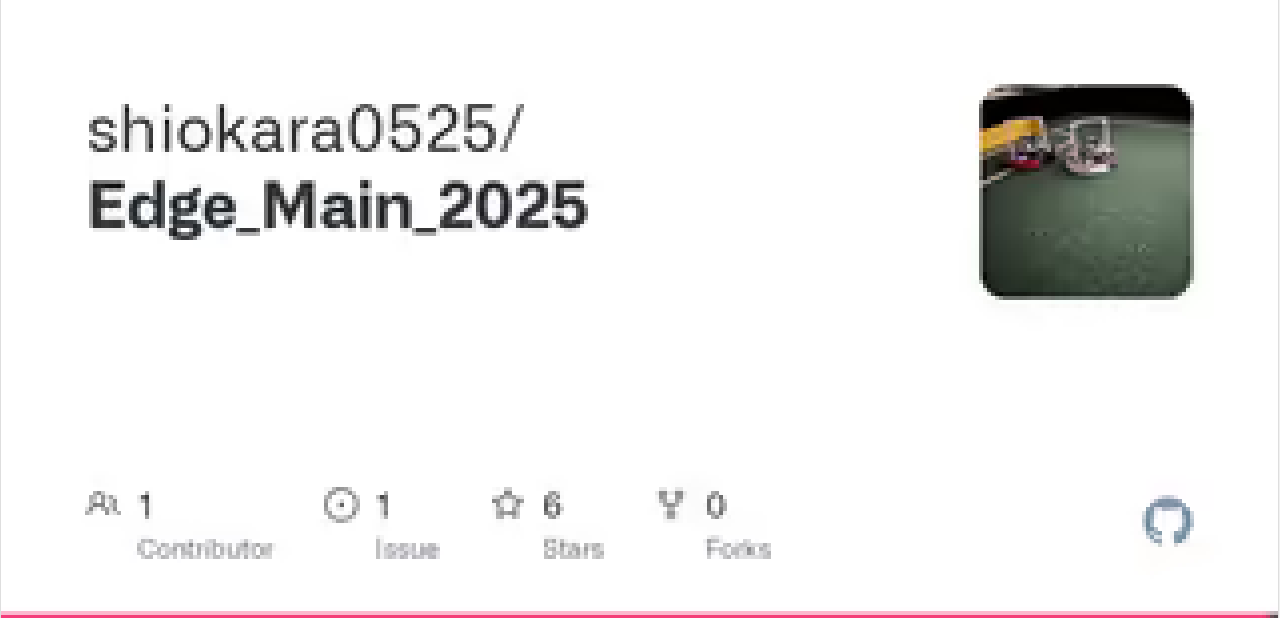
このステートは、ゴール前の白線からちょっとはみ出てしまったような状況を想定しています。この時は、前述した「ラインに対するベクトル」に従って動けば戻れます。例えば相手アタッカーに押されてペナルティエリア内に出た状況だと、最後にディフェンスが読んだラインに対するベクトルは前方を向いているので、これで戻れることがわかりますね。

ゴール際に戻る（強め）

ここで想定しているのは、キーパーダッシュやマルチプルディフェンス、またアウトオブバウンズでロボットがゴールから離れたところにいる状況です。この時は、ロボットについている後ろカメラから見たゴールで戻り方を判定します。計算式を示してもいいのですが、人によってカメラの画素数とかがややこしいので示さないでおきます。

最後に

こんな感じでEdgeのディフェンスは動いています。思ったよりもシンプルなものだったと思います。全部情報を公開するかちょっと迷ったのですが、競技の趣旨や、何よりもこの情報でよりRCJサッカーが面白くなってほしいなと思って公開しました。最初にもあるのですが、自分が優れているのはディフェンスの技術くらいなので、ぜひTwitterのDMやリプ欄で質問をもらえると嬉しいです。
最後にgithubのリポジトリを貼っておきます。lib->process->defence.cppで見れます。暇なら見てください。



shiokara0525/Edge_Main_2025

Contribute to shiokara0525/Edge_Main_2025 development by creating an account on GitHub.

GitHub