Relatório Final

Projeto "Meu Cão Guia"

Násser Yousef Santana Ali - 13/0034398
Universidade de Brasília - FGA
Área Especial de Indústria Projeção A, UNB - DF-480-Gama Leste, Brasília - DF, 72444-240
nasser.yousef.unb@gmail.com

Natália Santos Mendes – 13/0128082
Universidade de Brasília - FGA
Área Especial de Indústria Projeção A, UNB - DF-480-Gama Leste, Brasília - DF, 72444-240
nataliammendes04@gmail.com

Resumo: **A Raspberry pode ser explicada como um computador em uma pequena placa. Logo, seu poder de processamento é alto, e sua utilização ampla. Pensando nisso, pensou-se no projeto "Meu Cão Guia", que é um robô capaz de passar em pequenos ambientes, com capacidade de "visualizar" o lugar antes do usuário.**

Palavras-chave: **Raspberry Pi 3; Robô; Robô com câmera.**

***Abstract*— Raspberry can be explained as a computer on a small board. Therefore, its processing power is high, and its use wide. With this in mind, the "My Dog Guide" project was thought of as a robot capable of passing in small environments, with the ability to "visualize" the place before the user.**

Keywords: Raspberry Pi 3; Robot; Camera Robot.

I. Introdução

Atualmente tudo envolve tecnologia, uso de softwares, programas recursos, sensores, transdutores, robôs e etc. para o auxílio de tarefas, desde as mais simples até as mais complexas. Dessa forma, existem áreas de difícil acesso para seres humanos, seja pelo tamanho, temperatura, hostilidade do ambiente e etc. Exemplos dessas áreas existem dutos de ar, lajes, área que oferecem algum tipo de risco, as que necessitam de verificação constante ou precisam de manutenção. Algumas áreas ainda não contam com essa ajuda para tarefas do cotidiano. O acesso a determinadas áreas pode ser complicado e pode necessitar de maiores investimentos. O acesso proporcionado a essas áreas demanda serviço especializado, ou pelo menos envolve dúvidas e dificuldades em resolver os problemas associados.

Com o intuito de resolver tal, pensou-se no projeto "Meu Cão Guia". Tal como os cães guias direcionam o deficiente para os lugares que eles precisam e não podem enxergar, o robô proposto teria tamanho suficiente para adentrar em lugares de difícil acesso, "enxergar" todo o ambiente para o usuário e assim seria possível decidir o que deve ser feito no local antes mesmo de entrar nele. Em um projeto de maior porte pode-se citar que o próprio robô seja o agente neutralizador de problemas.

II. Objetivos

O objetivo é criar um robô que possa ser controlado pelo usuário para andar em ambientes de difícil acesso oferecendo imagens do local. Para isso, o robô terá uma câmera acoplada onde oferecerá as imagens em tempo real, além de um LED que servirá como uma lanterna caso o ambiente esteja escuro. Também terá que ter um tamanho reduzido e rodas que possam superar alguns obstáculos.

III. Descrição do Hardware

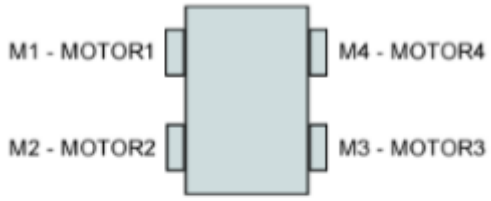
Para todo o projeto, serão utilizados seguintes componentes:

- Raspberry Pi 3 – Model B (RPi3);
- Controladores de Motor: CI L293D;
- Baterias de alimentação (7,4V – 3800mA/h);
- Bateria Auxiliar p/ Celular (5V – 2.1 A – 10000 mA/h);
- Reguladores de Tensão: CI 7805;
- Mini-Protoboards – 170 pinos;
- Câmera Module 5MP p/ Raspberry Pi;
- Estrutura do Robô (Chassi 4WD);
- 4 Motores DC, c/ caixa de redução;
- Fios e Jumpers Macho-Fêmea;
- 2 - Transistores NPN BC547;
- 1 - Servomotor Futaba S003;
- 2 – Resistores 1KΩ;
- LEDs de alto brilho 5mm;
- Buzzer 5V;
- Computador Pessoal - Notebook;
- Rede em Comum (Wi-Fi);

Neste ponto de controle, foram atualizadas as definições de de controle dos motores, LEDs de iluminação, servo-motor, buzzer e a alocação da imagem da PiCâmera.

É necessário, portanto, conhecer como se baseia o funcionamento dos motores, bem como o seu controle. Os motores DC, para a devida implementação, precisam agir de

forma coordenada. Para ir para frente, todos os motores precisam ir para frente; para trás, todos os motores devem girar para trás; para a direita, parte deve ir para frente e parte para trás, assim como para a esquerda. A *Figura 01* pode ilustrar tal ação com o seguinte esquemático.



AÇÃO

	M1	M2	M3	M4
P/ FRENTE	Frente	Frente	Frente	Frente
P/ TRÁS	Trás	Trás	Trás	Trás
P/ ESQUERDA	Trás	Trás	Frente	Frente
P/ DIREITA	Frente	Frente	Trás	Trás

Figura 1 - Esquemático de trabalho dos motores para realização dos movimentos.

Para haver a comutação de sentido de rotação do motor, é necessária uma ponte H, que inverte a polaridade que está sendo direcionada ao motor DC, que faz que com que seja possível a comutação de sentido. O esquemático básico de uma ponte H pode ser ilustrado na *Figura 02*. As comutações de ligações permitem que sejam invertidas a polaridades da alimentação que é dada ao motor, fazendo com que tal gire no sentido horário e anti-horário. Mais especificamente, uma ponte H é construída com transistores e diodos de proteção, para que garanta o chaveamento e a passagem de corrente elétrica na direção correta para cada caso.

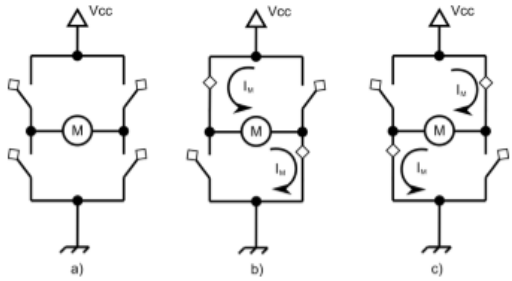


Figura 2 – Esquemático de funcionamento de uma ponte H.

O driver utilizado é o CI L293D, a qual permite o controle de 2 motores DC. O esquemático de ligação e controle do mesmo poder ser visualizado na *Figura 03*. Os pinos Pin 4, Pin 5, Pin 6 e Pin 7 são aqueles que são conectados ao RPi, responsáveis pela comutação dos motores. O pino 16 é responsável pela alimentação do circuito, ligado em 5v. Já os pinos 1 (motor A) e 9 (motor B) são pinos que recebem uma entrada analógica de 0v a 5v e, de acordo com a tensão, há o controle da velocidade dos motores. Ambos são colocados em +5v. O pino 8 é responsável pela alimentação dos motores, e tal pino pode variar de 3v a 36v. Tal foi colocado com 7,4v, diretamente com a alimentação da bateria

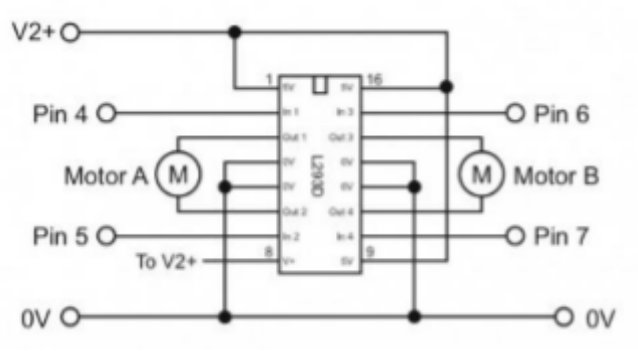


Figura 3 - Esquema de ligação e alimentação do CI L293D.

A ponte H, como já foi dito, é composto por um circuito composto de transistores, para a comutação das polaridades dos motores; e os diodos são colocados a caráter de proteção do circuito, além de apontar a direção correta da corrente elétrica. Tal ilustração pode ser vista por meio da *Figura 04*.

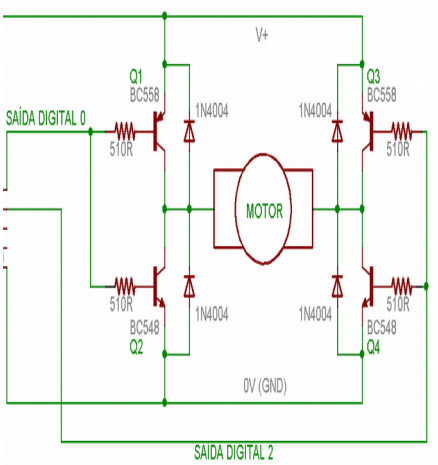


Figura 4 – Esquemático de uma Ponte H

Os servo-motores são mecanismos que giram determinada angulação, com relação a certo eixo de referência, por volta de 200º. Tal angulação é definida por meio de um sinal PWM. Tal sinal pode ser definido por uma onda quadrada, que varia o tempo que permanece em nível lógico baixo ou nível lógico alto, chamado DUTY CYCLE; tal alusão pode ser vista na *Figura 05*. No caso, de acordo com o datasheet e suas especificações técnicas, o servo motor Futaba S003, faz uso de

um sinal de 30Hz, variando sua angulação de 0º a 200º, com contagens de 500 a 3000 contagens de pulso dentro do período de 30 Hz.

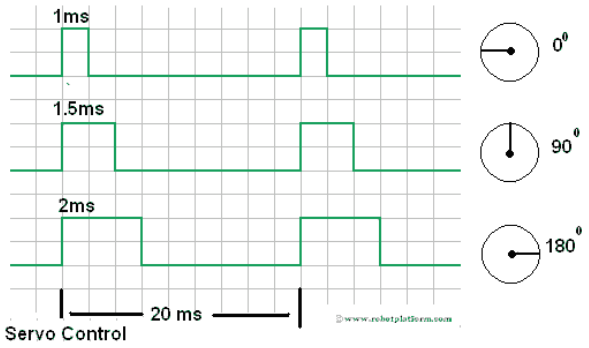


Figura 5 – Esquemático do sinal PWM com funcionamento da angulação do Servo-Motor

A ilustração do servo motor utilizado no projeto pode ser ilustrado na *Figura 07*. Bem como o chassi do carro, rodas e motores DC podem ser visualizados na *Figura 06*.

Vale lembrar que o servo motor será utilizado para ser o “pescoço” da PiCâmera, dando maior mobilidade para a visibilidade do usuário final, sem precisar mover o carrinho para a esquerda ou direita.

Figura 6 – Chassi do Carrinho

Figura 7 – Servo Motor Futaba S003

O RPi é um dispositivo de controle, por isso não é bem dotado de capacidade de fornecimento de corrente nem detém de tensões altas em seus pinos GPIO. Cada pino I/O do RPi fornece cerca de +3,3V e um máximo de 15mA por pino. Por esse motivo, não é possível ativar componentes que precisam de maior tensão ou corrente de forma direta, podendo queimar as portas e até maiores defeitos na RPi.

Uma solução para esse tipo de problema é a utilização de transistores, configurados para funcionarem como chave de seleção. No presente projeto, fora utilizado o transistor bipolar de junção (TBJ); sua ilustração pode ser vista na *Figura 8*. Tal chave é controlada por corrente elétrica, ou seja, de grosso modo, quando é injetada uma corrente elétrica na base do transistor, permite-se a passagem de corrente de maior ordem do coletor para o emissor. Lembrando que tal pensamento vale para transistores do tipo NPN, sendo do tipo PNP é o pensamento dual do apresentado.

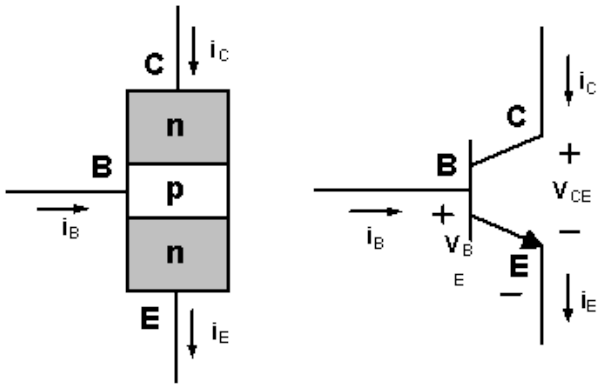


Figura 8 – Símbolo TBJ

Dessa forma, é possível especificar um resistor de polarização, que ajuste a corrente necessária para ser injetada na base do transistor para chaveamento. Tal corrente deve ser

dimensionada para o máximo de 15mA, por limitação da RPi. Sabendo que a RPi fornece +3,3V e que a tensão Vbe (tensão de base-emissor) típica do TBJ BC547B é de 0,7V, faz-se a seguinte conta:

```
Pela Lei de Ohm: I = V / R;
Tomando R = 1K
Ib = 3,3 – 0,7/1K = 2,6mA
```

Dessa forma, obtém-se uma corrente de polarização é de, aproximadamente, 2,6mA, não ferindo o máximo da capacidade da RPi, com Rb = 1KΩ. Isso faz com que o transistor seja “ativado” quando o pino I/O vai pra nível lógico alto (+3,3V), permitindo a passagem da corrente Ic, passando para a carga Rc e ativando o dispositivo pendurado.

O circuito que foi utilizado para o chaveamento de componentes pode ser visto na *Figura 9*. Sendo assim, é possível “pendurar” as cargas no transistor, representado por Rc. O Vcc é a alimentação das baterias, já passado pelo regulador de tensão, 7805; sendo então Vcc = 5V. O Vb são os +3,3v advindos do RPi. Tal tipo de ligação foi feita para a ativação do Buzzer e dos LEDs de iluminação.

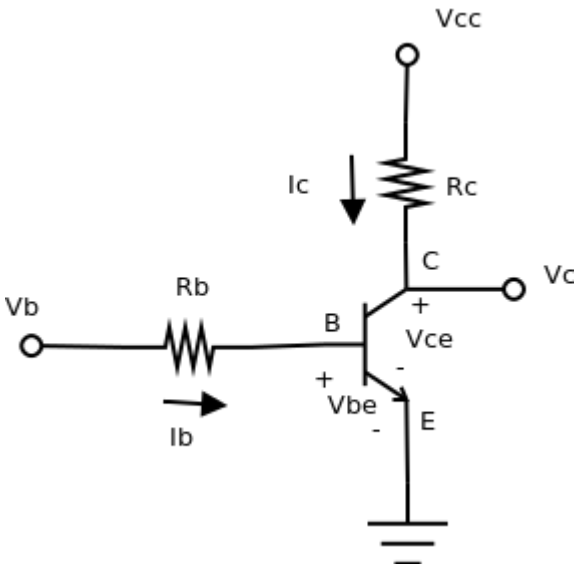


Figura 9 – Esquemático circuito de Chaveamento com Transistor TBJ

A ilustração dos LEDs de alto brilho utilizados, bem como o buzzer de aviso podem ser vistos na *Figura 10* e *Figura 11*.

Figura 10 – Buzzer 5V

Figura 11 – LED de alto brilho BRANCO 5mm

IV. Descrição do Software

Primeiramente, foram declaradas as bibliotecas usadas para esse código. Foi usada a WiringPi.h para a parte GPIO, Pthread.h para declaração e utilização de threads, stdio.h para entrada e saída, stdlib.h para funções como printf, termios.h e unistd.h para a função de obter as entradas. Também utilizou-se a softServo.h para controle do servo motor, errno.h para tratamento de erros em aberturas de bibliotecas. Posteriormente foram definidos todos os pinos I/O responsáveis por cada dispositivo já descrito.

Na *Figura 12* pode ser ver o mapeamento dos pinos que foi feito até o presente momento, juntamente com a tabela que descreve o andar dos motores DC.

```
/*****************************************************
****************** PROJETO EMBARCADOS ******************
*****************************************************

Integrantes:
    >> Násser Yousef Santana Ali - 13/0034398
    >> Natália Santos Mendes - 13/0128082

>> MAPEAMENTO DE PINOS

    PINOS   GPIO (WIRINGPI)     FUNÇÃO          COR

1.  11 -------- GPIO 00 -------- M1 - I1 ------ VERMELHO
2.  12 -------- GPIO 01 -------- M1 - I2 ------ LARANJA
3.  13 -------- GPIO 02 -------- M2 - I1 ------ AMARELO
4.  15 -------- GPIO 03 -------- M2 - I2 ------ VERDE
5.  16 -------- GPIO 04 -------- M3 - I1 ------ BRANCO
6.  18 -------- GPIO 05 -------- M3 - I2 ------ CINZA
7.  22 -------- GPIO 06 -------- M4 - I1 ------ VIOLETA
8.  07 -------- GPIO 07 -------- M4 - I2 ------ PRETO
9.  29 -------- GPIO 21 --------
10. 31 -------- GPIO 22 -------- BUZZER ------- VIOLETA
11. 33 -------- GPIO 23 -------- LED ---------- CINZA
12. 35 -------- GPIO 24 --------
13. 37 -------- GPIO 25 -------- SERVO MOTOR -- AZUL
14. 32 -------- GPIO 26 --------
15. 36 -------- GPIO 27 --------
16. 38 -------- GPIO 28 --------
17. 40 -------- GPIO 29 --------

>> DEFINIÇÕES PARA O CONTROLE DOS MOTORES

I1  I2      AÇÃO
0   0       PARADO
1   0       IR P/ FRENTE
0   1       IR P/ TRÁS
1   1       PARADO

*****************************************************/
```

Figura 12 – Mapeamento de Pinos

Na *Figura 13* são definidas as constantes que serão utilizadas durante a execução do programa, bem como a declaração de variáveis globais e os protótipos das funções usadas.

```
#define motor1Pin1 0 // Definindo os pinos para
#define motor1Pin2 1 // cada um dos dois fios de
#define motor2Pin1 2 // cada motor
#define motor2Pin2 3
#define motor3Pin1 4
#define motor3Pin2 5
#define motor4Pin1 6
#define motor4Pin2 7

#define SERVO 25
#define BUZZER 22
#define LED 23

#define MAX_SERVOS 1

// Variáveis e Funções Globais
static int pinMap     [MAX_SERVOS] ; // Para mapear os pinos
static int pulseWidth [MAX_SERVOS] ; // microsegundos
static PI_THREAD (softServoThread); // Função para o PWM

// Protótipos das Funções
// Função que define a angulação do servomotor
void softServoWrite (int servoPin, int value);
// Setup para o servomotor
int softServoSetup (int p0, int p1, int p2, int p3, int p4, i
// Função para pegar a entrada sem precisar do enter
int getch(void);
```

Figura 13 – Declaração de Constantes e Variáveis Globais

Ainda a citar a seção de código presente na *Figura 13*, a função que cria um sinal PWM para controle do servo motor, é possível endereçar até 8 pinos de controle. Mas como só se utiliza um único servo nesse projeto, a constante MAX_SERVOS foi mudada para 1. Dessa forma é criado um array de apenas uma posição, que é o controle desse único pino de PWM. As variáveis pinMap e PulseWidth são variáveis globais utilizadas pelas funções de controle do servo motor. As funções softServoWrite() serve para escrever o valor da angulação em determinado pino, enquanto a softServoSetup() é para dar início à configuração do sinal e controle do servo motor, nos pinos indicados em seus parâmetros.

```
int softServoSetup (int p0, int p1, int p2, int p3,
 int p4, int p5, int p6, int p7)
{
    int servo ;

    if (p0 != -1) { pinMode (p0, OUTPUT) ;
     digitalWrite (p0, LOW) ; }
    if (p1 != -1) { pinMode (p1, OUTPUT) ;
     digitalWrite (p1, LOW) ; }
    if (p2 != -1) { pinMode (p2, OUTPUT) ;
        digitalWrite (p2, LOW) ; }
    if (p3 != -1) { pinMode (p3, OUTPUT) ;
        digitalWrite (p3, LOW) ; }
    if (p4 != -1) { pinMode (p4, OUTPUT) ;
        digitalWrite (p4, LOW) ; }
    if (p5 != -1) { pinMode (p5, OUTPUT) ;
        digitalWrite (p5, LOW) ; }
    if (p6 != -1) { pinMode (p6, OUTPUT) ;
        digitalWrite (p6, LOW) ; }
    if (p7 != -1) { pinMode (p7, OUTPUT) ;
        digitalWrite (p7, LOW) ; }

    pinMap [0] = p0 ;
    pinMap [1] = p1 ;
    pinMap [2] = p2 ;
    pinMap [3] = p3 ;
    pinMap [4] = p4 ;
    pinMap [5] = p5 ;
    pinMap [6] = p6 ;
    pinMap [7] = p7 ;

    for (servo = 0 ; servo < MAX_SERVOS ; ++servo)
        pulseWidth [servo] = 1500 ;      // Mid point

    return piThreadCreate (softServoThread) ;
}
```

Figura 14 – Função de Setup do Servo

Na *Figura 14*, é possível verificar a função de dar início ao controle do sinal PWM, onde, primeiramente, são definidos os pinos como saída digital e são mandados para nível lógico baixo. Cada pino é devidamente mapeado no array pinMap. Sabendo que, por só termos 1 único servo, ele só tem a primeira posição.

É definido uma largura de pulso inicial no vetor pulseWidth para esse mesmo pino criado e é dado um return na piThreadCreate para a devida criação do sinal PWM, esta biblioteca vai montar o sinal conforme as especificações de período de onda quadrada para controle do servo.

Na *Figura 15* pode-ser ver o início do programa principal, iniciando a biblioteca WiringPi.h e informando uma mensagem de erro caso o setup não seja concluído. No comando system – que tem como função executar uma linha de comando do terminal – consta todo o comando para a utilização da câmera para streaming. Foi usado uma biblioteca que já consta na Raspberry Pi, então o único programa que foi necessário instalar foi o VLC Player. Traduzindo essa linha de comando, ela passa a imagem stream para outro dispositivo através do protocolo http na porta 8090.

Esse comando "raspivid" pode receber diversos parâmetros. Esses parâmetros podem ser a definição da resolução da imagem, filtros (tais como preto e branco ou sépia) ou até girar a imagem. Como a câmera estava em um ponto parada, os únicos parâmetros utilizados foram o "-o" que define a saída para ser escrita em stdout, o "-t 0" coloca para capturar a imagem por "tempo infinito" (ou seja, não para a streaming até que o usuário force a parada); o "-w 600 -h 600" definem a resolução da imagem e o "-fps 12" define quantas frames por segundo a câmera irá capturar.

O "cvlc" é o comando do VLC Player. A imagem passada pela câmera pode ser transmitida por outro computador apenas abrindo o VLC Player e digitando o protocolo utilizado, o IP da Raspberry e a porta utilizada.

Ainda na *Figura 15*, foi iniciado a função de controle do servo motor, bem como são definidos os pinos de saída para o controle dos motores DC, LEDs (iluminação) e BUZZER (alerta sonoro). A função utilizada é a PinMode, própria da biblioteca WiringPi.h, utilizada para configurar pinos de entrada ou saída.

```
int main(int argc, char *argv[])
{
    if (wiringPiSetup() == -1)
    {
        fprintf (stdout, "oops: %s\n", strerror (errno)) ;
        exit(-1);
    }

    // Comando da PiCâmera
    system("raspivid -o - -t 0 -w 600 -h 400 -fps 12
     |cvlc -vvv stream:///dev/stdin --sout
     '#standard{access=http,mux=ts,dst=:8090}'
     :demux=h264");

    softServoSetup(SERVO, -1, -1, -1, -1, -1, -1, -1);

    pinMode(motor1Pin1, OUTPUT); // Declara todos como
    pinMode(motor1Pin2, OUTPUT); // saída
    pinMode(motor2Pin1, OUTPUT);
    pinMode(motor2Pin2, OUTPUT);
    pinMode(motor3Pin1, OUTPUT);
    pinMode(motor3Pin2, OUTPUT);
    pinMode(motor4Pin1, OUTPUT);
    pinMode(motor4Pin2, OUTPUT);

    pinMode(LED, OUTPUT);
    pinMode(BUZZER, OUTPUT);
```

Figura 15 – Setups das bibliotecas

Na *Figura 16*, é mostrado o comando para o motor que controla as rodas. Nesse caso, o comando para ir para frente. Como já foi explicado o funcionamento de cada roda para ir para uma determinada direção, o necessário foi apenas setar e zerar os pinos necessários através da função digitalWrite da wiringPi.h, cujos parâmetros são o pino (que foi definido anteriormente no código) e o nível que deseja-se colocar.

Além disso, colocou-se o delay de 25 ms. Inicialmente, o delay colocado era maior. Porém, se o usuário segurasse a tecla do teclado por muito tempo, o robô continuava andando por mais algum tempo depois que o usuário soltasse essa tecla. Para solucionar esse problema, diminuiu-se o tempo de delay e dessa forma independente de como o usuário pressionar o botão do teclado, o carro parará de andar assim que ele soltar.

```
while(1)
{
    ch = getch(); // Para pegar a entrada do usuário

    if (ch == 'w') // Ir para frente
    {
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, HIGH);
        digitalWrite(motor2Pin1, LOW);
        digitalWrite(motor2Pin2, HIGH);
        digitalWrite(motor3Pin1, LOW);
        digitalWrite(motor3Pin2, HIGH);
        digitalWrite(motor4Pin1, LOW);
        digitalWrite(motor4Pin2, HIGH);
        delay(25);
    }
```

Figura 16 – Trecho de comando do motor

As *Figuras 17 e 18* são as partes do código para controlar a lanterna (conjunto de LEDs) e o buzzer. Para lanterna, se o usuário pressionar a letra "l" uma vez, o pino responsável pelos LEDs fica em alto e a variável "lantern" recebe um novo valor diferente de zero para, até o usuário pressionar a letra "L" novamente (para apagar), a luz continuará acesa. Já para o buzzer, o pino dele fica em alto por um curto período de tempo emitindo seu som, depois vai para o nível baixo para parar.

```
else if (ch == 'l') // On/Off LEDs
{
    if (lantern == 0){
        digitalWrite(LED, HIGH);
        lantern = 1;
    }else{
        digitalWrite(LED, LOW);
        lantern = 0;
    }
}
```

Figura 17 – On/Off LEDs

```
else if (ch == 'b') // BUZZER
{
    digitalWrite(BUZZER, HIGH);
    delay(200);
    digitalWrite(BUZZER, LOW);
}
```

Figura 18 – Controle do Alerta Sonoro (Buzzer)

O código da *Figura 19* descreve a lógica para o controle do servo motor. As entradas dadas pelo usuário através do teclado para isso são “V” (para virar o servo para esquerda) e “V” (para virar o servo para direita). Para isso, utilizou-se de controlar uma mesma váriavel: value_servo. É válido usar a mesma variável em situações diferentes pois a entrada nunca será “C” e “V” de uma vez, logo, a variável value_servo só será atribuída com algum valor separadamente.

Então, para controlar, basta enviar um valor para o servo através da função softServoWrite que a mesma anda a angulação descrita. Para esse caso, a cada entrada do usuário, aumenta-se em 100 o valor do value_servo. Os valores máximo e mínimo são de -250 e 1250. Se a value_servo receber um valor menor ou maior que isso, o servo motor tem comportamentos inesperados.

```
else if (ch == 'c') // AUMENTA SERVO
{
    value_servo += 100;

    if (value_servo < -250)
        value_servo = -250;
    else if (value_servo > 1250)
        value_servo = 1250;

    softServoWrite (SERVO, value_servo);
}
else if (ch == 'v') // DIMINUI SERVO
{
    value_servo -= 100;

    if (value_servo < -250)
        value_servo = -250;
    else if (value_servo > 1250)
        value_servo = 1250;

    softServoWrite (SERVO, value_servo);
}
```

Figura 19 – Escrita no Servo Motor

O programa principal primeiramente obtém-se a entrada, que são letras (ou seja, tipo char) e podem ser “W” (para o robô ir para frente), “A” (para ir para esquerda), “S” (para ir para trás), “D” (para ir para direita), “L” para ligar ou desligar a lanterna (LEDs) e, por fim, “B” para dar o alerta sonoro, por meio do BUZZER . Para conseguir armazenar essa entrada sem ser preciso apertar “enter”, foi necessário usar uma função que usa a struct que existe na biblioteca termios.h. Dessa forma, ao digitar a entrada, não é necessário apertar “enter”, pois a entrada já é armazenada assim que é inserida.

```
void *function_camera()
{
    struct sched_param sp;
    sp.sched_priority = 80; // Dá prioridade 80 p/ essa thread

    if(pthread_setschedparam(pthread_self(), SCHED_FIFO, &sp))
        fprintf(stderr,"WARNING: Failed to set stepper thread
         to real-time priority.\n");

    // Comando da PiCâmera: Realiza o Streamming
    system("raspivid -n -o - -t 0 -w 600 -h 400 -fps 23 |cvlc
     -vvv stream:///dev/stdin --sout '#standard{access=http,
        mux=ts,dst=:8090}' :demux=h264");

    return NULL;
}
```

Figura 20 – Thread de Câmera

A *Figura 20*, por fim, mostra que para todo esse controle de todos esses componentes, foi usada uma thread somente para a câmera (que inclusive não precisa de nenhum parâmetro como entrada na função) e os outros controles na thread principal da main.

```
struct sched_param sp;
sp.sched_priority = 99; // Dá prioridade 99 p/ essa thread

if(pthread_setschedparam(pthread_self(), SCHED_FIFO, &sp))
    fprintf(stderr,"WARNING: Failed to set stepper thread
     to real-time priority.\n");
```

Figura 21 – Prioridade de motores

Um dos problemas encontrados durante a execução do projeto foi que como a Raspberry executava vários processos ao mesmo tempo, os processos principais para o projeto (tais como controle dos motores e câmera) não tinham prioridade e eram executados assim que possível, causando lentidão no robô.

Para solucionar esse problema, foi necessário dar prioridade para thread desejada. Para isso, configurou-se o sistema operacional para tempo real preemptivo. Esse tempo real (também conhecido como RTOS) é um sistema operacional com tempo de resposta pré-definido. O preemptivo, nesse contexto, é quando o OS é capaz de tirar um processo de execução e colocar outro em execução, dada a prioridade de cada processo.

Isso se dá através dos algoritmos de escalonamento. Para o sistema preemptivo, o algoritmo é classificado como estático e chamado por escalonamento por taxas monotônicas. Esse algoritmo, se não dada a prioridade diretamente para o processo anteriormente, distribui prioridades de acordo com a frequência em que ele é executado e coloca em uma “agenda” para execução. [1]

Para configurar o sistema operacional, foi necessário fazer o download do patch do sistema real preemptivo, de acordo com o hardware e modelo da Raspberry utilizada no projeto (Pi 3 Modelo B). Como o arquivo já estava pronto, foi necessário apenas executar alguns passos de exclusão de algumas pastas e posteriormente dar o reboot na Raspberry. Para verificar se ocorreu tudo certo na configuração, usou-se o comando “uname -a” no terminal e foi possível visualizar que o sistema preemptivo tinha sido instalado.

Finalmente, escreveu-se alguns comandos no código do projeto para dar a prioridade na thread da câmera e controle dos motores, como é possível ver nas figuras 20 e 21. Para dar essa prioridade, cria-se uma variável do tipo struct sched_param com nome sp, e depois atribui-se uma prioridade com valor máximo de 99. Depois, na função p_thread_setschedparam coloca-se esse processo com prioridade em uma “agenda” para executar. Como parâmetros, essa função recebe qual thread será executada (no caso, será ela mesmo), o tipo de entrada e saída (FIFO = first in first out) e a variável do tipo struct criada.

Esse código para dar prioridades às threads foi feito na thread da câmera e do controle dos motores. A única diferença é a prioridade dada para cada processo: 99 – máxima – para os motores e 80 para câmera.

Para compilar o programa que fora descrito, são usados os seguintes comandos, considerando o ambiente como um compilador GCC em GNU/Linux, Ubuntu 16.04 LTS. Considere que o nome do arquivo seja “project.c” e o nome do executável gerado seja “exe”.

```
gcc project.c –o exe –lwiringPi –lpthread
```

V. Resultados

Tendo feito e executado tal projeto, foram encontradas dificuldades na montagem e organização para que tudo funcionasse em conformidade. Nos motores DC, a dificuldade ficou em sua plena alimentação, bem como a conformidade dos 4 motores para realizarem os movimentos de ir para frente, trás, esquerda e direita.

Como a streaming (transmissão ao vivo), requer bastante processamento da RPi, o programa foi simplificado ao máximo, com o intuito de dispor apenas de 2 threads (fluxos de execução) principais, sendo a primeira para o controle dos motores, alerta sonoro, on/off dos LEDs e o controle do servo motor e a segunda para o streaming com a PiCâmera. O problema de mutithreading pode ser visível e significativo quanto se está trabalhando com aplicações que estão rodando em paralelo e necessitam de maior poder de processamento.

Porém, com todas as dificuldades, ao final, a etapa fora concluída com êxito. O servo motor consegue alcançar toda a angulação necessária para a direita e esquerda. Entretanto, uma ideia de aprimoramento é informar quantos graus o servo está virado, pois assim, o usuário pode se localizar de onde o carro robô está bem como a posição do servo motor. Além disso, nesse ponto de controle também se acoplou a câmera ao servomotor e fez-se o teste para verificar se tanto os motores quanto a câmera funcionariam juntos sem nenhum tipo de “bug”, pois, como dito anteriormente, a câmera precisa de muito processamento da Raspberry, logo poderia atrapalhar o desenvolvimento dos outros controles.

Para os LEDs e o Buzzer que não haviam sido testados anteriormente, foi necessário adicionar ao circuito transistores para fornecer a corrente necessária aos componentes, como já mostrado no esquemático neste relatório.

Pelo controle de motores exigir certa precisão de controle e assim como a streaming de câmera também puxar certo processamento da raspberry, ambos davam alguns travamentos em sua execução. Um modo de melhorar tal execução é dar mais prioridade ao processo que está sendo executado. Foi configurado um sistema preemptivo, ou seja, que coloca o escalonador de processos com interrupção para processos que detém de maior prioridade. Dessa forma, os outros processos que estão sendo executados em paralelo pela raspberry são colocados na fila e as threads de câmera e de controle de motor detém de maior prioridade na execução.

VI. Conclusão

O RPi é uma boa plataforma de desenvolvimento de projetos, consegue se sair bem em projetos e controle, como fora feito com o robô. Por seus pinos fornecem tensões baixas e corrente, com relação aos outros microcontroladores, pode-se notar que o seu propósito é o controle de operações.

Dentre as dificuldades, uma das maiores foi a definição da melhor característica e configuração de bateria que era necessária para alimentação dos 4 motores DC, 2 CIs L293D e alimentação dos LEDs, de modo que tudo funcionasse de forma plena.

Além disso, um dos maiores desafios encontrados para o próximo ponto de controle é diminuir o delay da imagem da câmera sem precisar diminuir a resolução da imagem.

Dentre as dificuldades de implementação de um sistema preemptivo reside na instalação de um kernel preemptivo. É necessário realizar determinadas operações como super usuário no Raspbian de forma que seja possível compilar um kernel adequado para tal operação.

Para melhorias ao projeto, deseja-se além de informar ao usuário a angulação do servomotor, também colocar um sensor de proximidade para avisar ao usuário se algum obstáculo está chegando, diminuir o delay da câmera e melhorar a interface ser humano e máquina.

De maneira geral, tudo ocorreu conforme o esperado, dentro do comportamento e aparição de possíveis problemas no desenvolvimento do código de controle e execução por parte da RPi.

VII. Referências

[1] TANENBAUM, A. S. Sistemas Operacionais Modernos. 2ª ed. São Paulo, Pearson Prentice Hall, 2003.

ANEXO I

```
/*********************************************************
****************** PROJETO EMBARCADOS *******************
*********************************************************

Integrantes:
      >> Násser Yousef Santana Ali - 13/0034398
      >> Natália Santos Mendes - 13/0128082

>> MAPEAMENTO DE PINOS

      PINOS       GPIO (WIRINGPI)         FUNÇÃO                  COR

1.    11 -------- GPIO 00 -------- M1 - I1 ------ VERMELHO
2.    12 -------- GPIO 01 -------- M1 - I2 ------ LARANJA
3.    13 -------- GPIO 02 -------- M2 - I1 ------ AMARELO
4.    15 -------- GPIO 03 -------- M2 - I2 ------ VERDE
5.    16 -------- GPIO 04 -------- M3 - I1 ------ BRANCO
6.    18 -------- GPIO 05 -------- M3 - I2 ------ CINZA
7.    22 -------- GPIO 06 -------- M4 - I1 ------     VIOLETA
8.    07 -------- GPIO 07 -------- M4 - I2 ------ PRETO
9.    29 -------- GPIO 21 --------
10.   31 -------- GPIO 22 -------- BUZZER ------- VIOLETA
11.   33 -------- GPIO 23 -------- LED ---------- CINZA
12.   35 -------- GPIO 24 --------
13.   37 -------- GPIO 25 -------- SERVO MOTOR -- AZUL
14.   32 -------- GPIO 26 --------
15.   36 -------- GPIO 27 --------
16.   38 -------- GPIO 28 --------
17.   40 -------- GPIO 29 --------

>> DEFINIÇÕES PARA O CONTROLE DOS MOTORES

I1    I2          AÇÃO
0     0           PARADO
1     0           IR P/ FRENTE
0     1           IR P/ TRÁS
1     1           PARADO

*********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <termios.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <softServo.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>
```

```
#define motor1Pin1 0
#define motor1Pin2 1
#define motor2Pin1 2
#define motor2Pin2 3
#define motor3Pin1 4
#define motor3Pin2 5
#define motor4Pin1 6
#define motor4Pin2 7

#define SERVO 25
#define BUZZER 22
#define LED 23

#define     MAX_SERVOS 1

// Variáveis e Funções Globais
static int pinMap     [MAX_SERVOS] ; // P/ mapear os pinos PWM
static int pulseWidth [MAX_SERVOS] ; // P/ definir largura de pulso
static PI_THREAD (softServoThread); // Função para o PWM

// Protótipos das Funções
void softServoWrite (int servoPin, int value); // Função que define a angulação do
servomotor
int softServoSetup (int p0, int p1, int p2, int p3, int p4, int p5, int p6, int
p7); // Setup para o servomotor
int getch(void); // Função para pegar a entrada sem precisar do enter

void *function_camera()
{
      struct sched_param sp;
      sp.sched_priority = 80; // Dá prioridade 80 p/ essa thread

      if(pthread_setschedparam(pthread_self(), SCHED_FIFO, &sp))
            fprintf(stderr,"WARNING: Failed to set stepper thread to real-time
priority.\n");

      // Comando da PiCâmera: Realiza o Streamming
      system("raspivid -n -o - -t 0 -w 600 -h 400 -fps 23 |cvlc -vvv
stream:///dev/stdin --sout '#standard{access=http,mux=ts,dst=:8090}' :demux=h264");

      return NULL;
}

int main(int argc, char *argv[])
{
      if (wiringPiSetup() == -1)
      {
            fprintf (stdout, "oops: %s\n", strerror (errno)) ;
            exit(-1);
      }

      pthread_t camera; // Define Thread da Câmera
      pthread_create(&camera, NULL, function_camera, NULL); // Lança Thread da Câmera
```

```
	// Configura pino SERVO como PWM
	softServoSetup(SERVO, -1, -1, -1, -1, -1, -1, -1);

	// Configura os seguintes pinos com SAÍDA
	pinMode(motor1Pin1, OUTPUT);
	pinMode(motor1Pin2, OUTPUT);
	pinMode(motor2Pin1, OUTPUT);
	pinMode(motor2Pin2, OUTPUT);
	pinMode(motor3Pin1, OUTPUT);
	pinMode(motor3Pin2, OUTPUT);
	pinMode(motor4Pin1, OUTPUT);
	pinMode(motor4Pin2, OUTPUT);

	pinMode(LED, OUTPUT);
	pinMode(BUZZER, OUTPUT);

	printf("\n\t>> PROGRAMA DE CONTROLE DO ROBÔ\n\n");

	printf("Digite as letras de controle:\n");
	printf("\t W: p/ frente\n");
	printf("\t A: p/ esquerda\n");
	printf("\t S: p/ trás\n");
	printf("\t D: p/ direita\n");
	printf("\t L: p/ ON/OFF LEDs\n");
	printf("\t B: p/ BUZZER\n");
	printf("\t C: p/ AUMENTAR SERVO\n");
	printf("\t V: p/ DIMINUIR SERVO\n\n");

	printf("Para SAIR, pressione CTRL+C\n\n");

	struct sched_param sp;
	sp.sched_priority = 99; // Dá prioridade 99 p/ essa thread

	if(pthread_setschedparam(pthread_self(), SCHED_FIFO, &sp))
		fprintf(stderr,"WARNING: Failed to set stepper thread to real-time
priority.\n");

	char ch, ch_t;
	int lantern = 0, buzzer = 0, value_servo = 0;

	while(1)
	{
		ch = getch(); // Para pegar a entrada do usuário

		if (ch == 'w') // Ir para frente
		{
			digitalWrite(motor1Pin1, LOW);
			digitalWrite(motor1Pin2, HIGH);
			digitalWrite(motor2Pin1, LOW);
			digitalWrite(motor2Pin2, HIGH);
			digitalWrite(motor3Pin1, LOW);
			digitalWrite(motor3Pin2, HIGH);
			digitalWrite(motor4Pin1, LOW);
			digitalWrite(motor4Pin2, HIGH);
			delay(25);
```

```
}
else if (ch == 's') // Ir para trás
{
      digitalWrite(motor1Pin1, HIGH);
      digitalWrite(motor1Pin2, LOW);
      digitalWrite(motor2Pin1, HIGH);
      digitalWrite(motor2Pin2, LOW);
      digitalWrite(motor3Pin1, HIGH);
      digitalWrite(motor3Pin2, LOW);
      digitalWrite(motor4Pin1, HIGH);
      digitalWrite(motor4Pin2, LOW);
      delay(25);
}
else if (ch == 'a') // Ir para esquerda
{
      digitalWrite(motor1Pin1, HIGH);
      digitalWrite(motor1Pin2, LOW);
      digitalWrite(motor2Pin1, HIGH);
      digitalWrite(motor2Pin2, LOW);
      digitalWrite(motor3Pin1, LOW);
      digitalWrite(motor3Pin2, HIGH);
      digitalWrite(motor4Pin1, LOW);
      digitalWrite(motor4Pin2, HIGH);
      delay(25);
}
else if (ch == 'd') // Ir para direita
{
      digitalWrite(motor1Pin1, LOW);
      digitalWrite(motor1Pin2, HIGH);
      digitalWrite(motor2Pin1, LOW);
      digitalWrite(motor2Pin2, HIGH);
      digitalWrite(motor3Pin1, HIGH);
      digitalWrite(motor3Pin2, LOW);
      digitalWrite(motor4Pin1, HIGH);
      digitalWrite(motor4Pin2, LOW);
      delay(25);
}
else if (ch == 'l') // On/Off LEDs
{
      if (lantern == 0){
            digitalWrite(LED, HIGH);
            lantern = 1;
      }else{
            digitalWrite(LED, LOW);
            lantern = 0;
      }
}
else if (ch == 'b') // BUZZER
{
      digitalWrite(BUZZER, HIGH);
      delay(200);
      digitalWrite(BUZZER, LOW);
}
else if (ch == 'c') // AUMENTA SERVO
{
```

```
                value_servo += 100;

                if (value_servo < -250)
                        value_servo = -250;
                else if (value_servo > 1250)
                        value_servo = 1250;

                softServoWrite (SERVO, value_servo);
        }
        else if (ch == 'v') // DIMINUI SERVO
        {
                value_servo -= 100;

                if (value_servo < -250)
                        value_servo = -250;
                else if (value_servo > 1250)
                        value_servo = 1250;

                softServoWrite (SERVO, value_servo);
        }

        // Desliga os motores
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, LOW);
        digitalWrite(motor2Pin1, LOW);
        digitalWrite(motor2Pin2, LOW);
        digitalWrite(motor3Pin1, LOW);
        digitalWrite(motor3Pin2, LOW);
        digitalWrite(motor4Pin1, LOW);
        digitalWrite(motor4Pin2, LOW);
    }

    return 0;
}

/*********************************************************
*************** BIBLIOTECAS & FUNÇÕES ******************
*********************************************************/

int getch(void)
{
    struct termios oldattr, newattr;
    int ch;
    tcgetattr( STDIN_FILENO, &oldattr );
    newattr = oldattr;
    newattr.c_lflag &= ~( ICANON | ECHO );
    tcsetattr( STDIN_FILENO, TCSANOW, &newattr );
    ch = getchar();
    tcsetattr( STDIN_FILENO, TCSANOW, &oldattr );
    return ch;
}


static PI_THREAD(softServoThread)
{
```

```
	register int i, j, k, m, tmp;
	int lastDelay, pin, servo;

	int myDelays [MAX_SERVOS];
	int myPins   [MAX_SERVOS];

	struct timeval  tNow, tStart, tPeriod, tGap, tTotal;
	struct timespec tNs;

	tTotal.tv_sec  =    0;
	tTotal.tv_usec = 8000;

	piHiPri(50);

	for(;;)
	{
		gettimeofday (&tStart, NULL);

		memcpy (myDelays, pulseWidth, sizeof (myDelays));
		memcpy (myPins,   pinMap,     sizeof (myPins));

		// Sort the delays (& pins), shortest first

		for (m = MAX_SERVOS / 2 ; m > 0 ; m /= 2 )
			for (j = m ; j < MAX_SERVOS ; ++j)
		for (i = j - m ; i >= 0 ; i -= m)
		{
			k = i + m ;
			if (myDelays [k] >= myDelays [i])
				break;

			else // Swap
			{
				tmp = myDelays [i] ; myDelays [i] = myDelays [k] ; myDelays
[k] = tmp ;
				tmp = myPins    [i] ; myPins    [i] = myPins    [k] ; myPins
[k] = tmp ;
			}
		}

		// All on

		lastDelay = 0 ;
		for (servo = 0 ; servo < MAX_SERVOS ; ++servo)
		{
			if ((pin = myPins [servo]) == -1)
				continue;

			digitalWrite(pin, HIGH);
			myDelays [servo] = myDelays [servo] - lastDelay;
			lastDelay += myDelays [servo];
		}

		// Now loop, turning them all off as required
```

```
            for(servo = 0 ; servo < MAX_SERVOS ; ++servo)
            {
                  if ((pin = myPins [servo]) == -1)
                        continue ;

                  delayMicroseconds (myDelays [servo]) ;
                  digitalWrite (pin, LOW) ;
            }

            // Wait until the end of an 8mS time-slot

            gettimeofday (&tNow, NULL);
            timersub (&tNow, &tStart, &tPeriod);
            timersub (&tTotal, &tPeriod, &tGap);
            tNs.tv_sec  = tGap.tv_sec;
            tNs.tv_nsec = tGap.tv_usec * 1000;
            nanosleep (&tNs, NULL);
      }

      return NULL;
}

void softServoWrite (int servoPin, int value)
{
      int servo ;
      servoPin &= 63 ;

      if (value < -250)
            value = -250;
      else if (value > 1250)
            value = 1250;

      for (servo = 0 ; servo < MAX_SERVOS ; ++servo)
            if (pinMap [servo] == servoPin)
                  pulseWidth [servo] = value + 1000 ; // uS
}

int softServoSetup (int p0, int p1, int p2, int p3, int p4, int p5, int p6, int p7)
{
      int servo ;

      if (p0 != -1) { pinMode (p0, OUTPUT) ; digitalWrite (p0, LOW) ; }
      if (p1 != -1) { pinMode (p1, OUTPUT) ; digitalWrite (p1, LOW) ; }
      if (p2 != -1) { pinMode (p2, OUTPUT) ; digitalWrite (p2, LOW) ; }
      if (p3 != -1) { pinMode (p3, OUTPUT) ; digitalWrite (p3, LOW) ; }
      if (p4 != -1) { pinMode (p4, OUTPUT) ; digitalWrite (p4, LOW) ; }
      if (p5 != -1) { pinMode (p5, OUTPUT) ; digitalWrite (p5, LOW) ; }
      if (p6 != -1) { pinMode (p6, OUTPUT) ; digitalWrite (p6, LOW) ; }
      if (p7 != -1) { pinMode (p7, OUTPUT) ; digitalWrite (p7, LOW) ; }

      pinMap [0] = p0 ;
      pinMap [1] = p1 ;
      pinMap [2] = p2 ;
      pinMap [3] = p3 ;
      pinMap [4] = p4 ;
```

```
	pinMap [5] = p5 ;
	pinMap [6] = p6 ;
	pinMap [7] = p7 ;

	for (servo = 0 ; servo < MAX_SERVOS ; ++servo)
		pulseWidth [servo] = 1500 ;		// Mid point

	return piThreadCreate (softServoThread) ;
}
```