Trabalho prático 3 – Controle Deliberativo e Lançamento de Objeto

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

Na proposta do terceiro trabalho foi requerido como atividades do robô móvel, ser capaz de escolher uma trajetória em um campo mapeado a partir de um ponto inicial e um ponto final escolhidos em tempo de execução, assim como possíveis obstáculos a serem incluídos no campo partindo o Robô após captar um sinal luminoso na superfície do campo. Foi requerido também para o Robô, ser capaz de seguir um luz polarizada igualmente ao trabalho prático 2 e após encontrar a luz, disparar automaticamente um leve projétil na direção da mesma.

Foi possível realizar tais atividades através, da adaptação de um algoritmo de movimentação (Wavefront) ao controle de movimento do Robô e da montagem de um mecanismo de disparo acoplado ao corpo do Robô com acionamento automático, sendo essa adaptação de código e a montagem do mecanismo a tônica das atividades realizadas neste trabalho prático.

2. Melhorias no Robô

Antes de dar início a realização das atividades objetivo deste trabalho prático foi necessário novamente fazer melhorias na estrutura do Robô com o objetivo de otimizar o controle PD implementado. O controle PD implementado anteriormente em código ic estava correto, entretanto a baixa resolução não permitia que o algoritmo fizesse pequenas correções de trajetória. Por isso foi necessário reconstruir toda a parte de tração do Robô e eixo de contagem de pulsos para o encoder. A nova estrutura conta com os motores acionando o eixo da roda por meio de parafuso sem fim e depois são feitas reduções de 40 para 16 e de 40 para 8 até o eixo onde fica a roda perfurada para contagem dos pulsos dos shaft encoders.

A Figura abaixo exibe a nova estrutura:

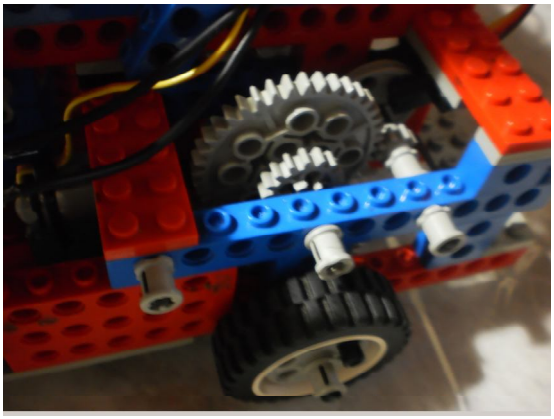

Figura 2.1 – Engrenagens no eixo de tração

Com base nessa estrutura é que foram redefinidos os novos parâmetros do controlador PD do mesmo algoritmo utilizado para o controle no trabalho pratico 2. As mudanças nos valores dos parâmetros do trabalho anterior para este não foram muito expressivas, entretanto, os resultados obtidos foram bastante diferentes considerando a situação de antes com uma contagem de 12 pulsos do encoder por volta para agora contando 144 pulsos por volta. Alguns dados coletados, até a obtenção dos melhores valores para os ganhos proporcional e derivativo, são exibidos nas figuras abaixo sendo o último o melhor resultado obtido.

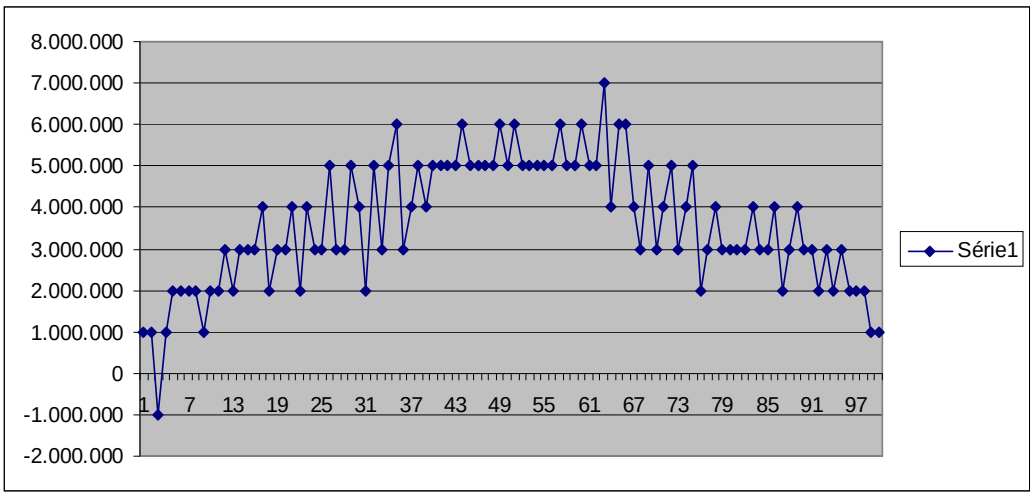

Figura 2.2 – Gráfico do erro de posicionamento com Kp = 0.1 e Kd = 0.01

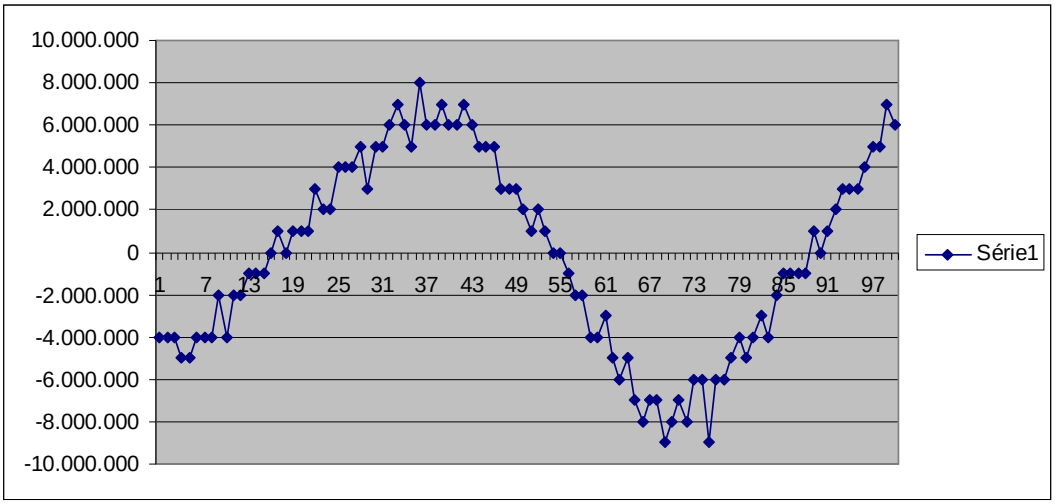

Figura 2.3 – Gráfico do erro de posicionamento com Kp = 0.2 e Kd = 0.01

Pelos dois primeiros gráficos é possível perceber que o ganho Kp deixou o sistema mais oscilatório. De fato essa tendência foi identificada com o aumento de Kp acompanhado de efeito desestabilizante. Por isso preferiu-se manter um Kp mais baixo.

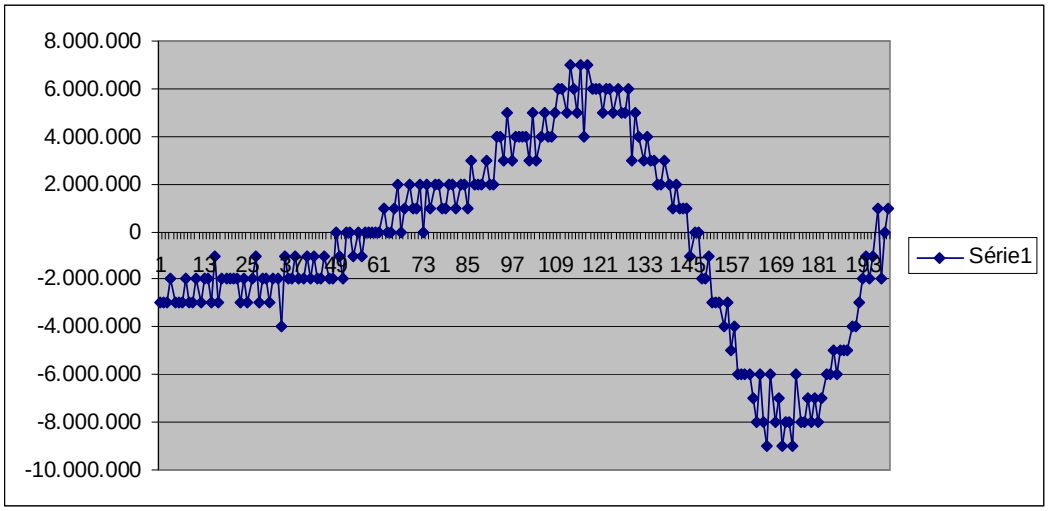

Figura 2.4 – Gráfico do erro de posicionamento com Kp = 0.1, Kd = 0.01 e Ki=0.002

No gráfico da Figura 2.4 acima foi inserido um ganho Ki tentando fazer a resposta atingir o set-point mais rapidamente, entretanto esse controle contou com um overshoot muito grande. Por isso o termo Ki foi removido e após variações de Kd foi possível notar que devido a natureza ruidosa do sinal de erro o melhor que pode ser feito foi o ajuste do Kp para o valor de 0.16 que como pode ser observado na Figura 2.5 abaixo, tem resposta oscilatória em torno do set-point mas com o menor overshoot conseguido.

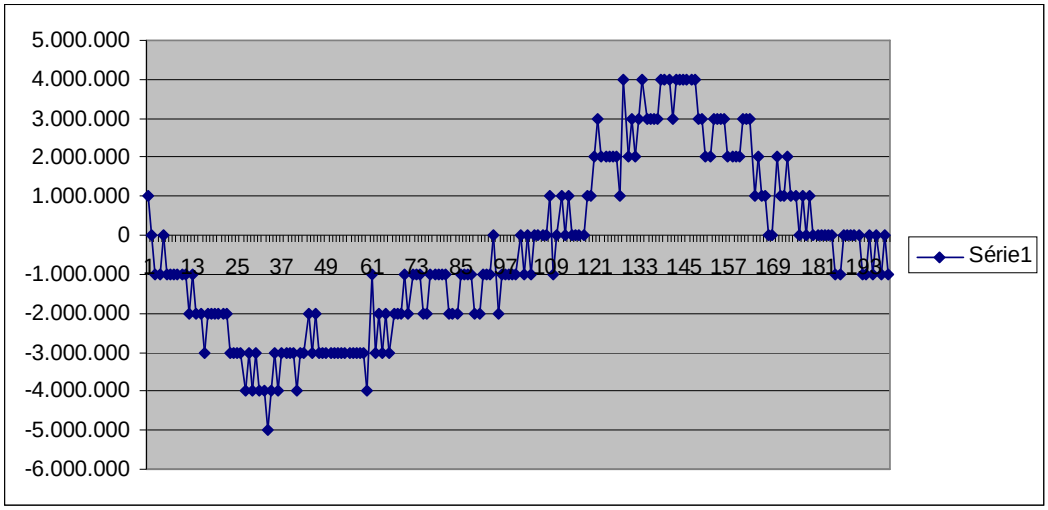

Figura 2.6 – Gráfico do erro de posicionamento com Kp = 0.16 e Kd = 0.01

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 no Controle Deliberativo 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.

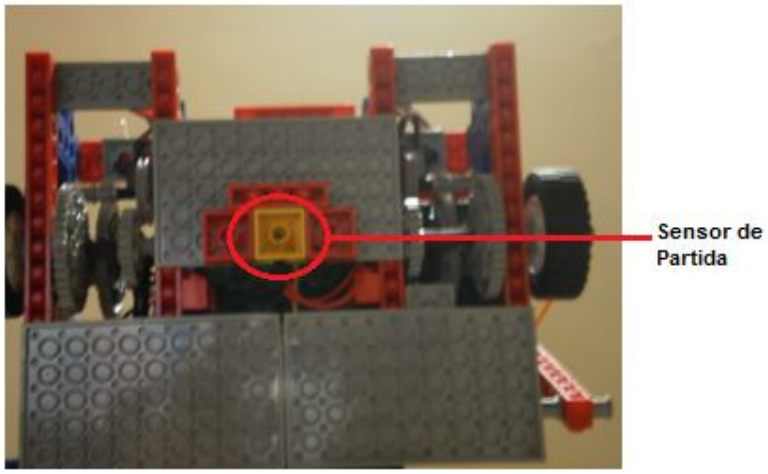

Figura 3.1 – Sensor de partida (a luz que vem debaixo me atinge)

Sensor Diferencial (polarizador) – Para este trabalho prático foi necessário fazer uso novamente do sensor diferencial (polarizador) construído para o trabalho prático 2 com a mesma finalidade de identificação no campo de uma luz polarizada pré-definida. Para o mesmo foi utilizada a mesma calibração e estrutura de montagem, apenas a posição no Robô mudou.

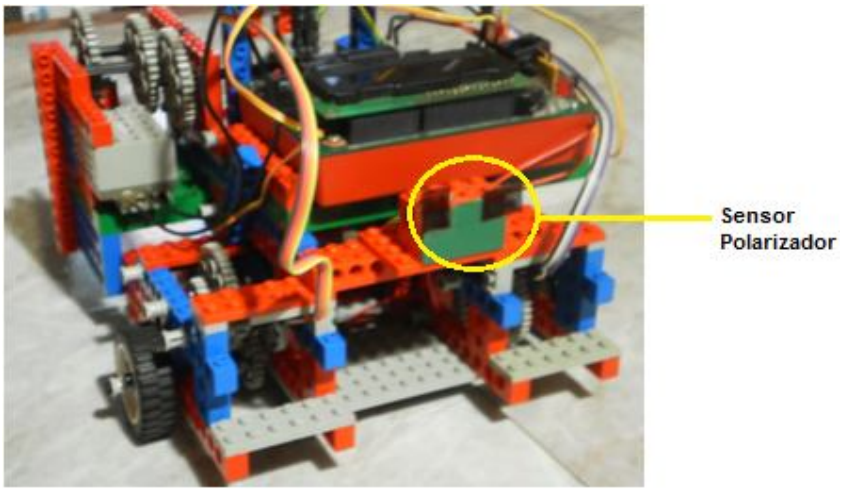

Figura 3.2 – Sensor polarizador

4. O Wavefront

O Wavefront, no contexto atual, trata-se de um algoritmo de identificação de um caminho a ser percorrido, pelo Robô autônomo, entre dois pontos pré-definidos em meio a um campo discretizado por um número pré-definido de células do mesmo tamanho podendo estas assumir dois valores, ocupada ou vazia. Ficando a cargo do algoritmo a identificação do melhor caminho entre os dois pontos de modo a se livrar das células ocupadas.

Foi fornecido o algoritmo do Wavefront pré-fabricado, restando como atividade à adaptação do código recebido ao código de movimentação do Robô e inserção desse código nas chamadas de menu do Robô.

Dentre as alterações no código, foi excluída a função *main()* e substituída por uma função chamada de *waveConfig()* tendo essa última, o mesmo papel desempenhado pela função original de definição das posições inicial e final do Robô e definição das posições dos obstáculos via interface da *Handyboard* com o usuário. Outra modificação importante nessa parte de definições de posicionamento foi feita na função *escolheMapa()* onde foram inseridas linhas de código forçando o mapeamento de obstáculos de forma a garantir que o campo de movimentação do Robô ficou de tamanho 4x4 ao invés de 10x10 conforme a definição original do programa.

Após essas definições iniciais o programa chama a função original para cálculo do caminho a ser feito e em seguida o Robô fica aguardando na posição sobre a luz de partida o acendimento de tal luz bloqueado na função *waitingLightStart()*.

Verificada a condição booleana de acendimento da luz é iniciado um processo de contagem de tempo, *controlTime()*, que irá atrelar o funcionamento do processo da rotina de movimentação, *go()*, do Robô à um tempo limite de 60 segundos. Terminados esses 60 segundos o processo será encerrado imediatamente, parando o movimento do Robô.

A rotina de movimentação *go()*, é chamada por meio de um processo como já foi mencionado cujo periodo de execução é dado pela condição de encerramento da função *controlTime()* ou pela completação do trajeto mapeado entre o ponto inicial e o final. A

rotina verifica os passos de movimento do Robô e chama outras rotinas que irão fazer os movimentos angulares esquerda ou direita ou o movimento em linha reta. Cada passo de linha reta possui 30cm sendo que esses passos de movimento são executados por meio de chamadas às rotinas de movimentação controladas *vel_control()*, *angular_dir()* e *angular_esq()*. E assim foi feito o controle deliberativo do Robô. O código fonte é exibido no item 8.

5. A Catapulta

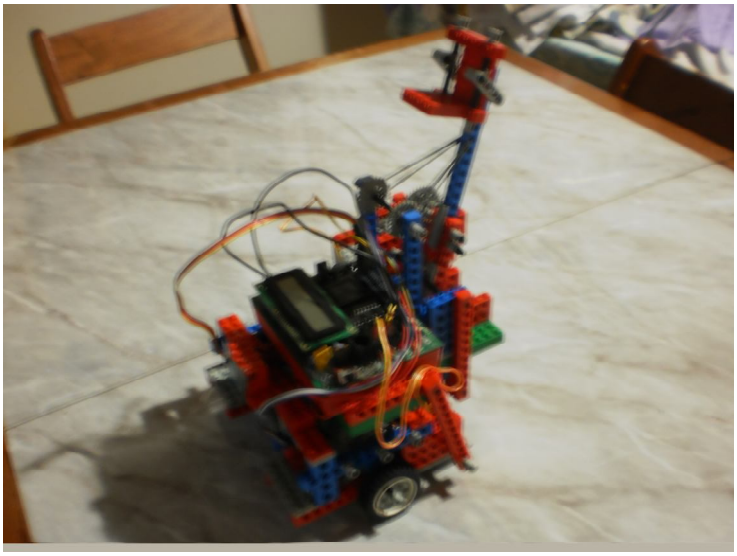

Figura 5.1 – O conjunto Robô + Catapulta

Conforme foi mencionado na introdução, foi requerida a montagem de um mecanismo de disparo de pequenos projeteis (ex: bolinhas de plástico) acoplado ao corpo do Robô com acionamento automático. No presente caso, foi desenvolvida uma catapulta para desempenhar tal função.

Para a estrutura de apoio da Catapulta, uma peça de base foi montada na parte inferior traseira do Robô. Sobre essa peça foi montado o corpo principal da Catapulta, isto é, uma haste com um eixo móvel na parte inferior e um receptáculo para o projétil a ser lançado na parte superior, estando esta mesma haste presa, por meio de um elástico em um ponto superior intermediário, a dois pontos em hastes fixas ligadas entre si por uma viga rígida aqui representada por uma peça de eixo do kit LEGO. A essa viga ficou incumbido, além do papel de ligação das hastes fixas, o papel de representar um batente para o movimento da haste principal de lançamento da Catapulta.

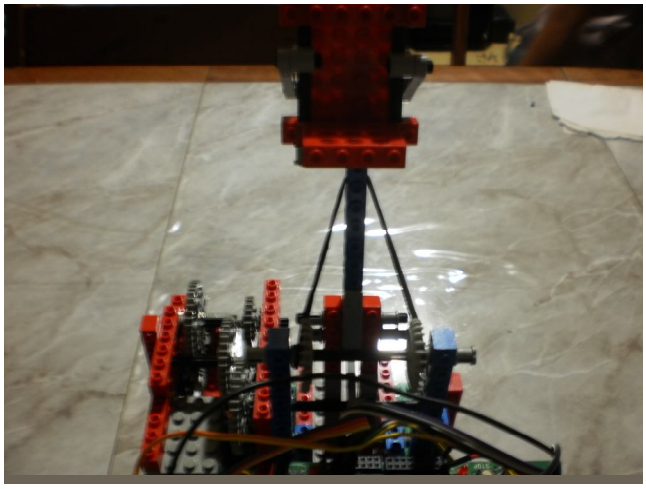

Figura 5.2 – Haste principal ligada por elástico às hastes fixas e receptáculo

Para o acionamento da catapulta, foi necessário utilizar duas peças paralelas para a montagem da parte inferior da haste principal de lançamento deixando um espaço vazio entre essas peças. Nesse espaço vazio, passa um excêntrico, de aproximadamente 3cm montado em um eixo rotativo. A esse excêntrico cabe o papel de forçar a haste principal da Catapulta para trás tocando a peça única superior, superando a força elástica da estrutura montada através do giro do eixo. Quando o excêntrico deixa de tocar a peça única e entra no espaço vazio entre as peças duplas da parte superior da haste, esta volta com aceleração proporcional à força elástica armazenada até tocar o batente entre as hastes fixas e voltar para a posição de equilíbrio. Este movimento brusco é suficiente para lançar um objeto de pequena massa à uma distancia de pouco mais de 60cm.

A automatização deste acionamento feito pelo excêntrico descrito foi feita por meio de um motor (LEGO) que aciona um conjunto de engrenagens de redução transmitindo o movimento até o eixo onde foi montado o excêntrico, com força suficiente para superar a força elástica na haste. Fora utilizadas três reduções de 8 para 40 para multiplicação de força.

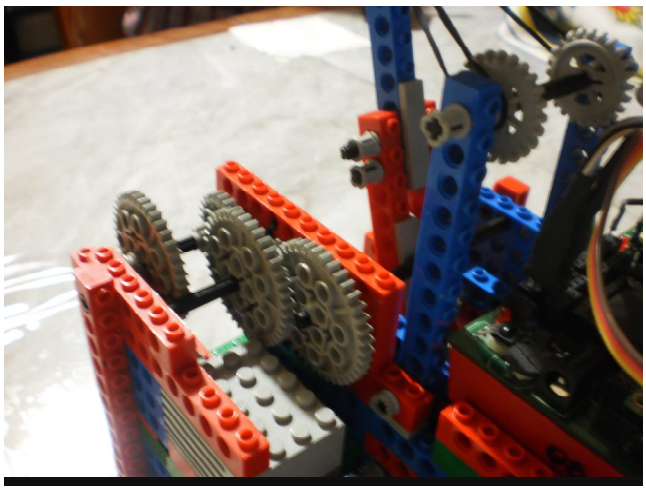

Figura 5.3 – Motor acionando via parafuso sem fim o conjunto de engrenagens de redução para movimentação da catapulta

Toda estrutura foi devidamente contra-ventada por peças de LEGO verticais às paredes da estrutura e barras horizontais interligando as laterais opostas deixando o conjunto bem resistente.

6. Tarefas executadas na apresentação

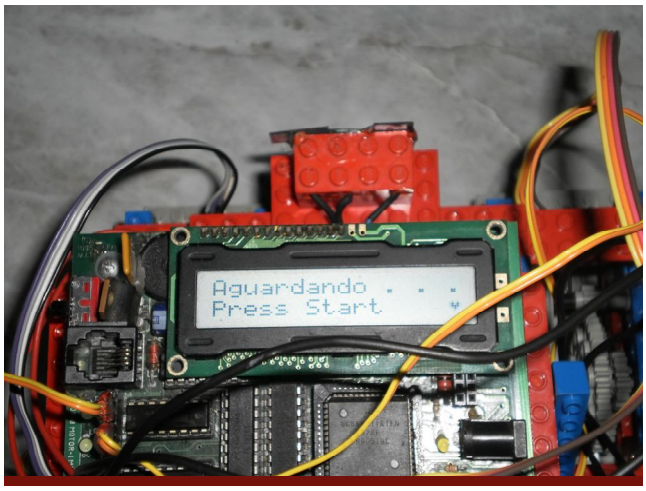

Figura 6.1 – Tela inicial na Handyboard

Na apresentação foi feita primeiramente a movimentação do Robô via programa Wavefront. A pedido do professor Mário, a partir do ponto de início, foi pedido ao que Robô que se deslocasse até um ponto a frente considerando um obstáculo na lateral do campo. O movimento simples foi satisfatoriamente executado após programação dos pontos descritos via teclas start/stop da *Handyboard*. A Figura abaixo exibe a trajetória.

Em seguida foi realizado o procedimento de reconhecimento de luz polarizada de maneira igual a do trabalho prático 2, gravada a luz ambiente, depois a luz a ser seguida, pela diferença o Robô escolhe o melhor ângulo na direção da luz. Depois de identificada a luz a catapulta foi acionada pelo excêntrico movimentado pelo terceiro motor da estrutura lançando o objeto no receptáculo, no caso, uma bola de rolon de

desodorante. Esse teste de identificação de luz e lançamento foi realizado duas vezes para confirmação dos resultados.

7. Conclusão

Após mais um árduo trabalho concluímos:

Sobre a movimentação e o Controle PD finalmente foi obtido um bom resultado principalmente quanto à movimentação linear, dado que a resolução dos encoders foi aumentada por causa da nova estrutura de redução montada no eixo das rodas. Foi identificada uma dificuldade de se adquirir exatidão elevada nos movimentos angulares provavelmente devido a uma pequena folga da nova estrutura das rodas. Em principio essa deficiência não comprometeu a movimentação do Robô.

Sobre o Controle Deliberativo os resultados foram bastante satisfatórios. Foram feitos testes de movimentação com vários mapeamentos diferentes do campo e em todos o Robô realizou a trajetória esperada chegando sempre com pelo menos 90% do corpo do Robô posicionado sobre o bloco marcado como ponto de chegada.

Sobre a identificação da luz polarizada pode-se de dizer que o novo controle angular melhorou um pouco a busca pela luz, mas é um ponto que ainda pode ser melhorado.

Sobre a Catapulta pode-se dizer que ficou bem montada e resistente às pancadas dos lançamentos. Sobre a força elástica armazenada ainda é possível melhorar com mais elásticos ou aumentando o comprimento do excêntrico, de forma a adquirir um disparo com maior longevidade. Entretanto, para isso, talvez seja necessário aumentar a força de acionamento com mais redução no jogo de engrenagens. Outra melhoria que deverá ser feita na catapulta refere-se ao encurtamento do giro do excêntrico até ocorrer o disparo. Atualmente esse giro dura cerca de 18 segundos entretanto a parte útil, onde a energia esta sendo armazenada leva apenas 5 segundos.

Sobre o trabalho de maneira geral pode-se dizer que dentre houve grande dificuldade na montagem das estruturas tanto do Robô quanto da Catapulta de forma que todo o conjunto ficasse bastante rígido. Não foi difícil pensar na montagem das engrenagens, mas a atividade foi bastante dispendiosa em termos de tempo. Na parte de sensores não houve grande dificuldade de implementação e montagem. Na parte de programação a dificuldade maior foi a de ajustar os parâmetros do controle PD. Durante os ajustes foi implementado até um controle PID, mas por fim acabou que os valores dos parâmetros ficaram próximos aos valores obtidos no trabalho prático anterior.

8. Código-fonte

TP3.c

```
#include wavefront.c
#include myBiblioteca.ic
#include catapulta.ic

int pidTime;
int pidWave;

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] = {"Wavefront", "Catapulta", "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 Wavefront\n");
                waveConfig();
                waitingLightStart();
                pidWave = start_process(go());
            }
            else if(opcao == 1){
                message("Selecionado Catapulta\n");
                catapulta();
            }
            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 waitingLightStart(){

    while(digital(15)!= 1);
    tempo0 = seconds();
    pidTime = start_process(controlTime());
    esquerda();
    vel_control(156);
    direita();
```

```
}

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

constantes.c

```
//#ifndef CONSTANTES
//#define CONSTANTES


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


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

int motor2 = 0;
int motor1 = 1;

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.ic

```
#include constantes.c
```

```
//-----------------------------------------------------------//
/*  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.ic

```
#include sencdr0.icb
#include sencdr1.icb
/*
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;

    float erro0anterior = -999.0;
    //float erro1anterior = 0.0;
    float erro0 = 0.0;
    //float erro1 = 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;
int power0;
int power1;
int vel0;
int vel1;
float Kp0 = 5.0;
float Kd0 = 0.1;
float Kp1 = 0.5;
float Kd1 = 0.1;

float Kp = 0.16;
float Kd = 0.01;
float Ki = 0.001;
if(power0 == 0 && power1 == 0);{
    power0 = 90;
    power1 = 75;
}
vel0 = 50;
vel1 = 50;
//printf("%d e %d\n",power0,power1);

//reset_system_time();
encoder0_counts = 0;
encoder1_counts = 0;
//while(seconds() < tempo){
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);
        //In case of error too small then stop intergration
        /*if(mod(erro)>2.0){
        integral = integral + erro*timedif;
        }
        */
        derivativo = (erro - erroanterior)/timedif;

        /*
        errodifanterior = errodif;
        errodif = (float)(vel_atual0 - vel_atual1) - (float)(vel0 -
vel1);
        derivativodif = (errodif - errodifanterior)/timedif;
        */

        //if(mod(erro)>1.0){
        //power1 -= (int)(erro * Kp + derivativo * Kd);
        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++;
        }*/

    }
}

void vel_control_ang(int counts) {

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

    float erro0anterior = -999.0;
    //float erro1anterior = 0.0;
    float erro0 = 0.0;
```

```
//float erro1 = 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;
int power0;
int power1;
int vel0;
int vel1;
float Kp0 = 5.0;
float Kd0 = 0.1;
float Kp1 = 0.5;
float Kd1 = 0.1;

float Kp = 0.01;
float Kd = 0.001;
float Ki = 0.001;
if(power0 == 0 && power1 == 0 && counts !=0);{
    power0 = -80 * (int)((float)counts/mod((float)counts));
    power1 = 80 * (int)((float)counts/mod((float)counts));
}

counts = (int)(mod((float)counts));
vel0 = 50;
vel1 = 50;
//printf("%d e %d\n",power0,power1);

//reset_system_time();
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);
        //In case of error too small then stop intergration
        /*if(mod(erro)>2.0){
        integral = integral + erro*timedif;
        }
        */
        derivativo = (erro - erroanterior)/timedif;

        /*
        errodifanterior = errodif;
        errodif = (float)(vel_atual0 - vel_atual1) - (float)(vel0 -
vel1);
        derivativodif = (errodif - errodifanterior)/timedif;
        */

        //if(mod(erro)>1.0){
        //power1 -= (int)(erro * Kp + derivativo * Kd);
        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;
    //counts = (int)(618.0 * ((ang-(180.0/ang))/360.0));
    if(ang != 0.0){
        if(ang == 360.0)
          {counts = 590;}
        else if(ang == 180.0)
          {counts = 290;}
        else if(ang == 90.0)
          {counts = 135;}
        else if(ang == 45.0)
          {counts = 56;}
        else if(ang == 135.0)
          {counts = 3*55;}
        else if(ang == 225.0)
          {counts = 5*55;}
        else if(ang == 270.0)
          {counts = 6*55;}
        else if(ang == 315.0)
          {counts = 7*55;}
        vel_control_ang(-counts);
        ao();
        sleep(0.2);
    }
    //motor(0,80);
    //motor(1,-90);

    printf("%d e %d\n",encoder0_counts,encoder1_counts);
    /*
    while(mot0 == 1 || mot1 == 1) {
        if(encoder0_counts >= counts)
          {off(0);
            mot0 = 0;}
        if(encoder1_counts >= counts)
          {off(1);
            mot1 = 0;}
    }
    */
}

void angular_esq(float ang){

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

```
    //counts = (int)(618.0 * ((ang-(180.0/ang))/360.0));
    if(ang == 360.0)
      {counts = 595;}
    else if(ang == 180.0)
      {counts = 290;}
    else if(ang == 90.0)
      {counts = 135;}
    vel_control_ang(counts);
    ao();
    sleep(1.0);
    //motor(0,80);
    //motor(1,-90);
    printf("%d e %d\n",encoder0_counts,encoder1_counts);
    /*
    while(mot0 == 1 || mot1 == 1) {
        if(encoder0_counts >= counts)
          {off(0);
            mot0 = 0;}
        if(encoder1_counts >= counts)
          {off(1);
            mot1 = 0;}
    }
    */
}

/*
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(333);
    ao();
}

/*
void main(){
    //Testes de movimentacao
    printf("Aguardando . . . \n");
    sleep(2.0);
    beep();
    while(!start_button());
```

```
    emFrente();
    esquerda();
    emFrente();
    direita();
    emFrente();


    vel_control(312);
    vel_control(312);
    ao();
    angular_esq(180.0);
    vel_control(312);
    vel_control(312);
    ao();


    angular_esq(360.0);
    sleep(5.0);
    angular_esq(180.0);
    sleep(5.0);
    angular_esq(90.0);
    sleep(5.0);

}
*/
```

wavefront.c

```
#include Controle.ic
#define ESQUERDA 1
#define FRENTE 0
#define DIREITA -1

#define LESTE 0
#define NORTE 90
#define OESTE 180
#define SUL 270

#define X_DIM 10
#define Y_DIM 10

#define MAX 255

//Controlam o tempo da partida < 60s.
float tempo0;
float tempo1;

/* Mapa contendo os obstaculos do ambiente */
int map[X_DIM][Y_DIM];

/* Pose = posicao (x y) + orientacao */
int pose_atual[3];    // posicao (x,y) atual + orientacao (N / S / L /
O)
int pose_inicial[3];  // posicao inicial
int pose_desejada[3]; // posicao final desejada (ignorar a orientacao)

/* Variaveis extras*/
int fifo[100][2];
int indice = 0;
int indice2 = 0;
```

```
//void motor(int m, int p) {}
//void ao() {}
//void msleep(long time) {}

/* Esta funcao deve fazer o seu robo andar 1 unidade de distancia para
 * frente e tambem deve atualizar a variavel pose_atual, de acordo
 * com a orientacao atual
 */
void goAhead() {
    if (pose_atual[2] == LESTE)
      pose_atual[0]++;
    if (pose_atual[2] == NORTE)
      pose_atual[1]++;
    if (pose_atual[2] == OESTE)
      pose_atual[0]--;
    if (pose_atual[2] == SUL)
      pose_atual[1]--;

    emFrente();
    /*
    motor(0,50);
    motor(1,50);
    msleep(1000L);
    ao();
 */
}

/* Esta funcao deve fazer o seu robo girar 90 graus para a esquerda e
 * tambem deve atualizar a variavel pose_atual
 */
void turnLeft() {
    pose_atual[2] += 90;
    if (pose_atual[2] == 360)
      pose_atual[2]= 0;

    /*
 motor(0,-30);
    motor(1,+30);
    msleep(2500L);
    ao();
 */
    esquerda();
}

/* Esta funcao deve fazer o seu robo girar 90 graus para a direita e
 * tambem deve atualizar a varivael pose_atual
 */
void turnRight() {
    pose_atual[2] -= 90;
    if (pose_atual[2] == -90)
      pose_atual[2]= 270;

    /*
    motor(0,+30);
    motor(1,-30);
    msleep(2500L);
    ao();
 */
```

```
    direita();
}

void push(int x, int y, int valor) {
    if (x >= 0 && x < X_DIM && y >=0 && y < Y_DIM && map[x][y] == 0) {
        //printf("fifo[%d] <= [%d %d]\n", indice, x, y);
        map[x][y] = valor;
        fifo[indice][0] = x;
        fifo[indice][1] = y;
        indice++;
    }
}

void pop(int *x, int *y) {
    *x = fifo[indice2][0];
    *y = fifo[indice2][1];
    indice2++;
    //printf("fifo[%d] => [%d %d]\n", indice2, *x, *y);
}


/* Esta funcao deve calcular o mapa de distancias usando o algoritmo
 * wavefront.
 * A variavel map deve conter as distancias entre o ponto final e
inicial
 * ao final da sua execucao
 */
void calculaWaveFront() {
    int x,y;
    int i,j;
    int stop = 0;

    for (i = 0; i < X_DIM; i++) {
        map[i][0] = MAX;
        map[i][Y_DIM - 1] = MAX;
    }
    for (j = 0; j < Y_DIM; j++) {
        map[0][j] = MAX;
        map[X_DIM- 1][j] = MAX;
    }

    push(pose_desejada[0], pose_desejada[1], 1);
    x = pose_desejada[0], y = pose_desejada[1];
    while (!stop) {
        pop(&x , &y);
        push(x, y - 1, map[x][y] + 1);
        push(x, y + 1, map[x][y] + 1);
        push(x + 1, y, map[x][y] + 1);
        push(x - 1, y, map[x][y] + 1);
        if (x == pose_inicial[0] && y == pose_inicial[1])
          stop = 1;
    }

    for (j = 0; j < Y_DIM; j++)
      for (i = 0; i < X_DIM; i++)
        if (map[i][j] == 0) map[i][j] = MAX;
}
```

```
/* Esta funcao deve retornar a direcao que o robo deve seguir para
sair
 * do quadrante atual para chegar ate o proximo quadrante, obedecendo
 * o mapa criado pelo algoritmo wavefront
 */
int calculaDirecao() {
    int x, y, xa, ya, xl, yl, xr, yr;

    x = pose_atual[0];
    y = pose_atual[1];
    xa = x;    xl = x;
    xr = x;    ya = y;
    yl = y;    yr = y;

    if (pose_atual[2] == LESTE) {
        xa = x + 1;
        yl = y + 1;
        yr = y - 1;
    }
    if (pose_atual[2] == NORTE) {
        ya = y + 1;
        xl = x - 1;
        xr = x + 1;
    }
    if (pose_atual[2] == OESTE) {
        xa = x - 1;
        yl = y - 1;
        yr = y + 1;
    }
    if (pose_atual[2] == SUL) {
        ya = y - 1;
        xl = x + 1;
        xr = x - 1;
    }
    if (map[xa][ya] < map[x][y])
      return FRENTE;
    if (map[xl][yl] < map[x][y])
      return ESQUERDA;
    if (map[xr][yr] < map[x][y])
      return DIREITA;
}

/* Imprime o mapa na tela. util para realizar a depuracao (so funciona
no PC).
 */
void printMap() {
    int i, j;
    for (j = 0; j < Y_DIM; j++) {
        for (i = 0; i < X_DIM; i++) {
            printf("%3d ", map[i][j]);
        }
        printf("\n");
    }
}

/* Le um valor do knob variando de 0 at intervalo*/
int getKnob(char texto[], int intervalo) {
    int valor;
    while (!start_button()) {
```

```
        valor = (intervalo * knob()) / 255;
        printf("%s %d\n", texto, valor);
        msleep(100L);
    }
    while (start_button());
    return valor;
}

void escolhePosicao() {
    int posicao;
    pose_inicial[0] = getKnob("X inicial: ", X_DIM);
    pose_inicial[1] = getKnob("Y inicial: ", Y_DIM);

    pose_atual[0] = pose_inicial[0];
    pose_atual[1] = pose_inicial[1];

    pose_desejada[0] = getKnob("X desejado: ",X_DIM);
    pose_desejada[1] = getKnob("Y desejado: ",Y_DIM);
}

void escolheMapa() {
    int x, y;
    int n, i;
    n = getKnob("Numero de obstaculos: ", 20);
    for (i = 0; i < n; i++) {
        x = getKnob("X obstaculo: ",X_DIM);
        y = getKnob("Y obstaculo: ",Y_DIM);
        map[x][y] = MAX;
    }

    //map[x][y] = MAX;
map[5][1] = MAX;
map[5][2] = MAX;
map[5][3] = MAX;
map[5][4] = MAX;
map[5][5] = MAX;
//map[x][y] = MAX;
map[1][5] = MAX;
map[2][5] = MAX;
map[3][5] = MAX;
map[4][5] = MAX;
map[5][5] = MAX;

}

void waveConfig() {//antigo main()

    printf("Wavefront       Press Start!\n");

    beep();
    while(!start_button());


    msleep(1000L);

    escolhePosicao();
    escolheMapa();

    printf("Comecando em [%d %d], alvo: [%d %d]\n",
```

```
            pose_inicial[0], pose_inicial[1],
            pose_desejada[0], pose_desejada[1]);

    calculaWaveFront();
    //printMap();
    beep();
}

void go(){ //Segunda parte do wave config
    int direcao;
    int stop = 0;
    beep();
    sleep(1.0);
    while (!stop) {
        direcao = calculaDirecao();
        if (direcao == ESQUERDA)
          turnLeft();
        if (direcao == DIREITA)
          turnRight();
        goAhead();

        printf("Estou em [%d %d]\n", pose_atual[0], pose_atual[1]);

        /* Verifica se ja chegou no alvo*/
        if (pose_atual[0] == pose_desejada[0] &&
            pose_atual[1] == pose_desejada[1])
          stop = 1;
    }
    //return 0;
}
```

catapulta.c

```
#include myBiblioteca.ic
#include Controle.ic

int luzAmbiente = 0;
int goal;

void catapulta() {
    int pidMostra = 0;
    int button = -1;
    luzAmbiente = 0;
    goal = 0;
    message("Funcao de polarizacao\n");

    message("Selecionar luz ambiente\n");
    pidMostra =
start_process(mostraContinuamenteAnalogico(sensorPolo));
    luzAmbiente = setAnalogValue(sensorPolo);
    printf("%d ", luzAmbiente);
    kill_process(pidMostra);
    message(" = luz ambiente\n");

    message("Selecionar a luz a seguir\n");
    pidMostra =
start_process(mostraContinuamenteAnalogico(sensorPolo));

    goal = setAnalogValue(sensorPolo);
```

```
    printf("%d ", goal - luzAmbiente);

    kill_process(pidMostra);
    message(" = luz selecionada\n");

    message("Start - search  Stop - Menu\n");
    sleep(0.2);
    button = waitingButtonPress();
    if (button == START) {
        busca();
    }
}

void busca() {
    int valor[8];
    int value = 0;
    int v0 = 0;
    int v1 = 1;

    int selectedAngle = 0;
    int angle = 0;
    int pidLinear = 0;
    int dif = 99999;
    int i = 0;

    beep();

    for (i = 0; i < 8; i++) {
        angle = i * 45;
        sleep(0.1);
        valor[i] = analog(sensorPolo);
        printf("Angulo = %d Now-%d\n", angle, valor[i] - goal);
        sleep(0.1);
        angular_dir(45.0);
        ao();
    }

    for (i = 0; i < 8; i++) {

        //if (goal > 0) { //Busca a maior diferenca positiva
        if (dif < valor[i] - goal) {
            dif = valor[i] - goal;
            selectedAngle = i * 45;
        }
        /* } else if (goal <= 0) {//Busca a maior diferenca negativa
            if (dif <   goal - valor[i]) {
                dif =  goal - valor[i];
                selectedAngle = i * 45;
            }
        */
        //}

        printf("Angulo = %d dif(%d) Now(%d)\n", angle, dif, valor[i] -
goal);

    }

    //angular_dir(45.0);
    //ao();
```

```
    //angular_dir(45.0);
    //ao();
    //sleep(0.5);
    if(goal - luzAmbiente < 0){

        selectedAngle = selectedAngle+180;
        if (selectedAngle >=360){
            selectedAngle = selectedAngle - 360;
        }
    }
    printf("Angulo a seguir : %d\n", selectedAngle);
    if(selectedAngle != 0 ){
        for(i=0;i<(selectedAngle/45);i++){
            //angular_dir((float) selectedAngle);
            angular_dir(45.0);
        }
    }
    /* sleep(0.3);
    ao();
    pidLinear = start_process(vel_control(3.0,3.0));
    sleep(2.0);
    kill_process(pidLinear); ao();
     */
    ao();
    motor(2,90);
    sleep(18.0);
    ao();

}
/*
void main(){
    polarizacao();
}*/
```

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