ID:112203

Desenvolvimento de Seguidor de Linha autônomo com refinamento de PID remoto

São José dos Campos - Brasil

Maio de 2019

ID:112203

Desenvolvimento de Seguidor de Linha autônomo com refinamento de PID remoto

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina Sistemas Embarcados.

Docente: Prof. Dr. Sérgio Ronaldo Barros

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Maio de 2019

Lista de ilustrações

Lista de tabelas

Sumário

1 Descrição do Projeto

Seguidor de linha é uma máquina que pode seguir um caminho, que pode ser determinado linha preta em uma superfície branca, assim a máquina deve possuir um sistema de *feedback* do ambiente (caminho) para manobrar e corrigir seus movimentos autonomamente.

Exemplos de aplicações de um robô seguidor de linha são carros automatizados rodando em estradas com ímãs incorporados, sistema de orientação para robôs industriais que se deslocam no chão de fábrica e robôs de competição.

A proposta de sistema de controle para execução dessa tarefa implementa um algoritmo PID (*Proportional Integrative Derivative*) que proporciona uma resposta a um erro calculado a partir do *feedback* do ambiente a partir da equação característica do algoritmo.

$$PID = K_p * e + K_i * \int de + K_d * de$$

Um grande desafio no desenvolvimento de um sistema de controle de desse tipo é o refinamento das constantes K_p, K_i e K_d que para sistemas pequenos é basicamente feita com experimentação, ou seja, definição manual e teste no circuito. Portanto, como grande diferencial desse projeto, se pretende implementar um método remoto para definir e alterar essas constantes, tornando o refinamento do controle menos trabalhoso e versátil.

2 Objetivos

2.1 Geral

Desenvolver um robô seguidor de linha autônomo controlado por algoritmo de controle PID e com comunicação remota para transmissão e recepção de dados.

2.2 Específico

Captar sinal de sensores para estimar a posição e o erro da trajetória do robô. Efetuar estimativa de erro e cálculo de PID para algoritmo de controle. Produzir resposta em sinal PWM para controle dos motores, sendo assim, da velocidade e direção do robô. Comunicar via sinal *bluetooth* com Kit PICGenius (plataforma Microgenius ©2019) para receber valores das constantes do algoritmo de PID refinando o comportamento do controle remotamente.

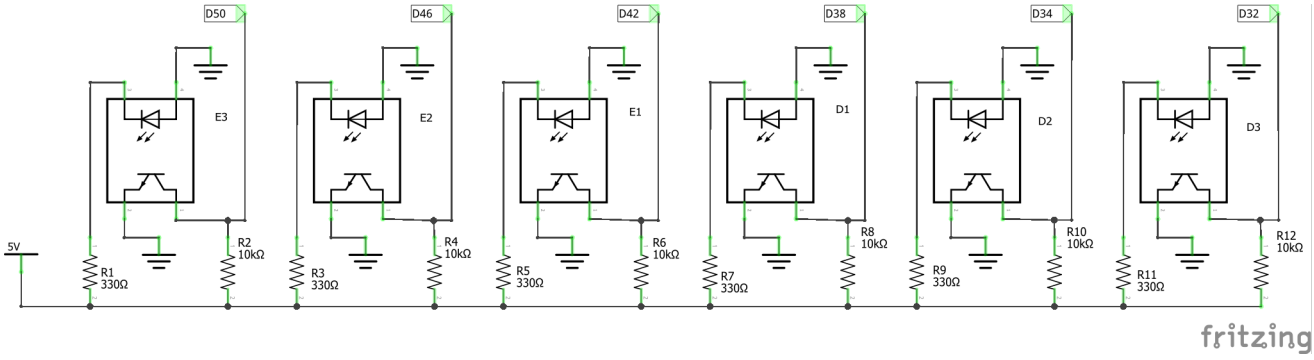
3 Funcionamento do Hardware

3.1 Sensor óptico reflexivo TCRT-5000 e cabeça de leitura

O TCRT5000 possui acoplado no mesmo dispositivo um sensor infravermelho (emissor) e um fototransistor (receptor) de faixa de operação de até 5V. O receptor é sensível a luminosidade refletida do emissor, que possui uma faixa de frequência de luz infravermelho, dessa forma os elétrons gerados pelos fótons incidem na base do transistor, e sua corrente de saída é amplificada pela sua operação (1).

Utilizando resistores para regular a corrente, 330 Ohms para os emissores e 10K Ohms para os receptores, foi produzida uma cabeça de leitura captar sinais da faixa que o robô deve seguir como mostra na Figura 1

Figura 1 – Esquemático da cabeça de leitura



Fonte: O Autor

O sinal coletado no circuito (representado nas *labels* Dxx da Figura 1), devido ao funcionamento do TCRT5000, será maior para superfícies menos reflexivas (faixa preta) e mais baixo para as mais reflexivas (pista de cor clara), portanto, possibilitando a determinação da posição da faixa em relação ao robô.Essa resposta pode ser coletada tanto em sinal digital quanto analógico, entretanto devido a aplicação do sensor optou-se por uma leitura digital da saída desses sensores.

3.2 Módulo de ponte H L298N e motores

O módulo de ponte H L298N é um módulo de controle de motor DC de dois canais que permite controlar velocidade e sentido de rotação de até dois motores. Suas entradas são: In1, In2, In3 e In4 para controlar, em pares, a direção de rotação do motor; Ena e Enb para controlar a velocidade; Vcc para alimentação dos motores com tensão de operação

7 35V; 5v para alimentação dos níveis lógicos do módulo com 5V; Gnd para conectar o terra da alimentação dos motores e do nível lógico. As saídas são OUT1, OUT2, OUT3 e OUT4 que fornecem a tensão DC para os motores, se um sinal PWM for aplicado em Ena e Enb respostas nessas saídas também será um sinal PWM (2).

Os motores utilizados juntamente com a ponte H foram 2 motores Pololu. Esses pequenos motores possuem diversas versões com diferentes caixas de redução, diferentes voltagens e escovas. Nesse projeto foram utilizados os que possuem redução de 100:1 (lê-se cem para um), voltagem de operação de 6V e escovas de carbono.

O módulo de ponte H com os motores 100:1 atendem as necessidades do projeto, o controle através do módulo consiste em aplicar sinais digitais nas entradas In1, In2, In3 e In4 para determinar a direção de rotação dos motores e sinais PWM em Ena e Enb para determinar a velocidade dos mesmos.

3.3 Módulo *bluetooth* HC-05 e HC-06

Estes módulos oferecem uma forma fácil de comunicação, sendo que o HC-05 pode ser configurado como modo mestre ou escravo, já o HC-06 opera somente em modo escravo. Em sua placa existe um regulador de tensão possibilitando uma tensão de operação 3.3V à 5V, bem como um LED indicador que pisca intermitentemente se está esperando uma conexão e pisca duas vezes a cada segundo quando conectado, seu alcance de comunicação pode chegar a até 10m de distância. A comunicação do microcontrolador com esse dispositivo acontece por protocolo USART por padrão do microcontrolador, porém esse módulo também utiliza padrão UART (3). A configuração de módulos *bluetooth* HC-05 é efetuada utilizando comandos AT, uma breve explicação sobre como isso foi feito para esse trabalho se encontra no Apêndice A.

3.4 Baterias

Em um robô seguidor de linha existe a necessidade de separar o fornecimento de energia entre os motores e o microcontrolador, pois os motores quando em funcionamento causam grandes oscilações na corrente do circuito de alimentação, isso torna o funcionamento do microcontrolador imprevisível, visto que um seguidor de linha depende do funcionamento correto do microcontrolador.

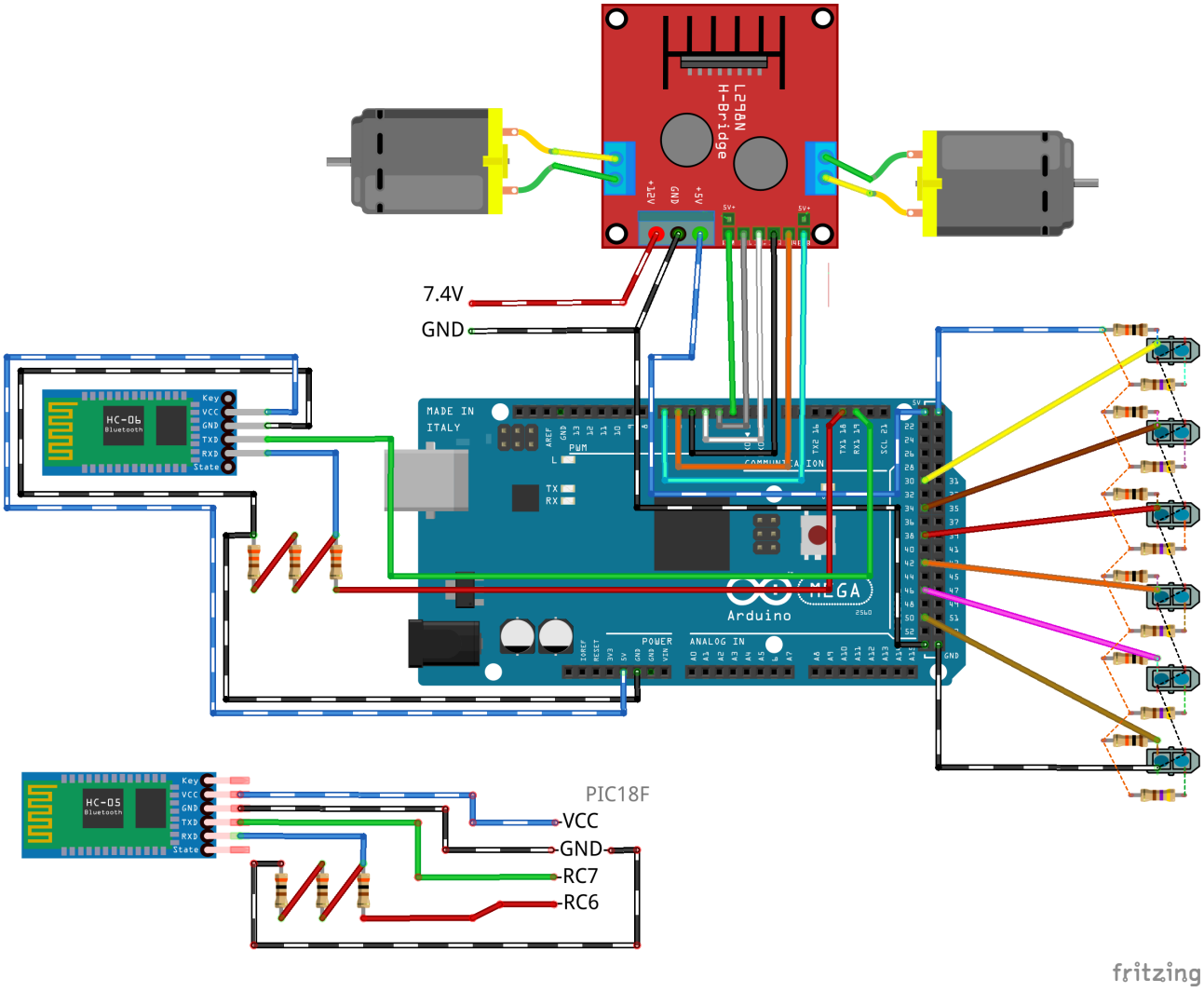
Portanto para alimentação dos motores foram utilizadas células li-íon 18650-22H de tensão nominal de 3.6V, ligando duas em série se obtém 7.2V DC nominal, um pouco acima da capacidade dos motores, entretanto ocorre uma pequena queda de tensão na ponte H proporcionando funcionamento dos motores sem prejudicar sua vida útil. Essas células, nessa associação, desempenham 2200mAh proporcionando um longo tempo de

uso para um seguidor de linha. Para a alimentação do microcontrolador foi utilizado uma bateria convencional zinco-carbono de 9V.

3.5 Esquemático do sistema elétrico e conexões dos pinos

A Figura 2 contém o esquemático do sistema elétrico do robô seguidor de linha. Nela é possível observar a forma como os componentes estão conectados com os microcontroladores.

Figura 2 – Esquemático do circuito completo



Fonte: O Autor

Note que as portas usadas para cada componente na placa Arduíno são:

- As portas 50, 46, 42, 38, 34 e 30 do Arduíno configuradas como entradas digitais estão dedicadas aos sensores da cabeça de leitura, portanto assumem nível lógico

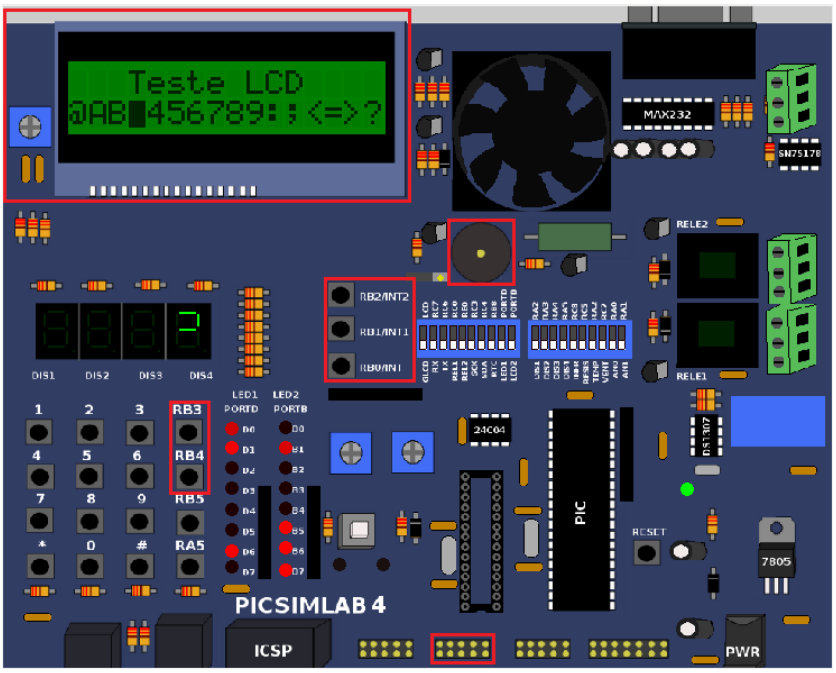
alto a partir de certo nível de sinal dos sensores e baixo quando esse sinão não atinge esse nível.

- As portas 2 e 7 do Arduíno configuradas como saídas PWM, dedicadas ao controle de velocidade dos motores DC, conectadas nos pinos ENA e ENB respectivamente do módulo de ponte H.
- As portas 3, 4, 5 e 6 do Arduíno configuradas como saídas digitais, dedicadas ao controle da direção de rotação dos motores DC, conectadas nos pinos IN1, IN2, IN3 e IN4 respectivamente do módulo de ponte H.
- As portas 18 e 19 do Arduíno configuradas como portas serias de comunicação TX e RX, dedicadas a comunicação USART via sinal bluetooth, conectadas aos pinos RXD e TXD respectivamente do módulo bluetooth HC-06 configurado como escravo.
- Por último as portas RC6 e RC7 do Kit PICGenius configuradas como portas seriais de comunicação TX e RX, dedicadas a comunicação USART via sinal bluetooth, conectadas aos pinos RXD e TXD respectivamente do módulo bluetooth HC-05 configurado como mestre.

3.6 Componentes utilizados do Kit PICGenius

O kit acompanha o PIC18F4520 que possui diversos recursos, dentre eles: 13 canais AD, UART, SPI. O kits de desenvolvimento possui diversos recursos integrados e suporta mais de 300 modelos de microcontroladores PIC, Aplicações complexas podem ser implementadas e testadas fácilmente usando Linguagem ASM, C, BASIC ou Pascal (4).

Figura 3 – Ilustração alusiva ao Kit PICGenius



Fonte: PIC Simulator Laboratory (5)

Na Figura 3 está destacado em vermelho os componentes utilizados para integração no projeto. Para o d*diplay* LCD são utilizados os pinos RD4, RD5, RD6, RD7, RE1 e RE2 do microcontrolador enviando os bits dos caracteres, para os botões são utilizados os pinos RB0, RB1, RB2, RB3 e RB4 recebendo sinais que indicam o estado dos botões, para o buzzer é utilizado o pino RC1 enviando o sinal que modifica o estado entre ligado e desligado desse componente, por fim os pinos RC6 e RC7 para a comunicação bluetooth como descrito na seção anterior.

4 Software da plataforma Arduíno

Para controle do seguidor de linha foi desenvolvido um software baseado em algoritmo de controle PID. O software contém uma forma simplificada e intuitiva de efetuar o controle do robô baseando-se na determinação do erro pela resposta do circuito da cabeça de leitura.

4.1 Leitura dos sensores

Segue o código referente a leitura dos sensores:

```
1  /***************/
2  /*Sensor Header*/
3  /***************/
4  //Sensor header pins
5  #define E3 50
6  #define E2 46
7  #define E1 42
8  #define D1 38
9  #define D2 34
10 #define D3 30
11 //Sensor header atributes
12 float error;
13 //Sensor header functions
14 int updateError(){
15   error=0;
16   float avg=0;
17   if(digitalRead(E3)) {error=error-2.5;avg+=1; }
18   if(digitalRead(E2)) {error=error-1.5;avg+=1; }
19   if(digitalRead(E1)) {error=error-0.5;avg+=1; }
20   if(digitalRead(D1)) {error=error+0.5+1;avg+=1; }
21   if(digitalRead(D2)) {error=error+1.5+1;avg+=1; }
22   if(digitalRead(D3)) {error=error+2.5+1;avg+=1; }
23   error=(!avg)? 0:error/avg;
24   return (avg>=5)? 1:0;
25 }
```

A variável global *error* representa o erro estimado entre o centro da cabeça de leitura e a posição da faixa. Esse erro é estimado nessa função *updateError()* em centímetros a partir das respostas dos sensores que, baseado em sua posição na cabeça de leitura, acrescentam um valor ao erro total (linhas 17-22) produzindo, então, uma média entre o valor total e a quantidade de sensores que emitiram sinal alto (linha 23), o retorno dessa função representa o caso em que o seguidor de linha passa pela linha de partida/chegada para demarcação de voltas (linha 24).

4.2 Controle dos motores

Segue o código referente ao controle dos motores:

```
1  /**************/
2  //Motor driver pins
3  #define ENL   7
4  #define INL1  5
5  #define INL2  6
6  #define ENR   2
7  #define INR1  3
8  #define INR2  4
9  //Motor driver atributes
10 int LMSpeed, RMSpeed;
11 byte spConst = 80;
12 byte turnSpeed = 80;
13 byte rOffset;
14 byte lOffset;
15 bool startDrive=false;
16 //Motor driver functions
17 void drive(int cor){
18   LMSpeed = spConst + cor;
19   RMSpeed = spConst - cor;
20
21   if(LMSpeed < 0) {LMSpeed = 0;}
22   if(LMSpeed > 255) {LMSpeed = 255;}
23
24   if(RMSpeed < 0) {RMSpeed = 0;}
25   if(RMSpeed > 255) {RMSpeed = 255;}
26
27   analogWrite(ENL,LMSpeed);
28   analogWrite(ENR,RMSpeed);
29   digitalWrite(INL1,LOW);
30   digitalWrite(INL2,HIGH);
31   digitalWrite(INR1,HIGH);
32   digitalWrite(INR2,LOW);
33 }
34 void neutralMove(){
35   analogWrite(ENL,0);
36   analogWrite(ENR,0);
37   digitalWrite(INL1,LOW);
38   digitalWrite(INL2,LOW);
39   digitalWrite(INR1,LOW);
40   digitalWrite(INR2,LOW);
41 }
```

A função *neutralMove()* (linha 34) para o funcionamento dos motores, fornecendo sinais PWM com *duty cycle* 0 para as velocidades dos motores e sinais baixos para o controle da direção, permite que os eixos dos motores estejam livres para girar, algo semelhante ao ponto morto de sistemas automotivos.

A função *drive()* (linha 17) recebe a variável inteira (*corner*) que é utilizada para efetuar curvas a direita se positiva e a esquerda se negativa (linhas 18 e 19), nas linhas 21 à 25 é limitado o valor da velocidade de cada motor segundo o range do sinal PWM produzido pelo microcontrolador e por fim nas linhas 27 à 32 é efetivamente aplicado os sinais que efetuam o controle dos motores.

4.3 Comunicação *bluetooth*

Segue o código da comunicação *bluetooth*

```
1  /*****************/
2  /*Bluetooth HC-05*/
3  /*****************/
4  //Bluetooth atrinutes
5  bool isMessageNew=0;
6  String command;
7  //Bluetooth functions
8  void listenMessage() {
9    if (Serial1.available())
10   {
11     while(Serial1.available()) {
12       delay(10);
13       command += (char)Serial1.read();
14     }
15     isMessageNew=true;
16     //Serial1.println(command);
17     //Serial.println(command);
18   }
19 }
```

A função *listenMessage()* (linha 8) checa se há algum dado enviado do dispositivo móvel, quando isso é verdadeiro, salva essa informação na variável global *command* e retorna para o dispositivo a mensagem recebida.

Segue outra parte referente a mesma funcionalidade:

```
1  void parseMessage(){
2    char op[]={command[0],command[1]};
3    float num;
4    command.remove(0,2);
5    num = command.toFloat();
6    switch(op[0]){
7      case 'k':
8        switch(op[1]){
9          case 'p': Kp=num; break;
10         case 'i': Ki=num; break;
11         case 'd': Kd=num; break;
12         default:
13           Serial1.println("\nUnknown cmd\n");
14         break;
15       }
16     break;
17       case 'm':
18       switch(op[1]){
19         case 'r': rOffset=num; break;
20         case 'l': lOffset=num; break;
21         case 't': spConst=num; break;
22         default:
23           Serial1.println("\nUnknown cmd\n");
24         break;
25       }
26     break;
27     case 'g': startDrive = true; break;
28     case 's': startDrive = false; break;
29     default:
```

```
30        if(!command[0]=='e' && !command[1]=='c')
31          Serial1.println("\nUnknown cmd\n");
32      break;
33    }
34    command="";
35    isMessageNew=false;
36  }
```

Na função *parseMessage()* a variável global *command* sofre um tratamento dividindo-a em um comando de 2 caracteres e um valor numérico, o comando chaveia as alternativas para atualizar alguma constante numérica do algoritmo de PID, da velocidade base dos motores ou comandos de andar ou parar para o seguidor de linha.

4.4 Calculo de PID e controle

Segue o código da função *loop()* e de cálculo de PID:

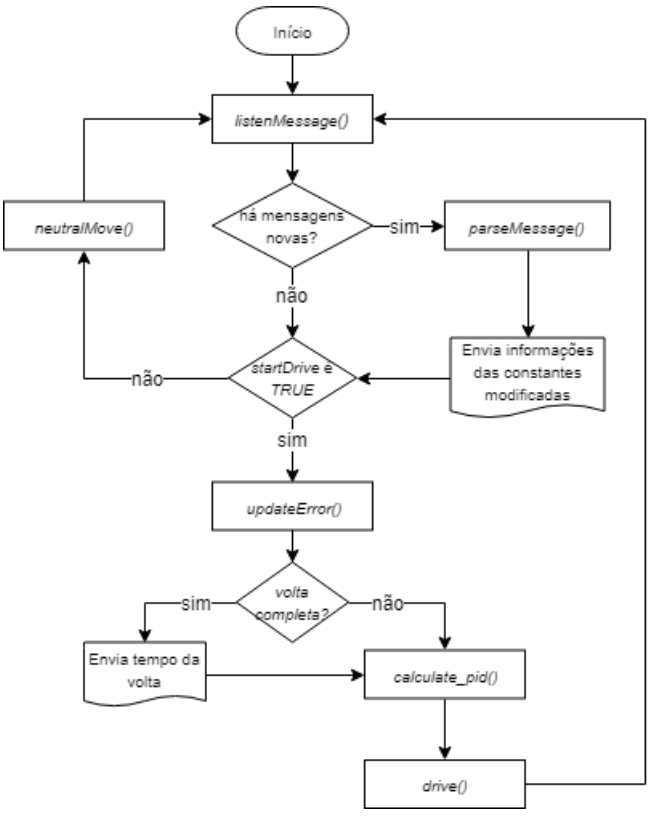
```
1   void loop() {
2     listenMessage();
3     if(isMessageNew){
4         parseMessage();
5         //Serial1.print("state: ");
6         //Serial1.println(startDrive);
7         //Serial1.print("Kp: ");
8         //Serial1.print(Kp,4);
9         //Serial1.print(" Ki: ");
10        //Serial1.print(Ki,4);
11        //Serial1.print(" Kd: ");
12        //Serial1.print(Kd,4);
13        //Serial1.print(" Speed:");
14        //Serial1.println(spConst);
15    }
16    if(startDrive){
17      if(updateError()){
18        Serial1.print("L ");
19        //Serial1.print("Lap: ");
20        //Serial1.println((float)(millis()-lapTime)/1000,3);
21        while(updateError());
22        //lapTime=millis();
23      }
24      calculate_pid();
25
26      drive((int)PID_value);
27
28    }else{
29      neutralMove();
30    }
31  }
32
33  void calculate_pid()
34  {
35
36    P = error;
37    I = I + error*0.001;
38    D = (error - previous_error)/.001;
39
```

```
40    PID_value = (Kp*P) + (Ki*I) + (Kd*D);
41
42    previous_error=error;
43  }
```

Na função *loop()* temos o funcionamento do segue faixa como um todo. Inicialmente chama a função *listenMessage()* para efetuar comunicação com o remota, caso haja mensagem o comando recebido é tratado pela função *parseMessage()*. Então caso a variável global startDrive possua valor *TRUE* (linha 16), representando que o segue faixa está em modo de direção, do contrário ele estará em modo parado chamando a função *neutralMove()* (linha 29), ele atualizará o erro de trajetória com a função *updateError()* (linha 17) bem como determinar se uma volta foi efetuada emitindo uma mensagem via comunicação serial *bluetooth*. Por fim a função *calculate_pid()*é chamada (linha 24), cujas instruções estão nas linhas 33 à 43, determinado um valor para a variável global *PID_value*, que aplicada a chamada de função *drive()* (linha 26) conduz o robô a fazer curvas, corrigir trajetória e se manter seguindo a linha.

Segue um fluxograma do algoritmo de controle, que demonstra o comportamento do robô já explicado, mas em forma gráfica facilitando o entendimento na Figura 4.

Figura 4 – Fluxograma do algoritmo



Fonte: O Autor

5 Software da plataforma PICGenius

5.1 Configurações iniciais

Segue o código referente às configurações para o *display* LCD:

```
// Selecionando pinos utilizados para comunicacao com display LCD
sbit LCD_RS at RE2_bit;// PINO 1 DO PORTE INTERLIGADO AO RS DO DISPLAY
sbit LCD_EN at RE1_bit;// PINO 3 DO PORTD INTERLIGADO AO EN DO DISPLAY
sbit LCD_D7 at RD7_bit;// PINO 7 DO PORTD INTERLIGADO AO D7 DO DISPLAY
sbit LCD_D6 at RD6_bit;// PINO 6 DO PORTD INTERLIGADO AO D6 DO DISPLAY
sbit LCD_D5 at RD5_bit;// PINO 5 DO PORTD INTERLIGADO AO D5 DO DISPLAY
sbit LCD_D4 at RD4_bit;// PINO 4 DO PORTD INTERLIGADO AO D4 DO DISPLAY

// Selecionando direcao de fluxo de dados dos pinos utilizados para a comunicacao com
    display LCD
sbit LCD_RS_Direction at TRISE2_bit;// DIRECAO DO FLUXO DE DADOS DO PINO 2 DO PORTD
sbit LCD_EN_Direction at TRISE1_bit;// DIRECAO DO FLUXO DE DADOS DO PINO 3 DO PORTD
sbit LCD_D7_Direction at TRISD7_bit;// DIRECAO DO FLUXO DE DADOS DO PINO 7 DO PORTD
sbit LCD_D6_Direction at TRISD6_bit;// DIRECAO DO FLUXO DE DADOS DO PINO 6 DO PORTD
sbit LCD_D5_Direction at TRISD5_bit;// DIRECAO DO FLUXO DE DADOS DO PINO 5 DO PORTD
sbit LCD_D4_Direction at TRISD4_bit;// DIRECAO DO FLUXO DE DADOS DO PINO 4 DO PORTD
```

Como mostra o código está a definição dos pinos para envio dos bits dos caracteres do display. Segue o código referete às configurações dos registradores de interrupção e demais componentes:

```
void main()
{
    //TIMER INTERRUPTION CONFIG
    T0CON.TMR0ON = 1;        // Liga timer0
    T0CON.T08BIT = 0;        // Define contagem no modo 16 bits.
    T0CON.T0CS = 0;          // Timer0 operando como temporizador.
    T0CON.T0SE = 0;          // Contagem borda de decida
    T0CON.PSA = 0;           // Prescaler ativado.
    T0CON.T0PS2 = 0;         // Define prescaler 1:2
    T0CON.T0PS1 = 0;         // Define prescaler 1:2
    T0CON.T0PS0 = 0;         // Define prescaler 1:2
    // Valor para 1 milesimo.
    TMR0H = 0xFC;            // 64536 high bits
    TMR0L = 0x18;            // 64536 low bits

    INTCON.TMR0IE = 1;       // Habilita interrupcao do timer0.
    INTCON.TMR0IF = 0;       // Apaga flag de estouro do timer0
    INTCON.GIE = 1;          // Habilita as interrupcoes nao-mascaradas.
    INTCON.PEIE = 1;         // Habilita as interrupcoes dos perifericos.


    //LCD CONFIG
    ADCON1=0x0E;        //Configura pinos do PORTB como digitais e RA0 (PORTA) como
        analogico
    Lcd_Init();         //INICIALIZA DISPLAY LCD
    Lcd_Cmd(_LCD_CURSOR_OFF);//Apaga cursor
    Lcd_Cmd(_LCD_CLEAR);//ENVIA O COMANDO DE LIMPAR TELA PARA O DISPLAY LCD
```

```
27
28      //SERIAL BLUETOOTH CONFIG
29      UART1_Init(9600);
30      Delay_ms(1000);
31
32      //BUZZER CONFIG
33      TRISC.RC1 = 0;
34
35      //PUSH BUTTOM CONFIG
36      TRISB.RB0=1;
37      TRISB.RB1=1;
38      TRISB.RB2=1;
39      TRISB.RB3=1;
40      TRISB.RB4=1;
```

As primeiras configurações são referentes ao registrador TIMER0 definindo uma contagem em mili-segundos uma vez que o *prescaler* está definido 1:2, o modo para 16 bits e o valor de contagem como 64536, tal como é demonstrado na equação a seguir:

$$1000us = 0,5us * 2 * (65536 - 64536)$$

As interrupções são habilitadas no registrador INTCON (linhas 18 e 19) e definidas para o estouro de contagem do TIMER0 na liha 16. As linhas de condigo seguintes iniciam a utilização do periférico LCD linhas, da comunicação UART via *bluetooth* (linha 29), do buzzer (linha 33) e dos *push buttons* (linha 36 à 40).

5.2 Laço principal do programa

Segue o laço principal do programa que lida com as ações relacionadas ao acionamento dos *push buttons*, recebimento de mensagens via comunicação serial *bluetooth*, o controle do *display* LCD e o acionamento do buzzer:

```
1       while(1)
2       {
3            //Comunicacao Serial bluetooth
4            if(UART1_Data_Ready()){
5               UART1_Read_Text(serial_input," ",16);
6               //UART1_Write_Text(serial_input);
7               if(serial_input[0]=='L'){
8                  miliseconds=milis;
9                  milis=0;
10                 setLapTime();
11                 lcd_Out(2,1,full_line);
12                 PORTC.RC1 = ~PORTC.RC1; //inversao de estado
13                 delay_ms(100);          //delay
14                 PORTC.RC1 = ~PORTC.RC1; //inversao de estado
15                 delay_ms(100);          //delay
16                 PORTC.RC1 = ~PORTC.RC1; //inversao de estado
17                 delay_ms(100);          //delay
18                 PORTC.RC1 = ~PORTC.RC1; //inversao de estado
19              }
20           }
21           //Acionamento dos push buttons
```

```
22          //botao ligado ao RB0
23          if((PORTB.RB0==0)&&(uc_RB0==0)){
24             uc_RB0=1;
25          }
26          if((PORTB.RB0==1)&&(uc_RB0==1)){
27             uc_RB0=0;
28             param=param-input_number;
29             input_number=(input_number+1)%10;
30             param=param+input_number;
31          }
32          //botao ligado ao RB1
33          if((PORTB.RB1==0)&&(uc_RB1==0)){
34             uc_RB1=1;
35          }
36          if((PORTB.RB1==1)&&(uc_RB1==1)){
37             uc_RB1=0;
38             param=param*10;
39             input_number=0;
40          }
41          //botao ligado ao RB2
42          if((PORTB.RB2==0)&&(uc_RB2==0)){
43             uc_RB2=1;
44          }
45          if((PORTB.RB2==1)&&(uc_RB2==1)){
46             uc_RB2=0;
47             param=0;
48             input_number=0;
49             cmd_state=(cmd_state+1)%4;
50             switch(cmd_state){
51                case 0:lcd_Out(1,1,"Speed:");
52                       prev=speed;
53                       break;
54                case 1:lcd_Out(1,1,"Kp:   ");
55                       prev=kp;
56                       break;
57                case 2:lcd_Out(1,1,"Ki:   ");
58                       prev=ki;
59                       break;
60                case 3:lcd_Out(1,1,"Kd:   ");
61                       prev=kd;
62                       break;
63                default:break;
64             }
65             setPreValue();
66          }
67          //botao ligado ao RB3
68          if((PORTB.RB3==0)&&(uc_RB3==0)){
69             uc_RB3=1;
70          }
71          if((PORTB.RB3==1)&&(uc_RB3==1)){
72             uc_RB3=0;
73             setCMD();
74             UART1_Write_Text(serial_output);
75             switch(cmd_state){
76                case 0:speed=param;break;
77                case 1:kp=param;break;
78                case 2:ki=param;break;
79                case 3:kd=param;break;
80                default:break;
81             }
```

```
82              prev=param;
83              setPreValue();
84          }
85          //botao ligado ao RB4
86          if((PORTB.RB4==0)&&(uc_RB4==0)){
87              uc_RB4=1;
88          }
89          if((PORTB.RB4==1)&&(uc_RB4==1)){
90              uc_RB4=0;
91              if(go_stop){
92                  UART1_Write_Text("s");
93                  go_stop=0;
94              }else{
95                  UART1_Write_Text("g");
96                  go_stop=1;
97              }
98          }
99          IntToStr(param,half_line);
100         lcd_Out(1,11,half_line);
101     }
102 }
```

Iniciando pelo acionamento dos *push buttons*:

- Ligado ao RB0 (linha 23 à 31): Edita os algarismos do parâmetro a ser enviado pela comunicação serial e que aparece no *display*.
- Ligado ao RB1 (linha 33 à 40): Confirma o algarismo multiplicando o parâmetro por 10, dessa forma o algarismo passa a ter ordem superior e o próximo algarismo passa a ser editável pelo botão ligado ao RB0.
- Ligado ao RB2 (linha 42 à 66): Cancela o valor editado pelos botões ligados ao RB0 e RB1 alternando o parâmetro cujo valor será editado.
- Ligado ao RB3 (linha 68 à 84): Envia o valor do parâmetro via comunicação serial *bluetooth* para o seguidor de linha (linha 67). É possível ver as chamadas de funções *setCMD()* (linha 66) e *setPreValue()* (linha 76), que efetuam formatação de strings e se encontram no Apêndice B, e a manutenção de um histórico dos últimos valores enviados (linha 68 à 74).
- Ligado ao RB4 (linha 86 à 98): Envia um único caractere via comunicação serial para o seguidor de linha controlando seu estado de movimento, seguindo linha para 'g' (linha 88) e parado para 's' (linha 85), segundo a variável *go_stop* alternante.

O recebimento das mensagens é tratado das linha 4 à 20 cuja única função é esperar o recebimento da mensagem "L "indicando que o seguidor de linha completou uma volta. Quando isso acontece a variável contador milis grava seu valor e reinicia, esse valor será tratado pela função *setLapTime()* (linha 10) e exibido no *display LCD* (linha 11), após

isso para sinalizar que uma volta foi realizada o buzzer apita duas vezes (linha 12 à 18), encerrando assim as funcionalidades do laço principal do programa.

5.3 Operações da Interrupção

Segue o código referente às operações realizadas quando uma interrupção de *timer* é gerada pelo TIMER0 ao levantar a *flag* de estouro de contagem definida para ocorrer a cada 1 mili-segundo:

```
1 void interrupt(){
2     if(INTCON.TMR0IF==1){       //quando ha overflow do timer 0.
3        // Recarrega o timer0.
4        TMR0H = 0xFC;            // 64536 high bits
5        TMR0L = 0x18;            // 64536 low bits
6        INTCON.TMR0IF = 0;       // Limpa o flag de estouro do timer0
7        milis++;
8     }
9 }
```

Apesar de simples tem papel fundamental na contagem com um bom nível de precisão do tempo das voltas efetuadas pelo seguidor de linha. Sempre que ocorre o estouro na contagem do TIMER0, representado pelo valor alto no bit do registrador INTCON.TMR0IF (linha 2), os registradores TMR0H e TMR0L são recarregados com o valor para contagem e a flag INTCON.TM0IF limpa permitindo uma nova contagem de 1 mili-segundo e por fim o mais importante a variável de contagem milis é atualizada.

6 Lista de componentes básicos

Segue na Tabela 1 a lista de componentes utilizados no projeto:

Tabela 1 – Componentes utilizados no projeto

Componente	**Quantidade**	**Função**
Arduíno Mega	1	Controle de módulos e entrada de sinais
Bateria 9V convencional	1	Fonte de alimentação do microcontrolador
Célula li-íon 18650-22H	2	Fonte de alimentação dos motores
Motor Pololu 100:1 6V	2	Proporcionar velocidade e controle de direção
Módulo *bluetooth* HC-05	1	Comunicação remota modo mestre
Módulo *bluetooth* HC-06	1	Comunicação remota modo escravo
Módulo de ponte H L298N	1	Driver para controle dos micromotores
Placa PCB perfurada (80mmx20mm)	1	Circuito da cabeça de leitura
Resistor 300 Ohms 1/4W	3	Divisor de tensão de sinal lógico para *bluetooth* mestre
Resistor 330 Ohms 1/4W	3	Divisor de tensão de sinal lógico para *bluetooth* escravo
Resistor 470 Ohms 1/4W	6	Regular corrente nos LED emissores do TCRT-5000
Resistor 10K Ohms 1/4W	6	Regular corrente nos fototransistores receptores do TCRT-5000
Roda boba	2	Suporte mecânico
Roda com pneu (até 65mm)	2	Tração e estabilidade
Sensor óptico reflexivo TCRT-5000	6	Sensor da cabeça de leitura

Fonte: O Autor

Referências

1 Vishay Semiconductors. *Reflective Optical Sensor with Transistor Output*. [S.l.], 2017. 12 p. Disponível em: <www.vishay.com>. Citado na página 7.

2 STMicroelectronics. *L298 DUAL FULL-BRIDGE DRIVER*. [S.l.], 2000. 13 p. Disponível em: <https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf>. Citado na página 8.

3 ITead Studio. *HC-05 Bluetooth to Serial Port Module*. [S.l.], 2010. 13 p. Disponível em: <http://www.electronicaestudio.com/docs/istd016A.pdf>. Citado na página 8.

4 Microgenios ©2019. *Kit PICGenios PIC18F e PIC16F Microchip – Kit Educacional c/ Gravador USB MicroICD - Microgenios*. Disponível em: <https://loja.microgenios.com.br/produto/kit-picgenios-pic18f-e-pic16f-microchip-kit-educacional-c-gravador-usb-microicd/22665>. Citado na página 10.

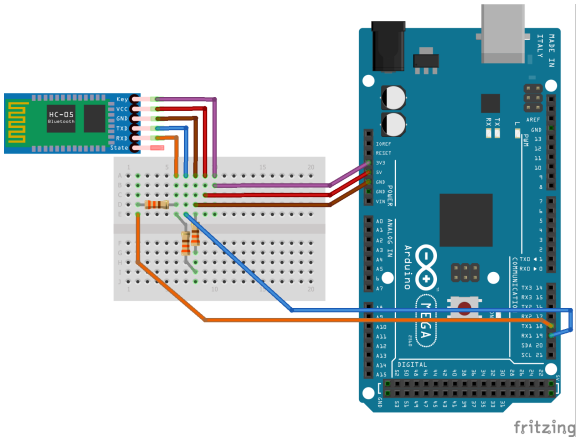
5 lcgamboa; jsmith-solectria. *PICsimLab - PIC Simulator Laboratory*. Disponível em: <https://github.com/lcgamboa/picsimlab>. Citado na página 11.

Apêndices

APÊNDICE A – Configurando módulo *bluetooth* HC-05 como mestre

O Módulo Bluetooth HC-05 pode ser alimentado com 5V, mas os pinos de RX/TX trabalham com 3.3V. Por este motivo é necessário a construção de um divisor de tensão para que a comunicação ocorra corretamente como mostra a Figura 5.Note que, a ligação do HC-05 deve ser feita encaixando o módulo na protoboard e alguns módulos possuem um *push button* no pino key (ou EN em alguns modelos).

Figura 5 – Montagem exemplo para configuração da placa



Fonte: O Autor

Para efetuar a configuração do módulo é preciso estabelecer comunicação Serial para enviar os comandos AT. Para isso carregue o seguinte programa no Arduíno:

```
String stream;

void setup() {
  Serial.begin(9600);
  Serial1.begin(38400);
  Serial.println("Inserir comando AT:");
}

void loop() {
  if(Serial.available()){
    stream="";
    while(Serial.available()){
      delay(10);
      stream+=(char)Serial.read();
    }
    Serial.print("Sent: ");
    Serial.println(stream);
    Serial1.println(stream);
  }
```

```
20    if(Serial1.available()){
21      stream="";
22      while(Serial1.available()){
23        delay(10);
24        stream+=(char)Serial1.read();
25      }
26      Serial.print("Received: ");
27      Serial.println(stream);
28    }
29  }
```

Após carregar o programa os comandos uma série de comandos AT, que deve ser enviada pelo terminal de comunicação da IDE Arduíno, mas antes disso é preciso habilitar esses comandos:

- Em módulos com pino EN e *push button*: Retire o módulo da protoboard; Mantenha o *push button* pressionado; reinsira o módulo na protobord.
- Módulos que não possuem o *push button*: A ligação de 3.3V no pino key é o bastante para habilitar os comandos AT.

Quando habilitado o padrão de luz no LED deve mudar de intermitente rápido para intermitente lento, com essa certificação os comandos podem ser enviados e todos devem responder com **OK**:

- **AT**: Testa comunicação. Se não responder altere o *baud rate* da comunicação com o módulo (linha 5) até responda com a mensagem correta.
- **AT+VERSION**: Retorna a versão do *firmware* da placa. Alguns comandos mudam de acordo com a versão do *firmware*, os comandos descritos nessa seção são para versões 2.0 até abaixo de 3.0.
- **AT+ORGL**: Reseta o módulo para a configuração padrão.
- **AT+UART?**: Retorna o *baud rate* definido para a comunicação do módulo. É possível alterar esse parametro com o comando *AT+UART=xxxx,0,0* com xxxx o valor de *baud rate* desejado.
- **AT+RMAAD**: Remove dispositivos anteriormente pareados.
- **AT+ROLE=1**: Define o modo de operação do módulo como mestre.
- **AT+CMODE=1**: Permite a conexão a qualquer endereço com o primeiro que conseguir estabelecer conexão.
- **AT+PSWD=xxxx**: Define a senha do módulo mestre, que deve ser a mesma do módulo escravo. Por padrão a senha é 1234 isso pode ser alterado configurando a

senha do módulo escravo com comandos AT. Módulos HC-06 recebem comandos sem nenhuma configuração especifica.

A partir disso o módulo configurado se conectará ao primeiro modulo escravo que possuir a configuração de senha igual. Uma conexão mais especifica pelo endereço do módulo (identificador único) pode ser configurada utilizando mais comandos. Mais informações sobre comandos AT e versões de firmware podem ser encontradas em *datasheet* dos módulos *bluetooth*. Seguem alguns *links* com informações complementares:

- <https://www.filipeflop.com/blog/tutorial-arduino-bluetooth-hc-05-mestre/>. Bluetooth HC-05: Configurando via Arduino. Acesso em 30/06/2019.
- <https://www.robocore.net/tutoriais/configurando-bluetooth-hc-05-via-arduino>. Como usar o Arduino Bluetooth HC-05 em modo mestre. Acesso em 30/06/2019.
- <https://www.arduinoecia.com.br/2013/03/modulo-bluetooth-jy-mcu-configuracao.html>. Módulo Bluetooth JY-MCU - HC-06 - Configuração. Acesso em 30/06/2019.
- <http://www.electronicaestudio.com/docs/istd016A.pdf>. Datasheet HC-05. Acesso em 30/06/2019.

APÊNDICE B – Funções complementares do software da plataforma PICGenius

Código das funções complementares para manipulação de string e atualização de variaveis de controle secundárias:

```
void blank(char * word,int len){
     int i;
     for(i=0; i<len;i++){
         word[i]=' ';
     }
}

void setCMD(){
    char value[8];
    int index,index2;
    blank(&serial_output,16);
    IntToStr(param,value);
    switch(cmd_state){
       case 0:serial_output[0]='m';
              serial_output[1]='t';
              break;
       case 1:serial_output[0]='k';
              serial_output[1]='p';
              break;
       case 2:serial_output[0]='k';
              serial_output[1]='i';
              break;
       case 3:serial_output[0]='k';
              serial_output[1]='d';
              break;
       default:break;
    }
    for(index=0;index<8;index++){
        if(value[index]!=' ')break;
    }
    for(index,index2=2;index<8;index++,index2++){
        if(value[index]=='\n')break;
        serial_output[index2]=value[index];
    }
}

void setPreValue(){
   lcd_Out(2,1,"Prev:           ");
   IntToStr(prev,half_line);
   lcd_Out(2,11,half_line);
}

void setLapTime(){
    int seconds;
    int minutes;
    char string_aux[7];

```

```
48      minutes=miliseconds/60000;
49      miliseconds=miliseconds%60000;
50      seconds=miliseconds/1000;
51      miliseconds=miliseconds%1000;
52
53      blank(full_line,16);
54      IntToStr(miliseconds,string_aux);
55      full_line[15]=(string_aux[5]!=' ')?string_aux[5]:'0';
56      full_line[14]=(string_aux[4]!=' ')?string_aux[4]:'0';
57      full_line[13]=(string_aux[3]!=' ')?string_aux[3]:'0';
58      full_line[12]=':';
59      IntToStr(seconds,string_aux);
60      full_line[11]=(string_aux[5]!=' ')?string_aux[5]:'0';
61      full_line[10]=(string_aux[4]!=' ')?string_aux[4]:'0';
62      full_line[9]=':';
63      if(minutes<100){
64         IntToStr(minutes,string_aux);
65         full_line[8]=(string_aux[5]!=' ')?string_aux[5]:'0';
66         full_line[7]=(string_aux[4]!=' ')?string_aux[4]:'0';
67      }else{
68         full_line[8]='+';
69         full_line[7]='9';
70         full_line[6]='9';
71      }
72      full_line[0]='L';
73      full_line[1]='a';
74      full_line[2]='p';
75  }
```