Trabalho Final – Competição *Smart Fool* em ação

Alisson R. Santos[1] , Mateus R. Martins. [2]

[1]Departamento de Ciências da Computação – Universidade Federal de Minas Gerais (UFMG)

[2]Engenharia de Controle e Automação - Graduação – Universidade Federal de Minas Gerais

alissonrs@ufmg.br, martinsrmateus@gmail.com

1. Introdução

A proposta deste último trabalho prático foi a de juntar todo o conhecimento adquirido nas experiências dos trabalhos anteriores, a fim de, adaptar o ultimo Robô construído para desempenhar tarefas de grande importância para um bom desempenho na competição de Robô de LEGO autônomos realizada no auditório do Instituto de Ciências Exatas da UFMG. Para isso foram consideradas algumas funcionalidades básicas a fazerem parte do projeto do Robô.

- Utilizar diversas capacidade sensoreiais
- Controlar corretamente a velocidade, posição e orientação do Robô
- Navegar em ambientes estruturados
- Identificar e coletar apenas os objetos de interesse
- Executar eficientemente a missão a partir de planos/estratégias

De forma mais específica, o Robô foi modificado de modo a ser capaz de se orientar em um campo como o da **Figura 1.1** abaixo. Tendo como principais objetivos:

- Obedecer ao sinal de luz de partida. Uma lampada posicionada em cada base nos extremos do campo.
- Derrubar as torres do lado oposto (blocos cinza abaixo dos verdes ou azuis)
- Coletar os blocos (verdes ou azuis) sobre as torres do lado oposto
- Coletar os blocos (vermelho ou amarelo) representantes do Rei adversário
- Se coletados os blocos, retornar com os mesmos para a prisão (cantos direito e esquerdo do próprio lado)
- Interromper a execução de todos os processos e parar os motores após 60 segundos contados a partir da recepção do sinal de partida.
- E por ultimo mas não menos importante, ser capaz de atravessar a ponte no centro que divide o campo em duas partes.

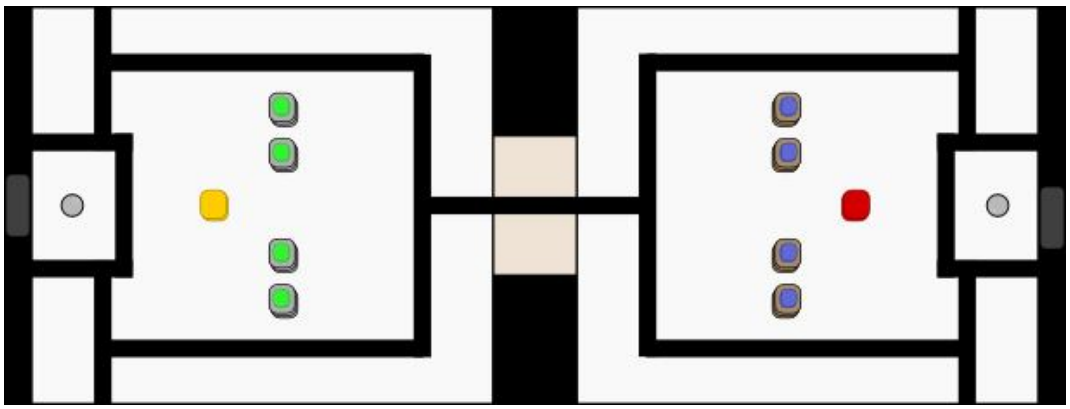

Figura 1.1 – Campo da Competição

2. Melhorias no Robô

Para desempenhar as atividades de acordo com os objetivos da competição introduzidos no item anterior, foram necessárias algumas modificações no Robô. Mecanicamente, o esquema de redução no eixo de tração se manteve, o que manteve também a resolução dos *shaft encoders*.

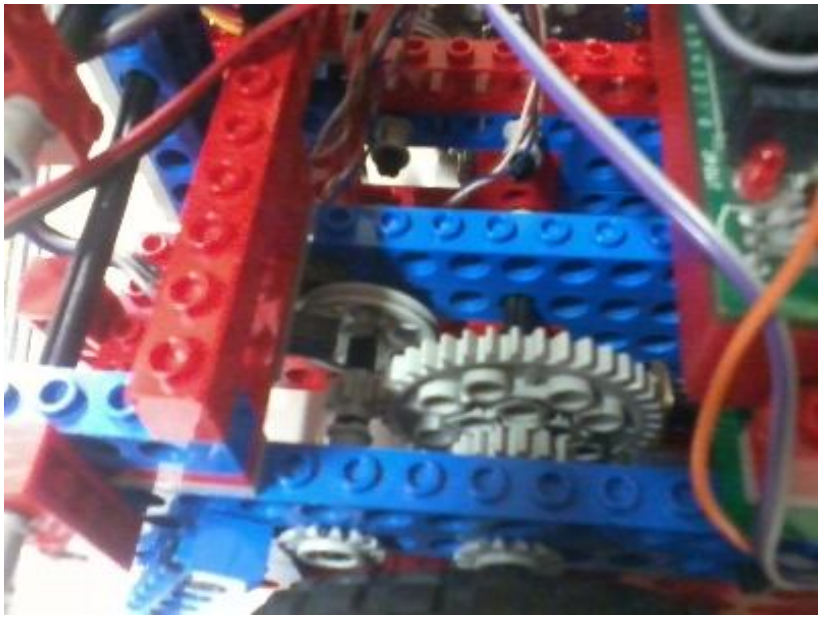

Figura 2.1 – Esquema de redução com shaf encoder

Entretanto, para a tarefa de atravessar o campo foi necessário modificar as rodas do Robô já que a ponte conta com uma rampa de inclinação de 18º. Foram colocados dois pares de rodas grandes para melhorar a aderência. As rodas foram preenchidas com cabos de rede com a mesma finalidade. A figura abaixo exibe em destaque as rodas no lado direito do Robô. O lado esquerdo é simétrico.

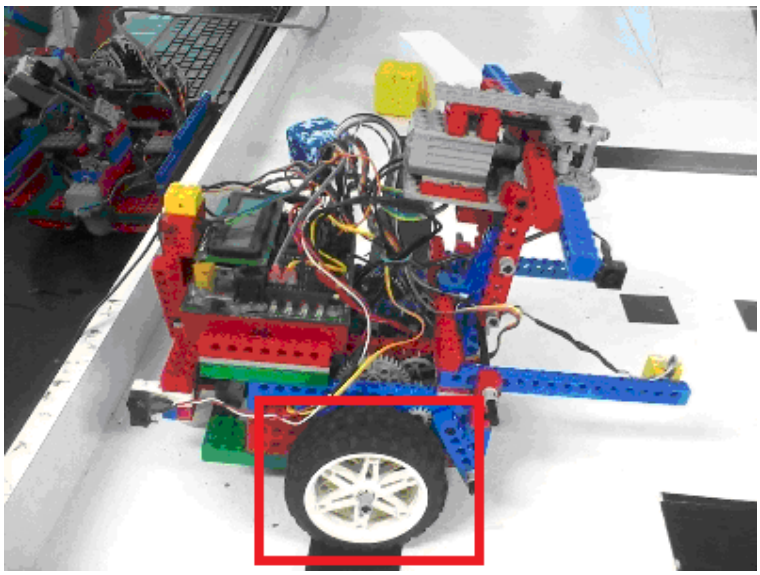

Figura 2.2 – Em destaque o par de rodas do lado direito

Outras melhorias feitas, dizem respeito aos mecanismos de inter-travamento. O Robô ficou mais firme. A figura abaixo destaca alguns desses pontos.

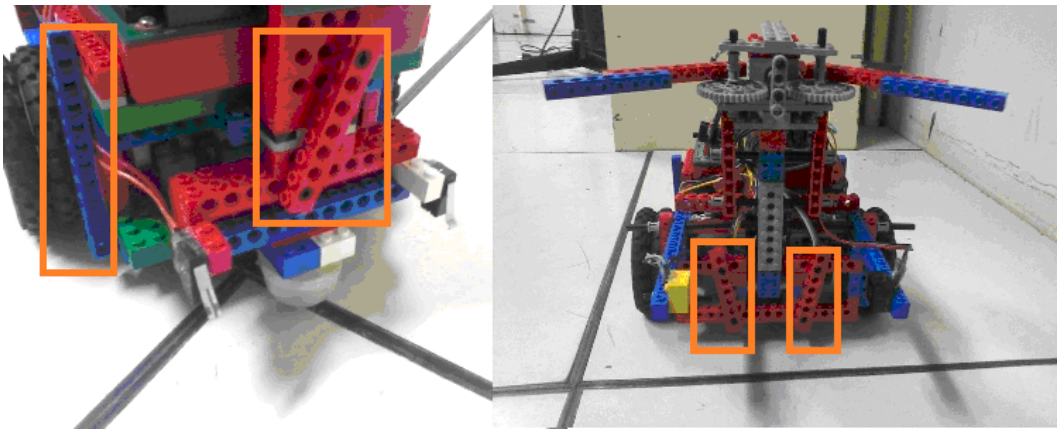

Figura 2.3 – Algumas partes inter-travadas do Robô maior confiabilidade construtiva

Optou-se por manter o eixo de tração centralizado para facilitar a execução de movimentos angulares. E a *Handyboard* foi deslocada levemente para trás. Isso foi necessário para dar estabilidade na descida da rampa, evitando tombamento.

3. Sensores

Sensor de partida – Foi montado na parte inferior do Robô um sensor receptor de luz utilizando um LDR para ser utilizado como sensor de partida. Para iniciar o Robô o sensor foi utilizado para identificar no chão do campo, logo abaixo do Robô, o acendimento de uma luz de sinal de início. Trata-se de um sensor digital o que garante o reconhecimento da luz de forma mais precisa. O mesmo sensor foi utilizado no trabalho prático 3.

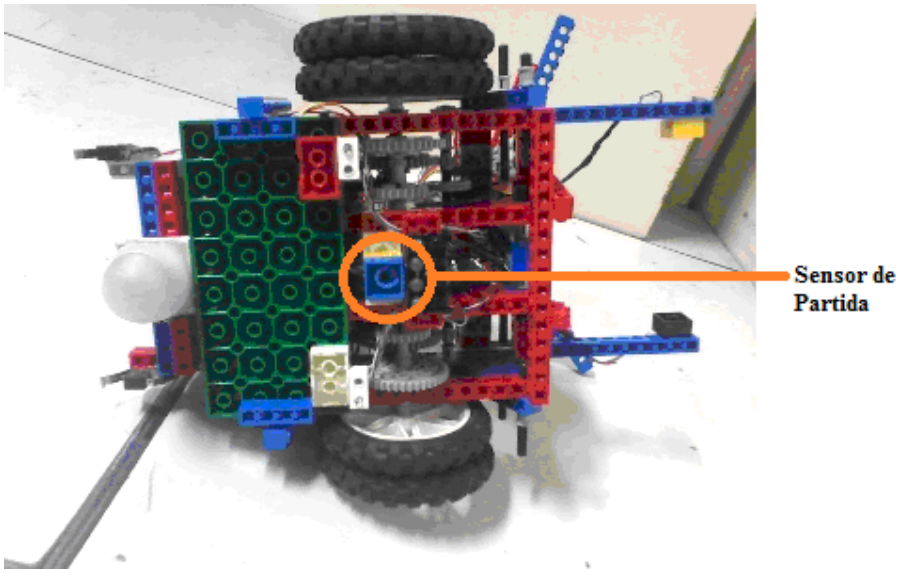

Figura 3.1 – Sensor de partida

Sensor Diferencial (polarizador) – O sensor de detecção da luz polarizada foi o mesmo dos trabalhos anteriores inclusive a calibração, apenas a posição na nova estrutura do Robô mudou

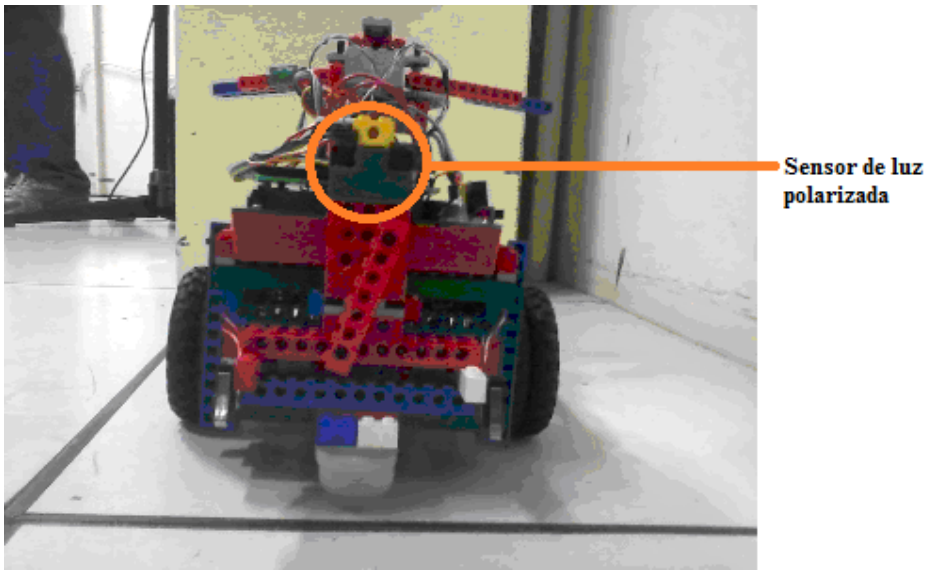

Figura 3.2 – Sensor polarizador

Sensores de toque traseiros (bumpers) – Foram montados na parte traseira do Robô dois sensores digitais de toque em lados opostos para alinhamento do Robô com a parede após a detecção da luz polarizada, ou seja, a direção de deslocamento desejada. Esse alinhamento foi de fundamental importância para que o Robô pudesse seguir em direção a ponte centralizada.

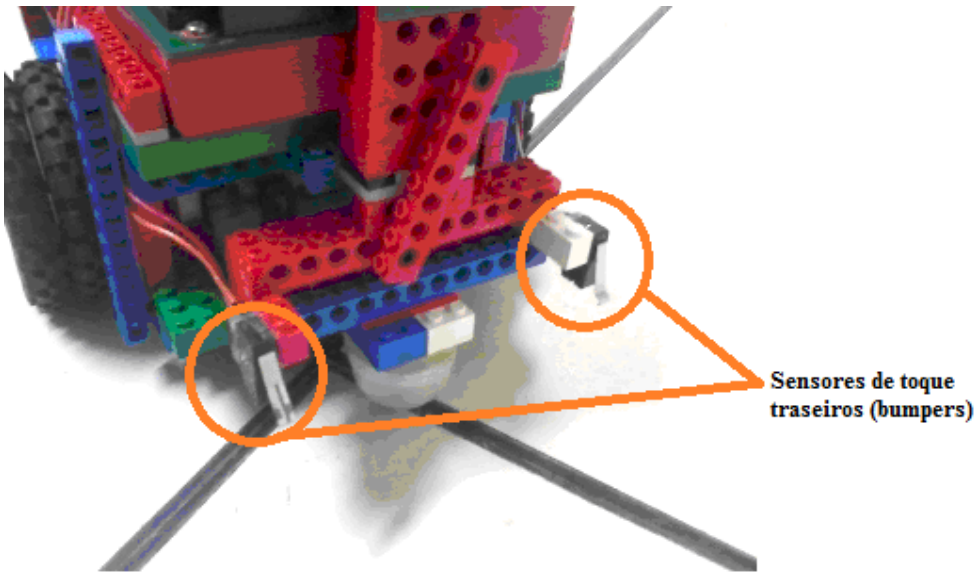

Figura 3.3 – Sensores de toque traseiros

Sensore de detecção de obstáculo – Este sensor consiste de um LED alinhado com um receptor LDR posicionados em lados opostos na parte da frente do Robô e apoiados em longos "braços". A detecção é analógica e funciona da seguinte forma. Caso um dos blocos das torres invada o espaço entre os braços, o sinal cai bruscamente caracterizando a detecção de obstáculo.

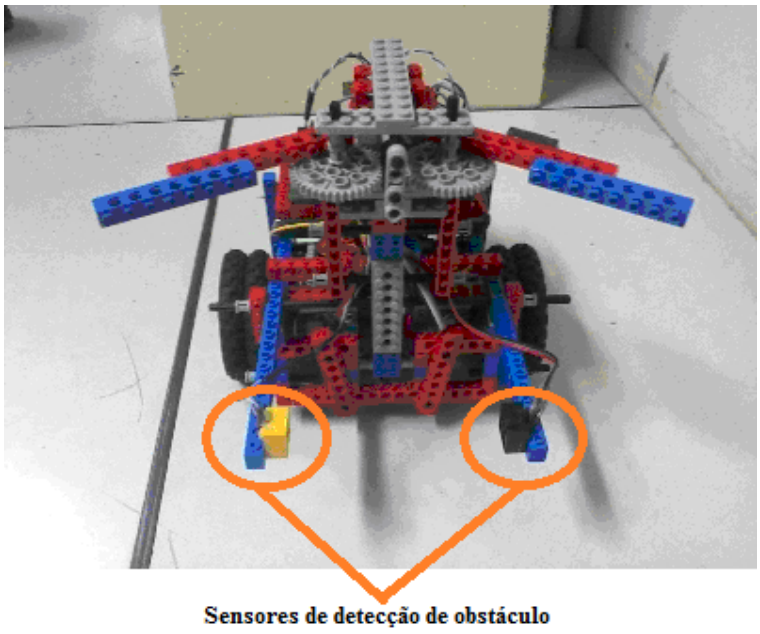

Figura 3.4 – Sensores de detecção de obstáculo

4. A Garra

Para a coleta de blocos no campo, foi utilizada uma garra com acionamento motorizado através de duas engrenagens girando em sentido oposto.

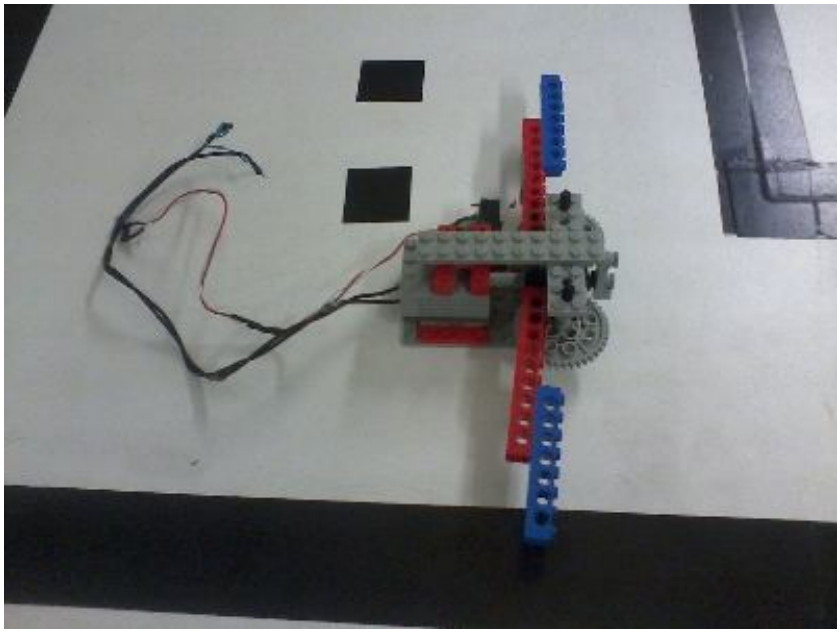

Figura 4.1 – A Garra

Esta garra foi montada na parte superior dianteira do Robô em altura calculada de modo que fosse capaz de coletar os blocos posicionados em cima das torres. O inconveniente desta garra é sua capacidade limitada de recolher apenas um bloco por vez.

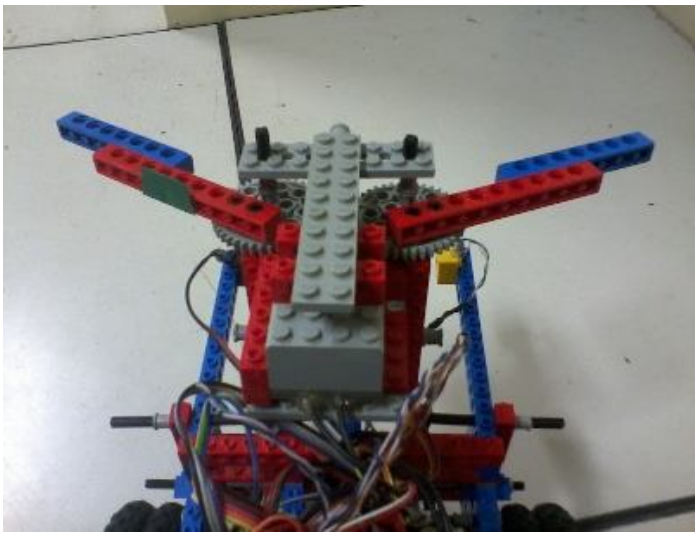

Figura 4.2 – Vista anterior da garra semi-aberta

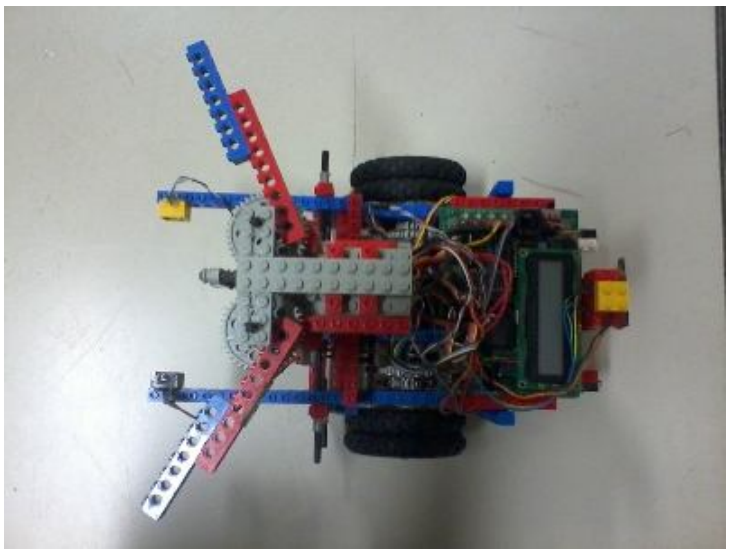

Figura 4.3 – Vista superior da garra semi-aberta

O acionamento da garra foi feito via linha de código em conjunto com o sinal de detecção dos sensores de detecção de obstáculo descritos anteriormente.

5. Estratégia

Definidas a pontuação das atividades necessárias ao bom desempenho na competição a equipe tratou de elaborar uma estratégia objetivando a conquista certa de pontos e priorizando a simplicidade na implementação. Para isso houve duas etapas ao longo do desenvolvimento deste projeto final.

5.1 Estratégia inicial

Inicialmente com o Robô do Trabalho Prático 3 em mãos e funcionando, a estratégia era, executar movimentação pelo campo via *Wave Front*, Atacar as torres inimigas com a catapulta e defender as próprias torres. Para isso, as únicas modificações necessárias provavelmente seriam a criação de um sistema de carregamento automático de munição para a catapulta e algum mecanismo de defesa das torres como, por exemplo, um escudo retrátil. Entretanto, algumas especificações da prévia da competição incluíam algumas atividades que o Robô do TP3 não era capaz de realizar, como por exemplo, subir a rampa, detectar e coletar bloco do outro lado. Por isso foi necessário repensar a estratégia. Daí o motivo para as melhorias descritas nos itens anteriores.

5.2 Estratégia após especificação da prévia

A nova estratégia pensada foi, avançar sobre o campo inimigo utilizando o sensores de segue-linha, capturar um bloco refém sobre uma das torres, no lado inimigo, derrubar todas as torres inimigas com o próprio movimento do Robô. As sucessivas tentativas frustradas de se utilizar o sensor de segue-linha diferencial para a movimentação sobre a ponte fizeram que a estratégia de avanço se modificasse passando a ser utilizado o *Wave Front* juntamente com o controle PD como mecanismo.

De fato após definida estratégia, após as horas de tentativa de uso do segue-linha e após horas de implementação do código final para enviar o Robô à competição esperou-se a correta execução dos seguintes passos:

- Robô calibrado via *menu* escolhendo campo Verde ou Azul;
- Inicio da movimentação após sinal luminoso no chão do campo;
- Procura pela luz polarizada, procurando o valor mínimo ou máximo de intensidade dependendo do campo definido;
- Após achar a luz polarizada, movimento para trás buscando alinhamento com a parede do campo através dos sensores de toque;
- Depois de alinhado, movimento para frente sustentado pelo bom controle PD feito e pré-calculo da distancia a percorrer;
- Ao chegar ao campo adversário, movimento angular para a direita e movimento em frente até a detecção de algum obstáculo (torre inimiga);
- Ao detectar a torre fechamento da garra para coleta do bloco (porquinho) sobre a torre;
- Coletado o bloco, movimentação para frente, meia volta e nova movimentação para frente até o outro lado do campo adversário com o objetivo de "varrer" as torres inimigas. Repetindo este passo até se esgotar os 60 segundos previstos.

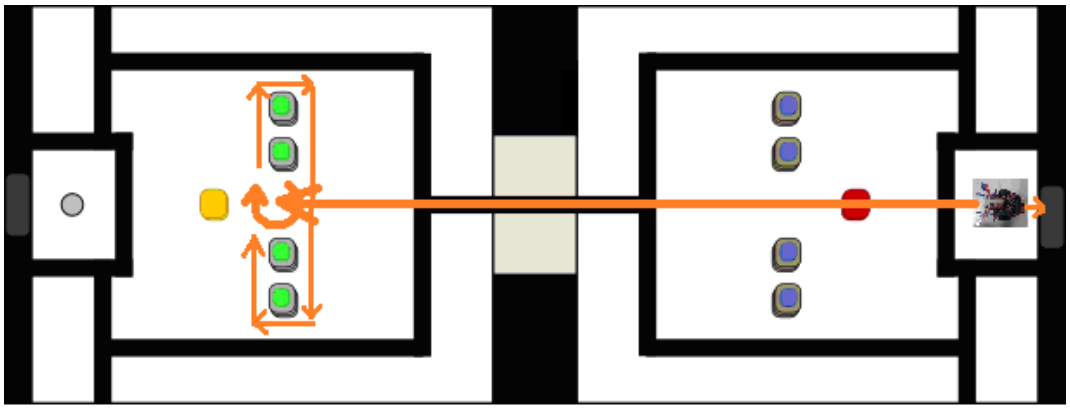

Figura 5.1 – Traçado planejado na competição

6. A Competição

Na competição o Robô *Smart Fool* disputou quatro partidas. A execução das tarefas previstas não ocorreu como esperado mas o desempenho em termos de pontuação não foi dos piores.

Na primeira partida o Robô não respondeu ao comando do sensor de partida, pois foi posicionado fora da posição correta pelo integrante da equipe, Alisson, ficando assim parado em sua base o que resultou em derrota na partida e menos um ponto de saldo.

Na segunda partida o Robô não conseguiu atravessar a ponte para o campo adversário devido a uma ruga de uma fita adesiva, não prevista nos testes anteriores à competição. A ruga era elevada e tinha aspecto de um quebra-molas para o Robô. Este ao não conseguir prosseguir adiante acabou mudando de direção e retornando ao próprio

campo até se chocar a um segundo do fim do tempo, com uma de suas torres, dando ponto ao adversário que venceu esta partida também.

Na terceira partida o Robô novamente não consegui atravessar a ponte devido a ruga da fita adesiva e ao bloco do Rei posicionado à sua frente. E ainda derrubou uma de suas próprias torres, entretanto, o adversário derrubou duas de suas torres, portanto a vitória foi do *Smart Fool* por 2x1.

Na quarta partida o Robô adversário acabou caindo na vala lateral à ponte e o *Smart Fool* novamente não conseguiu atravessar, dessa vez devido ao Rei que ficou enguiçado sob o Robô fazendo o mesmo perder contato com o solo e ficar com as rodas girando em falso. Como ninguém derrubou torres, foi declarado empate por 0x0.

A pontuação final foi de 4 com saldo de -1.

7. Conclusão

Após mais um árduo trabalho concluímos:

Sobre a preparação para a competição, foi difícil pensar em um Robô preparado para execução de todas as tarefas contempladas na competição por isso optou-se pelo simples, isto é, tentar conquistar pontos de alguma das maneiras possíveis. Tivemos vários problemas dado que a partir das definições da prévia, o Robô do Trabalho Pratico 3 com a catapulta teve que ser radicalmente modificado. Mas no fim um Robô compacto de dotado de vários sensores, mais veloz e mais forte foi o resultado.

Sobre a competição em si, concluímos que ficamos ao mesmo tempo frustrados e felizes com o resultado. Felizes, pois o controle de posicionamento programado para o Robô se mostrou bastante robusto garantindo uma execução em linha reta quase perfeita mesmo sem a técnica de seguimento de linha o que foi um tanto arriscado para a competição. Mas, frustrados devido a alguns problemas que ocorreram nas partidas, como a não partida na primeira, a ruga da fita adesiva que atrapalhou a movimentação do Robô em duas partidas e o bloco do rei, que por estar posicionado à frente do Robô tirou o mesmo de contato com o solo em uma das partidas.

Para solucionar estes problemas seria necessário um melhor controle de tração e um detector de obstáculo mais baixo e móvel (para não se chocar com a rampa da ponte) para detectar o Rei e possivelmente desviar do mesmo ou até empurrá-lo para o lado. Em uma das partidas o Robô *Smart Fool* derrubou uma de suas próprias torres no segundo final da partida, pois ficou se movimentando no próprio campo, talvez fosse necessário colocar um contador que parasse o Robô caso não fosse detectado nenhum bloco em 30 segundos (tempo suficiente para chegar ao campo adversário).

Sobre o Projeto em geral (incluindo os três trabalhos práticos e a preparação para a competição) é possível dizer que ficamos bastante satisfeitos com tudo que aprendemos, sobre Robôs autônomos, e principalmente por termos tido a oportunidade de aplicar os conceitos teóricos comprovando que na prática o que parece bastante fácil pode se tornar o maior dos problemas na implementação de um Robô de LEGO autônomo como, por exemplo, a movimentação em linha reta utilizando controle PD no TP1 que só foi alcançada com extrema satisfação no TP3, a detecção de cor do TP2 que se mostrou difícil pela influencia da luz ambiente, e a detecção de linha na previa para a competição que se mostrou difícil quando influenciada pelo controle PD.

Mesmo com essas dificuldades desafiadores encontradas e as dezenas de horas de trabalho no laboratório de Robótica até de madrugada o saldo foi bastante positivo em termos de aprendizado. O trabalho exige muita dedicação mas os resultados são compensatórios.

8. Código-fonte

TP4.c

```
#include myBiblioteca.ic
#include constantes.ic
#include auxiliar.c
#include polarizador.ic

int pidTime;
int pidWave;
float tempo0 = 0.0;
float tempo1 = 0.0;

void main(){
    sleep(2.0);

    printf("Aguardando . . .Press Start\n");
    beep();
    while(!start_button());
    while(1){
        menu();}
}

/*Funcao principal que devera somente chamar o menu
sendo funcao deste selecionar a acao*/

void menu(){

    int opcao = 0;
    int sair = FALSE;
    int button = -1;
    int n = 3;
    char strOpcao[3][16] = {"Calibrar", "Jogar", "Sair"};

    sleep(2.0);

    while( sair == FALSE){

        printf("Stp next Star OK");
        printf(strOpcao[opcao]);
        printf("\n");

        button = waitingButtonPress();
        if (button == START){
            if(opcao == 0){
                message("Selecionado Calibrar\n");
                calibrar();
            }
            else if(opcao == 1){
                message("Selecionado Jogar\n");
                waitingLightStart();
            }
            else if(opcao == 2){
```

```
                message("Selecionado Sair\n");
                sair = TRUE;
            }

        }
        else if (button == STOP){
            opcao = (opcao + 1)% n;
            //message("Opcao = %d\n",opcao);
        }
        button = -1;
    }

}

void calibrar(){ //Chama as funoes de configurao.
    configPolarizador();
}

void jogar(){

    polarizador();
    while((digital(10) != 1) || (digital(11) != 1)){
        if((digital(10) != 1) && (digital(11) != 1)){
            motor(0,-40);
            motor(1, -40);
        }
        else if ((digital(10) != 1) && (digital(11) == 1)){
            motor(0,-40);
            motor(1, 0);
        }
        else if ((digital(10) == 1) && (digital(11) != 1)){
            motor(0,0);
            motor(1,-40);
        }
    }
    ao();
    cruzaLinha();

    emFrente();emFrente();
    emFrente();
    //angular_dir(45.0);
    emFrente();
    emFrente();emFrente();emFrente();
    ao();
    direita();
    beep(); beep(); beep();
    pegaBloco();
    //Do outro lado
    //Segue at o outro lado
    while(analog(sensorInicio) < 200){
        vel_control(25);
    }
    printf("Aguardando . . .Press Start\n");
    emFrente();
    ao(); // Para na linha em frente a rampa


}
```

```
void waitingLightStart(){

    while(analog(sensorInicio)> 15); //Fica aguardando luz de inicio
    tempo0 = seconds();
    pidTime = start_process(jogar());

    controlTime();

}

void controlTime(){
    int continua = 1;
    while(continua == 1 ){
        tempo1 = seconds();
        if(tempo1-tempo0 >= 60.0){
            kill_process(pidTime); //QUAL PROCESSO MATAR.
            ao();
            beep(); beep();
            printf("Fim do tempo    [%d]\n");
            continua = 0;
        }
    }
    sleep(2.0);

}
```

Constantes.c

```
/* Arquivo com as constantes que definem elementos do programa */


//          CONSTANTES FISICAS DO ROBO
int sensorLinha = 6;
int sensorPresenca = 5;
int sensorCor = 4;
int sensorInicio = 3;
int sensorPolo = 2;
int shaftDireito = 1;
int shaftEsquerdo = 0;

int motor2 = 0; // Motor esquerdo
int motor1 = 1; // Motor direito

int ledVerde    = 0b00000100;
int ledVermelho = 0b00100000;
int ledAzul     = 0b00010000;


//          CONSTANTES LOGICAS DO PROGRAMA
int continua = -1;
int buttonStart = -1;
int buttonStop = -1;
int START = 1;
int STOP = 0;
int TRUE  = 1;
int FALSE  = 0;
float PAUSE = 1.0;
int LUZAMBIENTE = -1;


//#endif
```

myBiblioteca.c

```
#include constantes.ic

//-----------------------------------------------------------//
/*  FUNCAO AUXILIAR QUE MOSTRA UMA MENSAGEM E AGUARDA
    recebe um array de chars com \n ao final
    Nova implementacao de printf() com tempo para ler e
    sinal sonoro.
*/
void message(char msn[]){
    printf(msn);
    beep();
    sleep(PAUSE);
}


//-----------------------------------------------------------//
/*

FUNCOES QUE AGUARDAM O PRESSIONAMENTO DE ALGUM DOS BOTOES

As funcoes abaixo aguardam o pressionamento de algum dos
botoes start ou stop em uma thread paralela de execucao

*/
int waitingButtonPress(){
    int pidStart = 0;
    int pidStop = 0;
    int valueReturn = -1;

    pidStop = start_process(waitingStop());
    pidStart = start_process(waitingStart());
    while((buttonStart != 1) && (buttonStop != 1)){
        sleep(0.5);
    }
    kill_process(pidStop);
    kill_process(pidStart);

    if(buttonStart == 1){
        valueReturn = START;
    }
    else if (buttonStop == 1){
        valueReturn = STOP;
    }

    buttonStop = 0;
    buttonStart = 0;

    return valueReturn;
}


void waitingStop(){
    stop_press();
    buttonStop = 1;
    //message("Stop pressed. waitingStop\n");
    beep();
}
```

```
void waitingStart(){
    start_press();
    buttonStart = 1;
    //message("Start pressed. waitingStart\n");
    beep();
}

//-----------------------------------------------------------//
/*

FUNCOES QUE IMPRIME O VALOR DE UM SENSOR ANALOGICO
CONTINUAMENTE.

A funcao impreme no display o valor do sensor analgico
da porta x, passada como entrada, continuamente a cada 0,1s.

A funcao deve ser usada como um processo a parte de modo
a facilitar a calibracao. Enquanto este processo mostra
o valor do sensor outro seta o valor como padro e mata o
processo atual.

*/

void mostraContinuamenteAnalogico(int x){
    if( (x >= 0) && (x <= 6)){
        while(1) {
            //Sensor na porta x
            printf("Sensor %d = %d\n", x, analog(x));
            sleep(0.1);
        }
    }

}
//-----------------------------------------------------------//
/*

Busca um valor com o sensor x

*/

int setAnalogValue(int sensor){
    int returnValue = 0;
    int pidMostra = 0;
    printf("Sensor %d       ", sensor);
    message("Capturar valor\n");

    message("Press stop for choose value;\n");

    pidMostra =
start_process(mostraContinuamenteAnalogico(sensorPolo));

    stop_press();
    returnValue = analog(sensor);
    printf("%d ",returnValue );
    kill_process(pidMostra);
    message(" = sensor\n") ;
    return returnValue;
}
//-----------------------------------------------------------//
```

```
/*

Captura luz ambiente.

*/

int getLuzAmbiente(int sensor){
    message("Funcao: Buscar  luz ambiente");
    setAnalogValue(sensor);
}


//-----------------------------------------------------------//
/*

Obtem modulo do inteiro num

*/

int mod2(int num){
    if (num <= 0) num = (num * -1);
    return num;
}
```

Controle.c

```
#include sencdr0.icb
#include sencdr1.icb
#include segueLinha.ic
int power0;
int power1;
/*
float data_erro[200];
int data_vel0[200];
int data_vel1[200];
*/
float mod(float num){
    if (num <= 0.0) num = (num * -1.0);
    return num;
}

/*
Funcao de controle de movimentacao linear
312 counts  = 30 cm.
52 counts = 5 cm. (Menor valor inteiro.)
13 counts = 1,25 cm.
10,4 counts = 1 cm.
1 count ~~  0,96 mm.

*/
void vel_control(int counts) {
    int d = 0;
    int mot0 = 1;
    int mot1 = 1;

    int SL;
    float erro0anterior = -999.0;
    float erro0 = 0.0;
    float erro = 0.0;
    float erroanterior = -999.0;
```

```
    float derivativo = 0.0;
    float derivativo0 = 0.0;
    float errodif = 0.0;
    float errodifanterior = 0.0;
    float derivativodif = 0.0;
    float integral = 0.0;

    float time1;
    float time2;
    float timedif;
    int count_ini0;
    int count_ini1;
    int count_atual0;
    int count_atual1;
    int vel_atual0;
    int vel_atual1;

    float Kp = 0.16;
    float Kd = 0.1;
    if(power0 == 0 && power1 == 0);{
        power0 = 85;
        //power0 = 100;
        power1 = 85;
    }

    encoder0_counts = 0;
    encoder1_counts = 0;

    while(mot0 == 1 || mot1 == 1) {
        if(encoder0_counts >= counts)
          {off(0);
            mot0 = 0;}
        else{
            //motor(0, power0);
        }
        if(encoder1_counts >= counts)
          {off(1);
            mot1 = 0;}
        else{
            //motor(1, power1);
        }


        motor(0, power0);
        motor(1, power1);
        time1 = seconds();
        //Le a contagem do encoder no incio do intervalo
        count_ini0 = encoder0_counts;
        count_ini1 = encoder1_counts;
        //gerando um atraso de meio segundo
        sleep(0.1);
        //L a contagem do encoder ao final do intervalo
        time2 = seconds();
        count_atual0 = encoder0_counts;
        count_atual1 = encoder1_counts;
        /*determinando da velocidade*/
        timedif = time2 - time1;
        vel_atual0 = (int)((float)(count_atual0 -
count_ini0)*(1.0/timedif));
```

```
        vel_atual1 = (int)((float)(count_atual1 -
count_ini1)*(1.0/timedif));
        if(erroanterior == -999.0)
          {erroanterior = (float)(count_atual1 - count_atual0);}
        else
          {erroanterior = erro0;}
        erro = (float)(count_atual1 - count_atual0);

        }
        derivativo = (erro - erroanterior)/timedif;
        //correção feita
        power0 += (int)(erro * Kp + derivativo * Kd);
        //}

        if (power0 > 100) {power0 = 100;}
        if (power1 > 100) {power1 = 100;}
        if (power0 < -100) {power0 = -100;}
        if (power1 < -100) {power1 = -100;}
        printf("%f\n",erro);
        /*
        if (d<200){
            data_erro[d] = erro;
            data_vel0[d] = vel_atual0;
            data_vel1[d] = vel_atual1;
            d++;
        }
        */
    }
}

//Codigo de controle de movimentacao angular
void vel_control_ang(int counts) {

    int d = 0;
    int mot0 = 1;
    int mot1 = 1;

    float erro0anterior = -999.0;
    float erro0 = 0.0;
    float erro = 0.0;
    float erroanterior = -999.0;
    float derivativo = 0.0;
    float derivativo0 = 0.0;
    float errodif = 0.0;
    float errodifanterior = 0.0;
    float derivativodif = 0.0;

    float time1;
    float time2;
    float timedif;
    int count_ini0;
    int count_ini1;
    int count_atual0;
    int count_atual1;
    int vel_atual0;
    int vel_atual1;
    int power0;
    int power1;
```

```
    //Modificados apenas os ganhos proporcional e derivativo
    float Kp = 0.01;
    float Kd = 0.001;
    if(power0 == 0 && power1 == 0 && counts !=0);{
        power0 = -50 * (int)((float)counts/mod((float)counts));
        power1 = 50 * (int)((float)counts/mod((float)counts));
    }

    counts = (int)(mod((float)counts));

    encoder0_counts = 0;
    encoder1_counts = 0;

    while(mot0 == 1 || mot1 == 1) {
        if(encoder0_counts >= counts)
          {off(0);
            mot0 = 0;}
        if(encoder1_counts >= counts)
          {off(1);
            mot1 = 0;}

        motor(0, power0);
        motor(1, power1);
        time1 = seconds();
        //Le a contagem do encoder no incio do intervalo
        count_ini0 = encoder0_counts;
        count_ini1 = encoder1_counts;
        //gerando um atraso de meio segundo
        sleep(0.1);
        //L a contagem do encoder ao final do intervalo
        time2 = seconds();
        count_atual0 = encoder0_counts;
        count_atual1 = encoder1_counts;
        /*determinando da velocidade*/
        timedif = time2 - time1;
        vel_atual0 = (int)((float)(count_atual0 -
count_ini0)*(1.0/timedif));
        vel_atual1 = (int)((float)(count_atual1 -
count_ini1)*(1.0/timedif));
        if(erroanterior == -999.0)
          {erroanterior = (float)(count_atual1 - count_atual0);}
        else
          {erroanterior = erro0;}
        erro = (float)(count_atual1 - count_atual0);
        derivativo = (erro - erroanterior)/timedif;
             //Corrige apenas para erros acentuados
        if(mod(erro)>10.0){
            power0 += (int)(erro * Kp + derivativo * Kd);
        }

        if (power0 > 100) {power0 = 100;}
        if (power1 > 100) {power1 = 100;}
        if (power0 < -100) {power0 = -100;}
        if (power1 < -100) {power1 = -100;}
        printf("%f\n",erro);
        /*
        if (d<200){
            data_erro[d] = erro;
            data_vel0[d] = vel_atual0;
```

```
            data_vel1[d] = vel_atual1;
            d++;
        }*/
    }
}
/*
Distancia entre eixos (referencia meio do pneu) = 18 cm
Raio = 9 cm.
Para 360 graus.
 Cada roda gira 1/2 volta  = pi * R ~28,2743 cm.
 Utilizando a relao (13 counts = 1,25 cm) chegamos a
 1/2 volta = 294,05 counts

*/

void angular_dir(float ang){

    int angle;
    int counts;
    int mot0 = 1;
    int mot1 = 1;
    //sleep(1.0);
    encoder0_counts = 0;
    encoder1_counts = 0;

    if(ang != 0.0){
        if(ang == 360.0)
          {counts = 240;}
        else if(ang == 180.0)
          {counts = 180;}
        else if(ang == 90.0)
          {counts = 55;}
        else if(ang == 10.0)
          {counts = 1;}
        else if(ang == 45.0)
          {counts = 18;}
        vel_control_ang(-counts);
        ao();
        sleep(0.2);
    }

    printf("%d e %d\n",encoder0_counts,encoder1_counts);

}

void angular_esq(float ang){

    int angle;
    int counts;
    int mot0 = 1;
    int mot1 = 1;
    //sleep(1.0);
    encoder0_counts = 0;
    encoder1_counts = 0;
    if(ang == 10.0)
      {counts = 1;}
    else if(ang == 180.0)
      {counts = 290;}
    else if(ang == 90.0)
```

```
      {counts = 55;}
    vel_control_ang(counts);
    ao();

    printf("%d e %d\n",encoder0_counts,encoder1_counts);

}

/*
Gira 90 graus a direita
*/
void direita(){
    angular_dir(90.0);
    ao();
}

/*
Gira 90 graus a esquerda
*/
void esquerda(){
    angular_esq(90.0);
    ao();
}

/*
Caminha 30 cm em frente
*/
void emFrente(){
    vel_control(150);
    ao();

}
```

polarizador.c

```
#include Controle.c
#include myBiblioteca.ic
//int sensorPolo = 2;
int campo = -1;
int VERDE = 1;
int AZUL = 0;
int button = -1;

void configPolarizador(){
    printf("Start - AZUL    Stop - VERDE\n");
    //sleep(0.2);
    button = waitingButtonPress();
    if (button == START) {
        campo = AZUL;
        message("Campo Azul\n");
    }
    else{
        campo = VERDE;
        message("Campo Verde\n");
    }
}


void polarizador() {

    int ang = 90; // Angulo usado para girar
```

```
    int v1 = 0;

    beep();
    v1  = analog(sensorPolo);

    //CAMPO VERDE
    if(campo == VERDE){
        while(v1 > 120){
            angular_dir((float) ang);
            v1 =  analog(sensorPolo);
        }
    }
    //CAMPO AZUL
    else{
        while(v1 < 170){
            angular_dir((float) ang);
            v1 =  analog(sensorPolo);
        }
    }

    beep();beep();
    printf("Posicionado!");
    //emFrente();

}
/*
void main(){
    configPolarizador();
    polarizador();
}*/
```

auxiliar.c

```
//#ifndef AUXILIAR
//#define AUXILIAR


#include constantes.ic
#include Controle.c
int ST = 0;
int buscaBloco = TRUE;
//-----------------------------------------------------------//
/*  FUNOES PARA USO NA COMPETIAO

*/
void cruzaLinha() {
    long sensorIni = 0l;
    long sensorLin = 0l;
    int continua = TRUE;
    int motorDir = motor1;
    int motorEsq = motor2;
    int i = 0;

    ST = start_process(    vel_control(900));

    while(analog(sensorInicio) < 200){
        sleep(0.2);
    }
    ao();//Primeira Linha
    kill_process(ST);
```

```
    printf("Achou Linha\n");

    ST = start_process(    vel_control(900));
    while(analog(sensorInicio) < 200){
        sleep(0.2);
    }
    ao();//Primeira Linha
    kill_process(ST);
    printf("Achou segunda Linha\n");

void pegaBloco(){
    voltaGarra();
    ST = start_process(temTorre());

    while (buscaBloco == TRUE ){
        emFrente(); emFrente();
        direita(); direita();
    }


    ao();
    kill_process(ST);

    prendeBloco();

    motor(0,-40);
    motor(1,-40);
    sleep(2.0);
    ao();

    motor(0,40);
    motor(1,40);
    sleep(2.0);
    ao();

    while (1){
        emFrente(); emFrente();
        direita(); direita();
    }
}




int temTorre(){ // funo que verifica se a torre passou pelo sensor de
presena.
    if(analog(sensorPresenca) < 100 ){ // Achou bloco.
        beep(); printf("TEM TORRE\n");
        buscaBloco = FALSE;
        return TRUE;
    }else{
        buscaBloco = TRUE;
        return FALSE;
    }
    sleep(0.2);
}

void voltaGarra(){
```

```
    motor(2,-40);
    sleep(1.0);
    ao();
}

void prendeBloco(){
    motor(2,40);
    sleep(2.0);
    ao();
}

/*
void main(){

    beep();

    printf("Aguardando . . . \n");
    beep();
    while(!start_button());

    voltaGarra();
    //Chama funo a testar

    cruzaLinha();
}
*/

//#endif
```

9. Referencias Bibliográficas

1 FOWLER, Martin. UML essencial: um breve guia para a linguagem-padrão de modelagem de objetos. Porto Alegre: Bookman, 2005

2 MARTIN. Fred G. Robotic Explorations: A hands-on introduction to engineering