Trabalho prático 2 – Percepção: Sensores

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

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

alissonrs@ufmg.br, martinsrmateus@gmail.com

1. Introdução

Na proposta do segundo foi requerido como atividades do robô móvel, ser capaz de seguir em campo, uma de duas fontes de luz polarizadas inversamente de acordo com uma pré-seleção. Ser capaz de seguir também no mesmo campo, o traçado marcado com linha preta em meio ao fundo branco e ser capaz de identificar blocos de isopor no campo colocados no caminho do robô bem como a cor desses blocos para realizar movimentos de acordo com a cor desses blocos. Foi possível realizar tais atividades através de sensores e LEDs montados no corpo do robô sendo essa montagem e calibração de sensores a tônica das atividades realizadas neste trabalho prático.

2. Melhorias no robô

Paralelo a atividade de construção dos sensores foi feito um ajuste de programação no esquema de controle de movimentação do Robô, bem como a substituição da biblioteca padrão do IC pela biblioteca customizada por Julian Skidmore para obter maior resolução na gradação de potência do motor.

O controle de movimentação foi implementado de forma a manter uma determinada velocidade de *set-point* para cada motor sendo que o ajuste de potencia do motor 1 (à direita) foi feito baseado no erro entre a velocidade ajustada e a atual de forma bem suave e o ajuste de potencia do motor 0 (à esquerda) foi feito baseado no erro entre a diferença de velocidades ajustadas para 0 e 1 e a diferença atual com ganhos maiores em relação ao controle do motor 1. Inicialmente as potencias dos dois motores é levada a um valor que seja mais próximo do valor desejado de acordo com uma tabela de valores pré-calculados que relaciona velocidade com a potência para diferentes valores.

Ou seja, o controle é feito da seguinte maneira, após escolhidos os valores de velocidade passados como parâmetro, a potência de ambos os motores é ajustada para valores próximos aos correspondentes a essas velocidades. A velocidade de cada roda é então calculada em passos de 0.5 em 0.5 segundos com base na informação dos *encoders* e essa informação é usada para fazer o controle de alinhamento ajustando a potencia do motor 0 através de ganhos proporcional e derivativo e alem disso também é feito um controle direto de velocidade da roda acionada pelo motor 1 ajustando a potencia através de ganhos proporcional e derivativo bem menores que no controle do motor 0.

Dessa forma garantiu-se o controle de movimentação linear do Robô a ser utilizado em conjunto com o "cherador de linha".

3. Construção de sensores ópticos ativos

Canhão de cor – Sensor ótico construído com um LED vermelho, um LED azul, dois LED' verdes, um LDR, resistores e transistores NPN 2N2222 para cada LED.

O acionamento dos leds são feitos por meio dos comandos:

bit_set(0x1009, 0b00000100) - Define a porta D2 como saída.

bit_set(0x1008, 0b00000100) - Seta a porta D2 para on.

bit_clear(0x1008, 0b00000100) - Seta a porta D2 para off.

00000100 - Led Verde

00001000 - Led Azul

00010000 - Led Vermelho

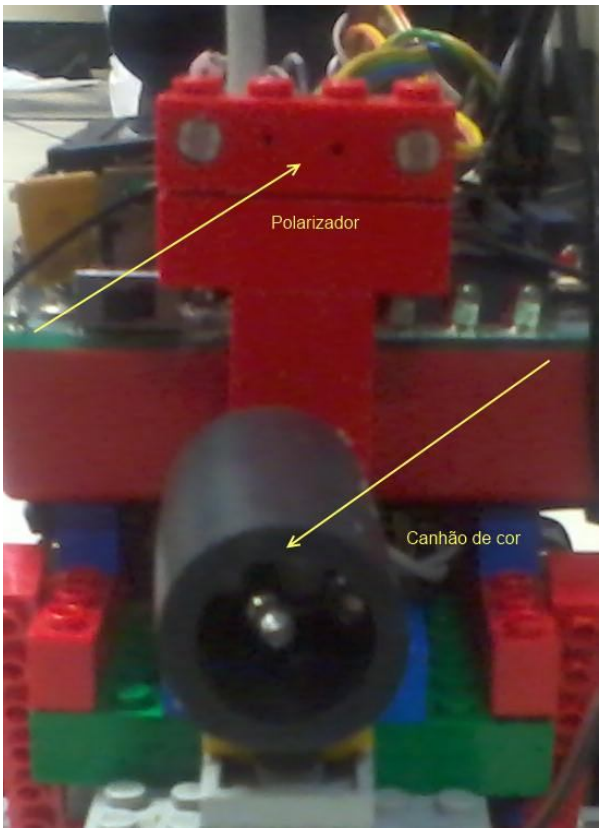

Figura 1. Canhão de cor e do Caolho

Esquema elétrico do Canhão de cor:

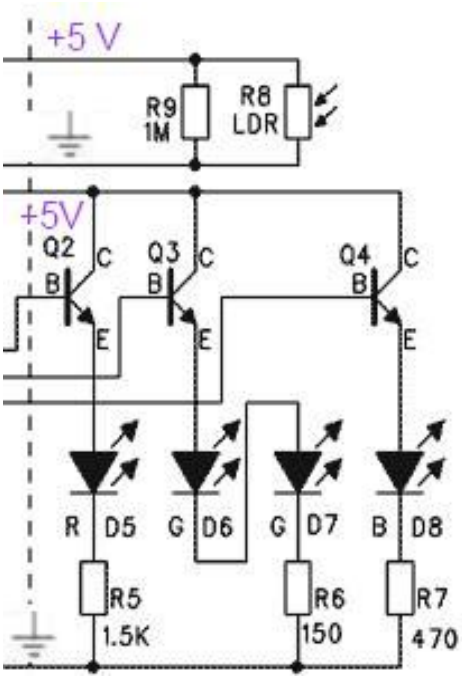

Figura 2. Esquema elétrico do canhão de cor

Caolho – Como sensor de distância utilizou-se um LED emissor de luz na faixa do visível (led branco) e um foto-diodo. A este sensor demos o nome de Caolho por sua aparência, conforme pode ser visto na figura 3 abaixo do canhão de cor.

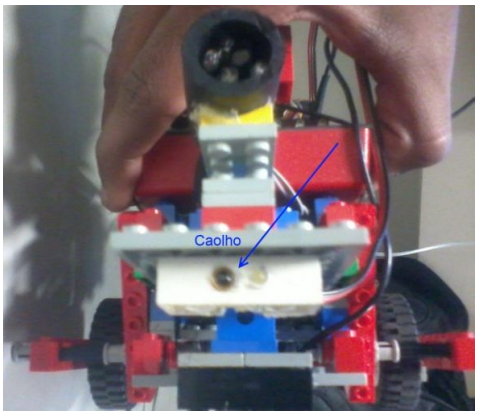

Figura 3. Sensor de distância (Caolho)

O sensor foi montado com um esquema elétrico simples utilizando um LED com um resistor de 150 ohms e um foto-diodo.

Sensores Diferenciais – Foram construídos dois sensores diferencias. Um para avaliar a polarização da luz no ambiente em que o robô se encontra, e por isso colocado na parte superior dianteira do robô como pode ser visto na figura 1, e um para que o robô pudesse seguir uma fita preta fixada na superfície branca onde o robô irá se

movimentar. Este sensor (Cherador de linha) foi posicionada na parte da frente do robô bem próximo ao chão, conforme pode ser visto na figura 7.

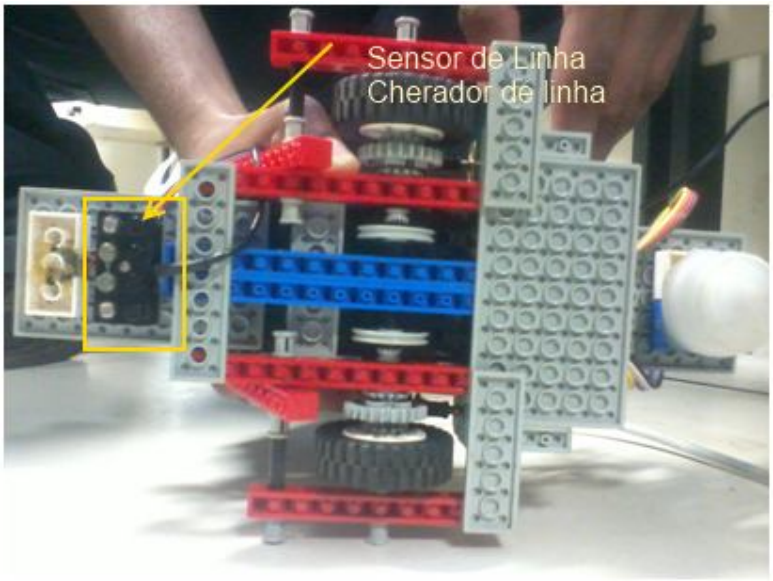

Figura 7. Sensor para acompanhar linha (Cherador de linha) montado sobre a base do robô, um pouco abaixo do sensor de presença de objeto (Caolho)..

Ambos os sensores utilizam um circuito diferencial simples conforme descrito na literatura e sua diferença básica é que o polarizador é constituído de apenas dois foto-receptores LDR's enquanto que o seguidor de linha possui também um LED e um resistor.

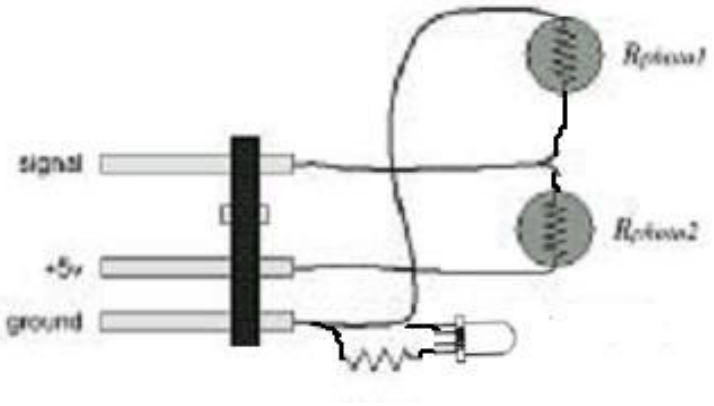

Figura 8. Esquema elétrico do sensor para acompanhar linha. A diferença para o sensor de polarização é que este não possui o led e o resistor e, na frente de cada receptor é colado um polarizador do tipo polaroide.

4. Caracterização dos sensores

Caolho – O resultado da coleta de dados de um objeto colocado a 3mm aproximadamente do sensor é apresentado abaixo.

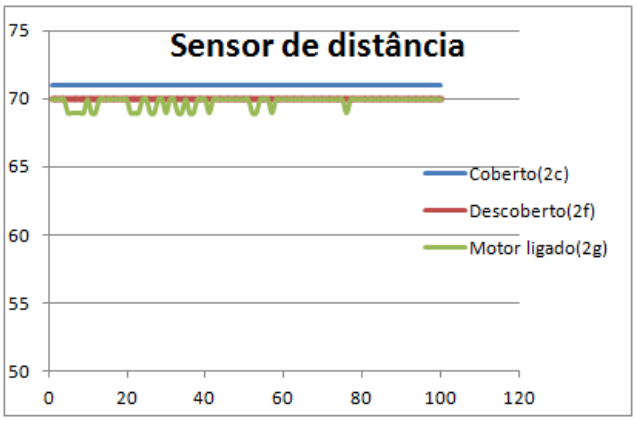

Figura 4. Dados do sensor de distância (Caolho)
Coberto: Média 71, Desvio padrão 0
Descoberto: Média 70, Desvio padrão 0
Motor ligado: Média 69, Desvio padrão: 0,42

Conforme pudemos observar não houve variações significativas entre as tomadas de dados. Um dos motivos pode ser o fato de que o sensor já estava montado no robô e o sensor permaneceu a uma distância de 1cm aproximadamente, já com algum isolamento da luz externa. Outro fator é que o LED receptor de luz possui um tipo de proteção na frente que funciona como uma lente que só recebe luz direta, o que elimina efeitos da luz do ambiente externo.

Porém não ficou claro o efeito do ligar os motores, já que o anteparo deveria estar a uma distância fixa. Acreditamos que o efeito poderia ser de variações do circuito interno de controle, mas não observamos nenhuma variação considerável.

Canhão de Cor – Para a caracterização deste sensor foram coletados dados de medida para diversas condições. A contemplação de várias condições diferentes consistiu em, variar a cor dos blocos posicionados a frente do sensor (Amarelo, Azul, Verde e Vermelho), variar a iluminação ambiente (muita luz, luz média, pouca luz e escuro total com o robô coberto por uma caixa) e variar o acendimento dos LEDs do canhão (sem LED, LED azul aceso e LED vermelho aceso). Os testes não foram feitos com os LEDs verdes acesos, pois não apresentaram respostas interessantes para a caracterização devido a baixa intensidade dos LEDs. As medições foram realizadas com os blocos posicionados a uma distância de aproximadamente 5mm uma vez que antes, foi calibrada a função de parada quando avistado um obstáculo e para vários testes o robô parou a uma distância aproximadamente igual a essa dos blocos. Outro ponto a se destacar é que para essa calibragem de reconhecimento de cor foram utilizados os dados coletados do sensor ótico de distância Caolho, já mencionado. Diante dessas considerações e das medidas feitas, construiu-se a tabela de valores registrados pelos sensores exibida abaixo:

BLOCO	SENSOR	Condição	LED	Valor Medio
Amarelo	Cor	Muita LUZ	Sem LED	225,8
			Azul	182,3
			Vermelho	176,9
		Pouca LUZ	Sem LED	232
			Azul	186
			Vermelho	180
		Penumbra	Sem LED	252,1
			Azul	209
			Vermelho	200
	Caolho	Muita LUZ	Sem LED	12,4
			Azul	13
			Vermelho	13
		Pouca LUZ	Sem LED	13
			Azul	13
			Vermelho	13
		Penumbra	Sem LED	39
			Azul	39
			Vermelho	39
Azul	Cor	Muita LUZ	Sem LED	243
			Azul	144,3
			Vermelho	240
		Pouca LUZ	Sem LED	244
			Azul	145
			Vermelho	241
		Penumbra	Sem LED	253,9
			Azul	158
			Vermelho	251,6
	Caolho	Muita LUZ	Sem LED	126
			Azul	126,1
			Vermelho	125,9
		Pouca LUZ	Sem LED	141
			Azul	140,1
			Vermelho	140
		Penumbra	Sem LED	212
			Azul	212
			Vermelho	212

BLOCO	SENSOR	Condição	LED	Valor Medio
Verde	Cor	Muita LUZ	Sem LED	232,6
			Azul	191
			Vermelho	225,3
		Pouca LUZ	Sem LED	240
			Azul	200
			Vermelho	234
		Penumbra	Sem LED	253,3
			Azul	214
			Vermelho	248
	Caolho	Muita LUZ	Sem LED	114,5
			Azul	114,4
			Vermelho	114,2
		Pouca LUZ	Sem LED	142
			Azul	142
			Vermelho	142
		Penumbra	Sem LED	185
			Azul	185
			Vermelho	185
Vermelho	Cor	Muita LUZ	Sem LED	228,7
			Azul	203
			Vermelho	187,3
		Pouca LUZ	Sem LED	228
			Azul	203
			Vermelho	187
		Penumbra	Sem LED	253
			Azul	229
			Vermelho	214
	Caolho	Muita LUZ	Sem LED	228,7
			Azul	203
			Vermelho	187,3
		Pouca LUZ	Sem LED	228
			Azul	203
			Vermelho	187
		Penumbra	Sem LED	253
			Azul	229
			Vermelho	214

A partir da tabela foi feito um cruzamento de dados a fim de se obter características únicas de cada cor para identificação em qualquer condição de iluminação. O resultado desse cruzamento de dados foi implementado diretamente no código.

Sensores Diferenciais – A coleta de dados dos dois sensores diferenciais construídos (o Polarizador e o "Cherador de Linha" foi feita mediante o programa da figura 9 abaixo que apresenta continuamente no display da Handyboard o valor coletado pelo sensor conectado na porta x.

```
void programa(){
    while(1){
    int x = 4; //Porta onde os sensor está conectado.
    printf("Sensor: %d    Sensor: %d\n",x, analog(x));
    sleep(0.5);
    }
}
```

Figura 9. Programa que lê continuamente o sinal coletado pelo sensor conectado na porta x.

O sensor de linha apresentou os seguintes resultados para medidas em um padrão claro-escuro:

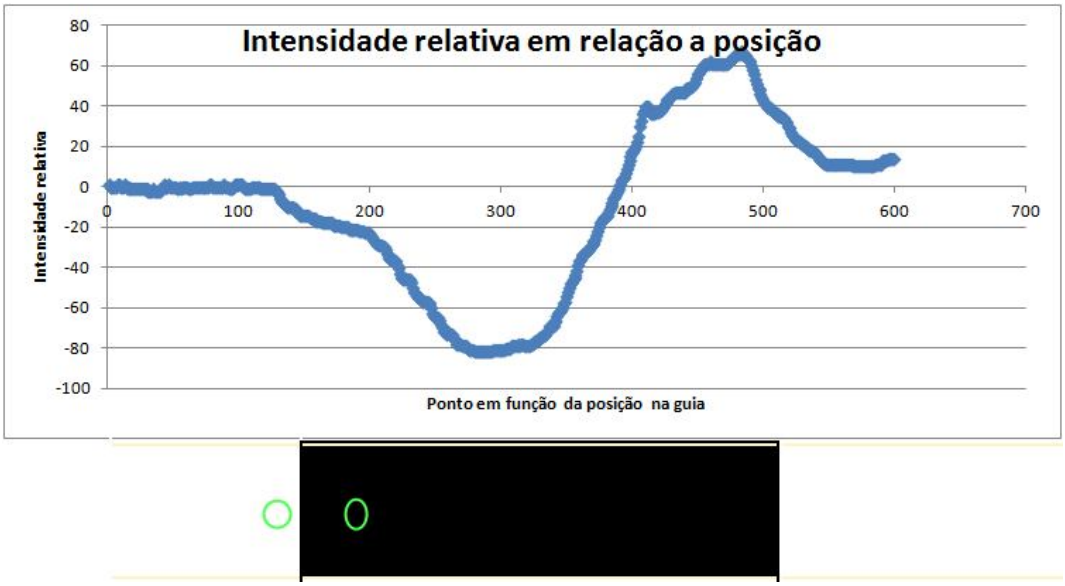

Figura 10. Padrão de intensidade em função da posição sobre a faixa escura no chão obtido com o sensor de seguir linha (Cherador de linha). Os dados estão transladados para o zero de modo que o valor de 129 (posição em que os dois sensores recebem luz igual) foi considerado o ponto zero de medida.

Conforme a figura 10 é possível observar que quando os dois sensores coletam a mesma quantidade de luz sua intensidade permanece próximo ao zero. Entre mais ou menos 20 temos uma faixa intermediária em que consideramos ainda os dois sensores com mesma intensidade, e acima ou abaixo destes valores temos um sensor recebendo mais luz que os demais.

Os testes com o polarizador mostram uma variação considerável quando da mudança da polarização da luz incidente sobre o sensor. Um gráfico que ilustra esta variação pode ser visto na figura 11.

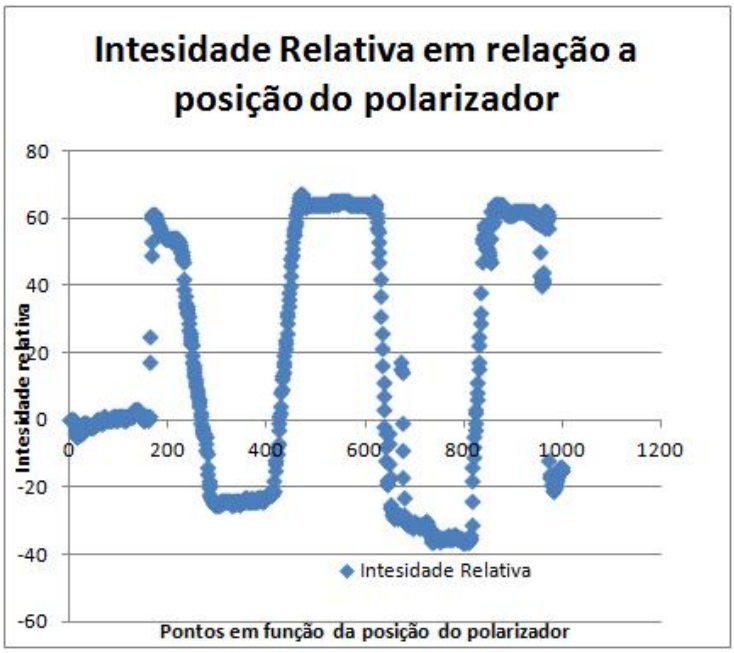

Figura 11. Padrão de intensidade em função da posição do polarizador obtido com o sensor de Polarização. Os dados estão transladados para o zero de modo que o valor de 153 (posição em que os dois sensores recebem luz sem polarização alguma) foi considerado o ponto zero de medida.

5. Tarefas executadas na apresentação

A apresentação teve que ser realizada em um dos corredores da Universidade por indisponibilidade da sala do laboratório de fato isso acabou por afetar o desempenho do robô nas atividades de reconhecimento de cores e luz já que a luz do sol estava muito intensa no ambiente. Outro fator que prejudicou um pouco na execução das tarefas de movimentação foi o fato de a bateria não estar a plena carga, esta condição afeta consideravelmente as relações potencia/torque e potencia/velocidade do motor.

O robô conseguiu encontrar as luzes polarizadas corretamente de acordo com a seleção do menu. Uma vez encontradas o robô tendia a seguir uma linha reta na direção calculada pelo sensor.

O reconhecimento de cor só foi possível realizar com uma pequena sombra nos arredores do campo de forma a atenuar a forte intensidade da luz solar. De acordo com a calibração feita o robô só conseguia enxergar a cor amarela apesar de parar corretamente em frente a cada um dos blocos. Fora do campo com uma intensidade de luz ambiente um pouco menor o reconhecimento de cor foi feito com sucesso.

Na apresentação foi executada apenas a movimentação de contorno à direita (bloco azul reconhecido) apesar de que as outras movimentações foram testadas com sucesso previamente. O contorno foi implementado utilizando giros de 90º e movimentos em linha reta.

O seguimento de linha foi executado com sucesso inclusive na condição em que o robô começou fora da linha sendo necessário fazer o reconhecimento da linha para entrada no caminho certo.

6. Conclusão

Após este árduo trabalho concluímos:

- Sobre a movimentação e o controle PD após testar vários algoritmos e por fim aceitarmos um que ainda assim não controla da forma desejada chegamos a conclusão que a resolução dos encoders está muito baixa e que será necessário resolver este problema para o próximo trabalho. A roda de contagem está ligada diretamente ao eixo da roda de contato com o solo o que fornece apenas 12 counts por volta. Esse valor é muito baixo para que o controle PD possa dar uma resposta rápida. Será necessária, portanto a inclusão de uma redução próxima ao eixo de tração mudando a posição dos contadores para obter melhor resolução. Só assim o controle poderá responder satisfatoriamente.

- Sobre o reconhecimento de cores faltou levar em conta nas calibrações feitas uma situação de iluminação extrema, no caso, fala-se da luz solar já que a condição de muita luz utilizada na caracterização considerou uma iluminação feita por lâmpadas de luz branca apenas. Dessa forma o reconhecimento de cor amarela deveria ser reconfigurado para a situação de robô movimentando sob forte luz solar.

- Sobre o trabalho de uma forma geral incluindo a construção dos sensores, caracterização dos mesmos e programação dos códigos, chegamos a conclusão que a organização, divisão de tarefas e aproveitamento do tempo, melhorou bastante em relação ao primeiro trabalho prático, mas ainda é preciso fazer melhor para evitar que atividades que aparentemente parecem ser fáceis de se concluir possam se tornar grandes dores de cabeça no desenvolvimento do trabalho como um todo.

7. Código Fonte

TP2.ic

```
#include Controle_Movimento.ic
#include constantes.c
#include polarizacao.ic
#include segueLinha.ic
#include segueBloco.ic
#include myBiblioteca.ic



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

void main(){

    message("Press Start     to Menu\n");

    start_press();

    menu();
```

```
    message("FIM\n");

}

//-----------------------------------------------------------//
/* main()
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 = 4;
    char strOpcao[4][16] = {"Polarizacao","Segue Linha","Segue
Blocos", "Sair"};


    while( sair == FALSE){

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

        button = waitingButtonPress();
        if (button == START){
            if(opcao == 0){
                //message("Selecionado polarizacao\n");
                polarizacao();
            }
            else if(opcao == 1){
                message("Selecionado Segue Linha\n");
                segueLinha();
            }
            else if(opcao == 2){
                message("Selecionado Segue Blocos\n");
                segueBloco();
            }
            else if(opcao == 3){
                message("Selecionado Sair\n");
                sair = TRUE;
            }

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

}
```

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 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;
```

myBlibioteca.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_Movimento.ic

```
#include sencdr0.icb
#include sencdr1.icb
float data_dif[50];
int data_power0[50];
int data_power1[50];
float vel0;
float vel1;

float mod(float num){
    if (num <= 0.0) num = (num * -1.0);
```

```
    return num;
}

void vel_control(float velocidade0, float velocidade1, float tempo){

    float erroanterior0;
    float erro1anterior;
    float derivativoerro1;
    float derivativo0;
    float errodifanterior;
    float errodif;
    float derivativodif;

    int count_ini0 = 0;
    int count_ini1 = 0;
    int count_atual0 = 0;
    int count_atual1 = 0;
    float vel_atual0 = 0.0;
    float vel_atual1 = 0.0;
    int d = 0;
    //velocidade em counts por segundo para diferentes potencias
    float curvamotor0[5]={2.12, 3.85, 4.54, 4.91, 5.27};
    float curvamotor1[5]={1.83, 3.45, 4.12, 4.46, 4.64};

    int power0;
    int power1;
    float power0aux;
    float power1aux;

    float erro0;
    float erro1;

    int i;
    int menor = 0;
    float dif0[5];
    float dif1[5];

    //velocidade setada em counts por segundo
    VEL0 = velocidade0;
    VEL1 = velocidade1;

    dif0[0] = mod(mod(VEL0) - curvamotor0[0]);
    dif0[1] = mod(mod(VEL0) - curvamotor0[1]);
    dif0[2] = mod(mod(VEL0) - curvamotor0[2]);
    dif0[3] = mod(mod(VEL0) - curvamotor0[3]);
    dif0[4] = mod(mod(VEL0) - curvamotor0[4]);
    for(i=1; i<5; i++){
        if(dif0[i]<dif0[i-1])
          menor = i;
    }
    power0aux = (VEL0/mod(VEL0))*(float)(menor+1)*20.0;
    menor = 0;
    dif1[0] = mod(mod(VEL1) - curvamotor1[0]);
    dif1[1] = mod(mod(VEL1) - curvamotor1[1]);
    dif1[2] = mod(mod(VEL1) - curvamotor1[2]);
    dif1[3] = mod(mod(VEL1) - curvamotor1[3]);
    dif1[4] = mod(mod(VEL1) - curvamotor1[4]);
    for(i=1; i<5; i++){
        if(dif1[i]<dif1[i-1])
```

```
            menor = i;
        }
        power1aux = (VEL1/mod(VEL1))*(float)(menor+1)*20.0;

        power0 = (int)(power0aux);
        power1 = (int)(power1aux);

        reset_system_time();
        encoder0_counts = 0;
        encoder1_counts = 0;

        while(seconds() < tempo){


            float Kp0 = 2.0;
            float Kd0 = 0.5;
            float Kp1 = 5.0;
            float Kd1 = 0.5;

            motor(0, power0);
            motor(1, power1);
            //Le a contagem do encoder no incio do intervalo
            count_ini0 = encoder0_counts;
            count_ini1 = encoder1_counts;
            //gerando um atraso de meio segundo
            sleep(0.5);
            //L a contagem do encoder ao final do intervalo
            count_atual0 = encoder0_counts;
            count_atual1 = encoder1_counts;
            /*determinando da velocidade*/
            vel_atual0 = ((float)(count_atual0 - count_ini0))*2.0;
            vel_atual1 = ((float)(count_atual1 - count_ini1))*2.0;
            //printf("%f e %f\n",vel_atual0, vel_atual1);

            erro1anterior = erro1;
            erro1 = VEL1 - vel_atual1;
            derivativoerro1 = (erro1 - erro1anterior)*2.0;
            errodifanterior = errodif;
            errodif = (vel_atual0 - vel_atual1) - (VEL0 - VEL1);
            derivativodif = (errodif - errodifanterior)*2.0;

            power0 += (int)(-errodif * Kp0 - derivativodif * Kd0);
            power1 += (int)(erro1 * Kp1 - derivativoerro1 * Kd1);

            if (power0 > 100) {power0 = 100;}
            if (power1 > 100) {power1 = 100;}
            if (power0 < -100) {power0 = -100;}
            if (power1 < -100) {power1 = -100;}



        }
}

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)(29.6 * (ang/360.0));
    motor(0,40);
    motor(1,-60);
    printf("counts = %d\n",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)(29.6 * (ang/360.0));
    motor(0,-40);
    motor(1,50);
    printf("counts = %d\n",counts);
    while(mot0 == 1 || mot1 == 1){
        if(encoder0_counts >= counts)
          {off(0);
            mot0 = 0;}
        if(encoder1_counts >= counts)
          {off(1);
            mot1 = 0;}
    }

}
```

polarizacao.ic

```
#include myBiblioteca.ic
#include Controle_Movimento.ic

int luzAmbiente = 0;
int goal;

void polarizacao() {
    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;
        angular_dir(45.0);
        ao();
        sleep(0.1);
        valor[i] = analog(sensorPolo);
        printf("Angulo = %d Now-%d\n", angle, valor[i] - goal);
        sleep(0.1);
    }

    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);

    angular_dir((float) selectedAngle);
    /* sleep(0.3);
    ao();
    pidLinear = start_process(vel_control(3.0,3.0));
    sleep(2.0);
    kill_process(pidLinear); ao();
     */

    motor(0,50); motor(1,50);
    sleep(3.0);
    ao();


}
```

segueLinha.ic

```
#include myBiblioteca.ic
#include Controle_Movimento.ic

int sup = 19600;
int inf = 10000;

void segueLinha() {
    int pidMostra = 0;
    int pid = 0;
    int button = -1;
    int valor = 0;
    int i = 0;
    float velD = 2.0;
    float velE = 2.0;
    int v1; int v0;
    //pid = start_process(vel_control(velE, velD)); //V0, V1
    //motor(0,50);motor(1,50);
    while (1) {

        valor = analog(sensorLinha);
        valor = valor * valor;
        i = i + 1;
        //printf("Sensor = %d, cont = %d,\n", valor, i);
```

```
        if (valor <= inf) {
            //motor(0,20);motor(1,10);
            velD = velD-1.0;
            vel1 = velD;

        } else if (valor >= sup) {

            velE = velE-1.0;
            vel0 = velE;

        } else if ((valor > inf) && (valor < sup)) {
            velD = 2.0;
            velE = 2.0;
            vel0 = velE;
            vel1 = velD;
        }

        v0 = (int)(velE*10.0);
        v1 = (int)(velD*10.0);
        motor(0,v0);
        motor(1,v1);
        sleep(0.3);
    }

    // sleep(0.2);
}
/*void main(){
    segueLinha();
}

*/
```

segueBloco.ic

```
#include myBiblioteca.ic
#include Controle_Movimento.ic
#include SegueLinha.ic

int ESQUERDA = 0;
int DIREITA = 1;
int EMPURRA = 2;
int PARE = 3;

int VERMELHO = 0;
int VERDE = 1;
int AMARELO = 2;
int AZUL = 3;

int numFuncao = 4; // Define quanti funcoes abaixo
int funcao[4]; // [Bloco]=acao

int tempo = 0;

int parar = FALSE;
int temObstaculo = FALSE;
int corObstaculo = -1;

int cao = 0;
int cor = 0;
```

```
int pidSegueLinha = 0;

void segueBloco() {

    //message("Funcao segueBloco\n");

    //menuBloco();

    configuracaoDefault();

    message("Inicio do movimento.\n");

    go();
    //pare();
}

void menuBloco() {

    int opcao = 0;
    int sair = FALSE;
    int button = -1;
    int n = 8;
    char strOpcao[8][16] = { "Tempo", "Azul", "Amarelo", "Vermelho",
"Verde",
          "Sair", "Config. Default", "Ver opcoes" };

    while (sair == FALSE) {

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

        button = waitingButtonPress();
        if (button == START) {
            if (opcao == 0) {
                //message("Selecionado polarizacao\n");
                getTime();
            } else if (opcao == 1) {
                message("Opcao do bloco AZUL\n");
                funcao[AZUL] = defineBloco();
            } else if (opcao == 2) {
                message("Opcao do bloco AMARELO\n");
                funcao[AMARELO] = defineBloco();
            } else if (opcao == 3) {
                message("Opcao do bloco VERMELHO\n");
                funcao[VERMELHO] = defineBloco();
            } else if (opcao == 4) {
                message("Opcao do bloco VERDE\n");
                funcao[VERDE] = defineBloco();
            } else if (opcao == 5) {
                message("Opcao SAIR\n");
                sair = TRUE;
            } else if (opcao == 6) {
                message("Opcoes DEFAULT\n");
                configuracaoDefault();
            } else if (opcao == 7) {
                message("Ver Opcoes Setadas\n");
                sair = TRUE;
            }
```

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

        button = -1;
    }

}

void configuracaoDefault() {
    funcao[AMARELO] = DIREITA;
    funcao[VERMELHO] = EMPURRA;
    funcao[VERDE] = PARE;
    funcao[AZUL] = ESQUERDA;
    tempo = 3;
}

void getTime() {
    int button = -1;
    int sair = FALSE;
    //char strOpcao[5][16] = {"Tempo","Azul","Amarelo", "Vermelho",
"Verde"};

    while (sair == FALSE) {

        printf("Stp END Star ++");
        printf("%d", tempo);
        printf("\n");

        button = waitingButtonPress();
        if (button == START) {
            tempo = tempo + 1;
        } else if (button == STOP) {
            sair = TRUE;
        }
        button = -1;
    }

}

int defineBloco() {
    int opcao = 0;
    int sair = FALSE;
    int button = -1;

    int n = 4;
    char strOpcao[4][16] = { "ESQUERDA", "DIREITA", "EMPURRA", "PARE"
};

    while (sair == FALSE) {

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

        button = waitingButtonPress();
        if (button == START) {
            if (opcao == 0) {
```

```
                message("Selecionado ESQUERDA\n");
                return ESQUERDA;
            } else if (opcao == 1) {
                message("Selecionado DIREITA\n");
                return DIREITA;
            } else if (opcao == 2) {
                message("Selecionado EMPURRA\n");
                return EMPURRA;
            } else if (opcao == 3) {
                message("Selecionado PARE\n");
                return PARE;
            }

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

}

void opcaoDireita() {
    int Anda;
    angular_dir(90.0);
    printf("angulo feito");
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < 3.0){
        //wait
    }
    kill_process(Anda);
    ao();
    angular_esq(90.0);
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < 6.0){
        //wait
    }
    kill_process(Anda);
    ao();
    angular_esq(90.0);
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < 3.0){
        //wait
    }
    kill_process(Anda);
    ao();
    angular_dir(90.0);
}

void opcaoEsquerda() {
    int Anda;
    angular_esq(90.0);
    printf("angulo feito");
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < 3.0){
```

```
        //wait
    }
    kill_process(Anda);
    ao();
    angular_dir(90.0);
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < 6.0){
        //wait
    }
    kill_process(Anda);
    ao();
    angular_dir(90.0);
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < 3.0){
        //wait
    }
    kill_process(Anda);
    ao();
    angular_esq(90.0);
}

void opcaoEmpurra() {
    int Anda;
    reset_system_time();
    Anda = start_process(vel_control(5.0,5.0));
    while(seconds() < (float)tempo){
        //wait
    }
    kill_process(Anda);
    ao();

}

void go() {
    int parar = FALSE;
    int pidSensor = 0;
    int bloco = -1;
    temObstaculo = FALSE;
    pidSegueLinha = 0;


    while (parar == FALSE) {
        pidSegueLinha = start_process(segueLinha());
        pidSensor = start_process(checaObstaculo());

        while (temObstaculo == FALSE) {
            sleep(0.5);
        }
        kill_process(pidSegueLinha);
        ao();
        kill_process(pidSensor);
        sleep(1.0);

        bloco = identificaBloco();

        sleep(1.0);
```

```
        executaFuncao(bloco);

        bloco = -1;
        temObstaculo = FALSE;
        corObstaculo = -1;

    }

}

void checaObstaculo() {
    int ambC = analog(4);
    int ambL = analog(5);
    int corini, corfin;
    int i = 0;
    int ComLedAzul;
    int ComLedVermelho;
    int SemLed;
    int difamb;

    /*beep();
  sleep(0.3);
  */
    beep();

    while (temObstaculo == FALSE) {
        bit_set(0x1009, 0b00111100);
        bit_clear(0x1008, 0b00111111);
        //bit_set(0x1008, ledAzul);

        cao = analog(5);
        cor = analog(4);
        sleep(0.5);
        motor(0, 20);
        motor(1, 20);
        //printf("Caolho(%d) Cor(%d)\n", cao, cor);
        //(corfin-corini)<20
        if(cao<140)//muita luz
          {
            while(cor<245 || difamb<10) {
                cor = analog(4);
                difamb = modint((analog(5)-cao));
                printf("cao%d caofx%d cor%d\n",analog(5),cao,cor);
                msleep(64L);
                //sleep(1.0);
                //corfin = analog(4);
                //  printf("Caolho(%d) Cor(%d)\n", cao, cor);
            }
        }
        if(cao>140)//pouca luz
          {
            while(cor<250 || difamb<10) {
                cor = analog(4);
                difamb = modint((analog(5)-cao));
                printf("cao%d caofx%d cor%d\n",analog(5),cao,cor);
                msleep(64L);
                //sleep(1.0);
                //corfin = analog(4);
                //  printf("Caolho(%d) Cor(%d)\n", cao, cor);
```

```
            }
        }
        kill_process(pidSegueLinha);
        ao();
        sleep(1.0);
        cor = analog(4);
        if(cao<120)
          {while (cor<250){
                motor(0, 15);
                motor(1, 15);
                cor = analog(4);
            }
        }
        else if(cao>120){
            while (cor<253){
                motor(0, 15);
                motor(1, 15);
                cor = analog(4);
            }
        }
        ao();
        SemLed = analog(4);
        cao = analog(5);
        bit_set(0x1008, ledAzul);
        sleep(1.0);
        ComLedAzul = analog(4);
        bit_clear(0x1008, 0b00111111);
        bit_set(0x1008, ledVermelho);
        sleep(1.0);
        ComLedVermelho = analog(4);
        if (cao < 80) {
            message("AMARELO\n");
            corObstaculo = AMARELO;
            temObstaculo = TRUE;
        } else if ((SemLed-ComLedVermelho)<20) {
            if(((float)ComLedAzul/(float)ComLedVermelho) < 0.7) {
                message("AZUL\n");
                corObstaculo = AZUL;
                temObstaculo = TRUE;}
            else if (((float)ComLedAzul/(float)ComLedVermelho) > 0.7)
{
                message("VERDE\n");
                corObstaculo = VERDE;
                temObstaculo = TRUE;
            }
        } else if((SemLed-ComLedVermelho)>30){

            message("VERMELHO\n");
            corObstaculo = VERMELHO;
            temObstaculo = TRUE;
        }

    }
    sleep(0.1);

}

int identificaBloco() {
```

```
    return corObstaculo;
}

void executaFuncao(int bloco) {
    int i = 0;

    if (bloco >= 0 && bloco < 4) {
        if (funcao[bloco] == ESQUERDA) {
            message("Funcao esquerda\n");
            opcaoEsquerda();
        } else if (funcao[bloco] == DIREITA) {
            opcaoDireita();
            message("Funcao direita\n");
        } else if (funcao[bloco] == EMPURRA) {
            message("Funcao empurra\n");
            opcaoEmpurra();
        } else if (funcao[bloco] == PARE) {
            message("Funcao parar\n");
            parar = TRUE;
        } else {
            printf("Bloco %d ", bloco);
            message("sem Func\n");
        }
    }else{
        printf("Bloco %d ", bloco);
        message("nao existe\n");


    }
}

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

8. Referencias Bibliográficas

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

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