INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

COMPUTAÇÃO FÍSICA

PROJECTO 4

André Fonseca
A39758@alunos.isel.pt

Daniel Santos
A32078@alunos.isel.pt

Guilherme Rodrigues
A41863@alunos.isel.pt

Professor Eng. Carlos Carvalho
cfc@cedet.isel.pt

Junho 2017

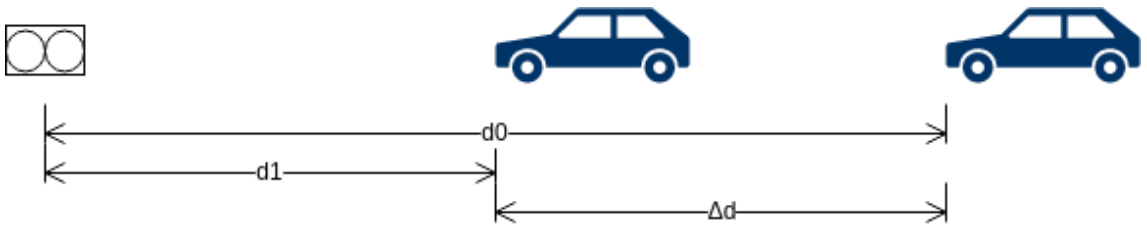
ÍNDICE

INTRODUÇÃO

Os processadores e/ou microcontroladores e sensores estão presentes no dia-a-dia de todas as pessoas na utilização dos mais diferentes e variados objectos, como: despertadores, pulseiras de desporto electrónicas, carros, computadores, etc. Este último projecto trata-se de mais uma aplicação prática destes componentes. Pretende-se a implementação de um radar de velocidades através de um sensor de distâncias.

O radar de velocidade utiliza um sensor de distância por ultra-sons HC-SR04 que será controlado por um botão de pressão. Quando o botão é pressionado, o radar realiza uma leitura de velocidade de um objecto em movimento. Assim que esta leitura seja realizada a velocidade é apresentada num display I2C.

Na fase seguinte foi implementada, em Python, uma interface gráfica do utilizador[1]. Esta GUI comunica com o radar de velocidade, executado em Arduino, através do canal da porta Serial.



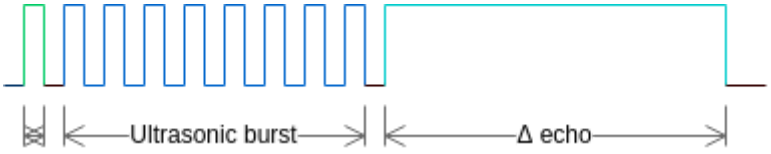
[1] GUI: Graphical User Interface

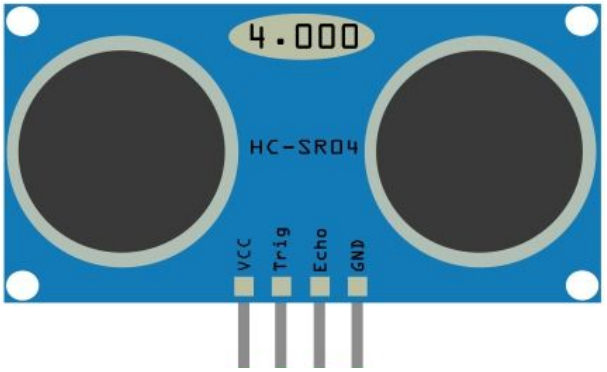
SENSOR DE DISTÂNCIA E CÁLCULO DE VELOCIDADES

O sensor de distância trata-se de um sonar HC-SR04. O processo de medição de distância deste sonar inicia-se pela emissão de um sinal sonoro. Este sinal viaja pelo espaço onde o sensor está contido, sendo reflectido em todos os objectos que entra em contacto. Quando o sinal é reflectido de volta ao sensor, denominado de eco, este mede a sua distância através da diferença de tempo entre a emissão e a recepção do sinal reflectido por um objecto. Pelas instruções do fabricante, esta diferença de tempo deve de ser divida por 58 para se obter a distância real.

$$d = \frac{\Delta eco}{58}$$

A emissão do sinal sonoro é realizada pela activação o pino trigger do sensor. De seguida é gerado um sinal acústico de *40 kHz*. O tempo de espera máximo para detectar o seu eco é de *38 ms*, sendo que é aconselhado pelo fabricante a haver um tempo mínimo entre trigger's de 60ms.

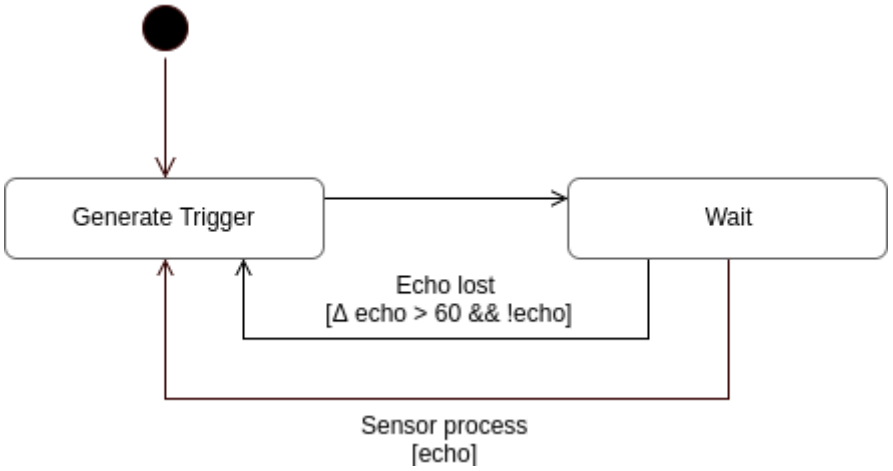




Certos objectos ou superfícies podem exibir propriedades não reflectoras e podem dificultar o processo de medição de distância. O intervalo de distâncias é limitado entre os 2 cm e 4 m.

O processo de medição de distâncias pode ser realizado através de uma máquina de estados e é representado por um diagrama de actividade.

A máquina de estados do sensor é constituída por dois estados: Gerar trigger e Esperar. O estado de Gerar trigger tem uma transição automática visto que deve de esperar pela recepção do sinal de eco. O estado Esperar apresenta duas transições para o estado anterior, quando recebe o sinal de eco ou quando ultrapassar o tempo de espera e não existir eco.




```
1.  void sensor_state_machine() {
2.      if (sensor == STATE_GENERATE_TRIGGER && button == STATE_BUTTON_CLICKED) {
3.          boolean trigger = generate_trigger();
4.
5.          if (trigger) {
6.              sensor = STATE_WAITING;
7.          }
8.
9.          return;
10.     }
11.
12.     if (sensor == STATE_WAITING) {
13.         long timer = millis() - sensor_trigger_timer;
14.
15.         if (timer > SENSOR_TRIGGER_GAP && !sensor_echo) {
16.             on_echo_lost();
17.             sensor = STATE_GENERATE_TRIGGER;
18.             return;
19.         }
20.
21.         if (sensor_echo) {
22.             sensor_process();
23.             sensor = STATE_GENERATE_TRIGGER;
24.             return;
25.         }
26.     }
27. }
```

Para determinar quando o eco é recebido são utilizadas duas funções de interrupts:

```
1.  void on_echo_released() {
2.      sensor_echo = false;
3.      sensor_echo_initial_time = micros();
4.      attachInterrupt(digitalPinToInterrupt(PIN_SENSOR_ECHO), on_echo_received,
    FALLING);
5.  }
6.
7.  void on_echo_received() {
8.      sensor_echo = true;
9.      sensor_echo_final_time = micros();
10.     attachInterrupt(digitalPinToInterrupt(PIN_SENSOR_ECHO), on_echo_released,
    RISING);
11. }
```

O cálculo de distâncias e velocidades ocorre quando a acção Sensor Process (*sensor_process()*) é realizada.

Inicialmente, a função *sensor_process()* calcula a distância do último impulso e verifica se essa distância está entre a gama de valores de distâncias suportadas pelo sensor ([2 cm; 4m]). Quando a distância calculada é válida, esta é armazenada - de forma a que quando se obter a próxima distância possa ser calculada a velocidade. A fórmula do cálculo da velocidade é:

$$v = \frac{\Delta d}{\Delta t}$$

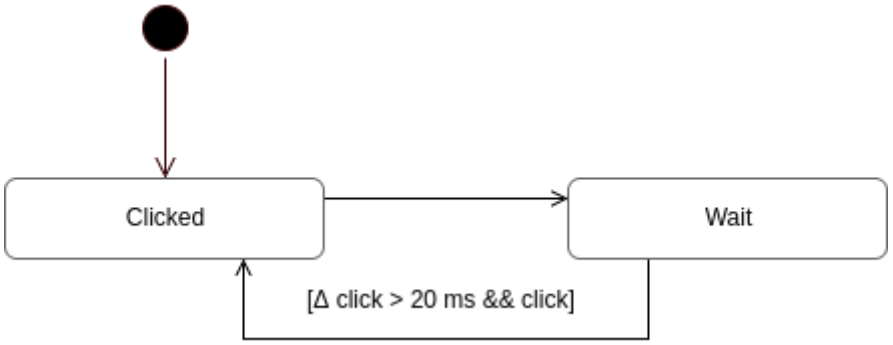
```
1.  void sensor_process() {
2.      unsigned long delta = sensor_echo_final_time - sensor_echo_initial_time;
3.      float distance = delta / 58;
4.
5.      if (distance < 2 || distance > 400) {
6.          return;
7.      }
8.
9.      object_distance[0] = object_distance[1];
10.     object_time[0] = object_time[1];
11.
12.     object_distance[1] = distance;
13.     object_time[1] = sensor_echo_final_time;
14.
15.     if (object_distance[0] == -1 || object_time[0] == -1) {
16.         return;
17.     }
18.
19.     float delta_distance = fabs(object_distance[1] - object_distance[0]);
20.     float delta_time = abs(object_time[1] - object_time[0]);
21.
22.     object_velocity = delta_distance / delta_time;
23.     object_velocity *= pow(10, 4);
24. }
```

BOTÃO DE PRESSÃO

O botão de pressão apresenta uma particularidade quando se pretende detectar se este está a ser devidamente pressionado.

Para validar se o botão de pressão está a ser pressionado deve-se realizar uma verificação num espaço temporal de 20 ms em que durante este tempo é necessário que a leitura do pino digital se encontre activa.

De forma a que seja realizada uma medição quando o botão de pressão seja pressionado foi elaborado o seguinte diagrama de actividades para a máquina de estados do botão de pressão:




```
1.  void button_state_machine() {
2.      boolean button_status_previous = button_status;
3.      button_status = digitalRead(PIN_BUTTON);
4.
5.      if (button == STATE_WAITING) {
6.          long timer = millis() - button_timer;
7.
8.          if (!button_status_previous && button_status && timer >
    BUTTON_PRESSING_INTERVAL) {
9.              button = STATE_BUTTON_CLICKED;
10.         }
11.         return;
12.     }
13.
14.     if (button == STATE_BUTTON_CLICKED) {
15.         on_button_clicked();
16.
17.         if (true) {
18.             button = STATE_WAITING;
19.         }
20.         return;
21.     }
22. }
```

```
1.  void on_button_clicked() {
2.      display_print_string("Speed (m/s):" + String(object_velocity));
3.      Serial.println(String(object_velocity)+" m/s");
4.  }
```

DISPLAY I2C

Pinos para interface física:

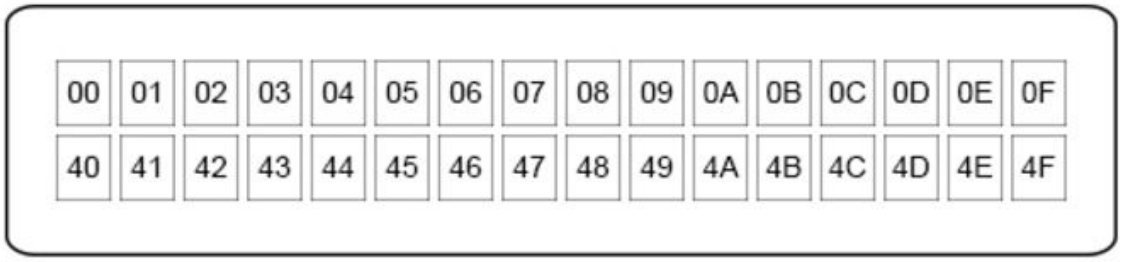
- D0..D7: Dados
- RS (Register Selector): Seleciona os registos entre IR (Instruction register) e DR (Data Register);
    - Se RS = “0” - A informação enviada para o display é um comando e a informação lida indica o estado do display(IR)
    - Se RS = “1” - A informação enviada para o display é identificada como dados/caracteres(DR);
- $R/!W$: Escrever/ler de/para o lCD;
    - $R/!W$ = “0” - Escrita de comando ou caracteres;
    - $R/!W$ = “1” - Leitura de caracteres ou informação de estado;
- E: Enabler pin Usado para iniciar a transferência de comandos ou dados.
    - Escrever no ecrã - Acontece quando ocorre uma transição descendente de E.
    - Ler a informação - Acontece logo após a transição descendente e fica disponível até á próxima transição descendente

Set de instruções LCD

- Clear display: Limpa o display(ecrã vazio).
- Cursor Home: Coloca o cursor na posição inicial.
- Character Entry Mode: Modo de escrita (para a esquerda ou direita). Estas operações são efectuadas durante a escrita/leitura dos dados da CG (Character Generator) ou DD (Display Data) RAM.
- Display On/OFF & Cursor: Controla se o display e o cursor estão ON/OFF e se o cursor está a piscar ou não.
- Cursor ou Display shift: Move o cursor e faz shift ao display sem alterar o conteúdo da DD RAM
- Function Set: Configuração básica de funcionamento do display, o comprimento dos dados, o número de linhas do display o tipo de fonte do carácter.

- Set CGRAM Address: Endereçar CG (Character Generator) RAM. Colocar RS a 1 manda os dados para o CG RAM em vez do DD RAM.
- Set Display Address: Endereçar DD (Display Data) RAM. Colocar RS a 1 envia os dados para a RAM do display, o cursor avança na direcção onde 1/D foi definido.

Endereçamento dos caracteres no display (16x2)



Codificação dos comandos para o display do LCD

Commando	Binário								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 ou 03
Character Entry Mode	0	0	0	0	0	1	1/D	S	04 a 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 a 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 a 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 a 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 a 7F
Set Display Address	1	A	A	A	A	A	A	A	80 a FF

1/D:	1=Increment, 0=Drecement	R/L:	1=Right Shift, 0=Left Shift
S:	1=Display shift On, 0=Off*	8/4:	1=8-bit interface*, 0=4-bit interface
D:	1=Display On, 0=Off	2/1:	1=2 line mode, 0=1 line mode*
U:	1=Cursor Underline On, 0 =Off*	10/7:	1=5x10 dot format, 0=5x7 dot format*
B:	1=Cursor blink On, 0=Off*		
D/C:	1=Display shift, 0=Cursor move	x=Don't care	*=Initialization Settings

Inicialização do LCD

Apesar do cabo do pino de saída consistir em data bus de 8 bits (DB0-DB7), tradicionalmente todos usam o LCD em modo de 4 bits para poupar pinos de I/O ao dispositivo interlocutor com o display.

Uma vez selecionado o modo de 4 bits, basta enviar 2 pacotes de 4 bits. Cada pacote de 4 bits é normalmente designado como “nible”. Inicialmente, é enviado o nible de maior peso (maior significado) primeiro e o nible de menor peso (menor significado) em último.

Para activar o modo de 4 bits do LCD é necessário seguir uma sequência especial de inicialização que indica ao LCD qual o modo escolhido pelo utilizador. Chama-se a esta sequência especial “resetting the LCD”:

```
1.      display_write_command_4(0x03);
2.      delay(5);
3.      display_write_command_4(0x03);
4.      delayMicroseconds(200);
5.      display_write_command_4(0x03);
6.      display_write_command_4(0x02);
7.      display_write_command_4(0x02);
8.      display_write_command_4(0x08);// (0x30 - para 8-bit e 0x20 para 4-bit)
9.
10.     delayMicroseconds(120);
```

A flag de busy só é válida após esta sequência de reset. No entanto, usualmente não é usada flag de busy no modo de 4 bit, pois seria necessário a escrita de código para ler 2 nibles do LCD.

Em vez disso, coloca-se uma certa quantidade de delays. O delay necessário pode variar consoante o LCD usado.

LCD Programa de interface

- void display_initialize(): Inicializa o LCD para funcionar em modo de 4 bits.
- display_write_command_4(byte data): Envia um comando para o LCD.
- display_write_command_8(byte data): Envia um comando para o LCD.Usando instruções display_write_command_4() e máscaras de bits para shiftar os bits de modo a enviar primeiro o nible de maior peso/significância.
- void display_write_data_4(): Envia um caracter para o LCD.
- void display_write_data_8(): Envia um caracter para o LCD.Usando instruções display_write_data_4() e máscaras de bits para shiftar os bits de modo a enviar primeiro o nible de maior peso/significância.
- void display_print_char(char data): Recebe uma variável do tipo char e faz print dele no ecrã do display usando os métodos display_write_data_4() e display_write_data_8().
- void display_print_string(String data): Limpa o ecrã e Recebe uma String e faz print desta no ecrã do display percorrendo cada caracter e usando o método void display_print_char().
- void display_clear(): Limpa o display.
- void display_set_cursor(byte line, byte column): Move o cursor para a address especificada pelos valores recebidos como parâmetro.

INTERFACE GRÁFICA DE UTILIZADOR

A interface gráfica é a ponte entre o utilizador e os dispositivos digitais. Neste caso concreto, corresponde ao sensor de distâncias com o qual se consegue obter velocidades.

Através da library Serial utilizam-se instruções como Serial.read() para obter informação sob a forma de bytes do arduino e Serial.write() e enviar também informação para o arduino sob a forma de bytes. Torna-se importante salientar a necessidade de "tratar" essa informação, ou seja, ao ler a informação recebida deve-se fazer decode e transformar para uma string. No caso de se enviar informação através da interface para o arduino, deve ser executado o encode a' string de modo a que esta seja enviada sob a forma de bytes.

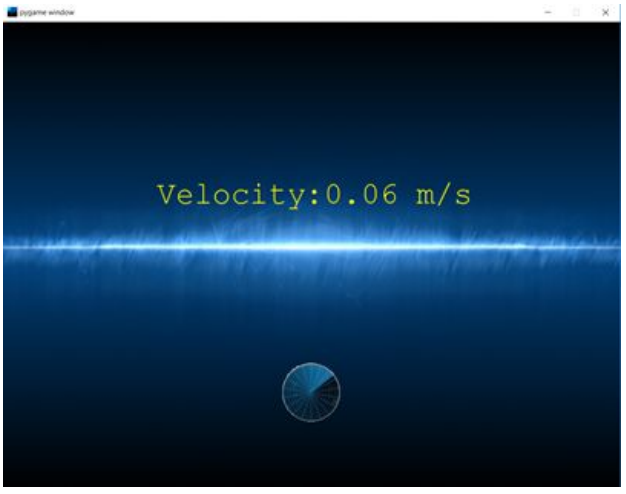
A comunicação entre a GUI e o arduino dá-se através de um loop em que se verifica se o botão da GUI foi clicado ou se existe alguma leitura do arduino à espera para ser apresentada no ecrã da GUI (executar uma leitura ao carregar no botão físico que se encontra na breadboard).

Descrição do funcionamento:

- **Botão pressionado (GUI)-** Envia-se informação para o arduino sob a forma de um caracter previamente definido e interpretado pelo arduino como um pedido de uma nova leitura. Assim, fica á espera de receber a mesma. Depois, faz-se um print da informação no ecrã e a reprodução do som de um sonar.

- **Botão pressionado (Breadboard)-** Assim que se detecta a execução de uma nova leitura de velocidades e que se encontra à espera na porta COM designada, a informação é tratada e avança-se com o print desta no ecrã. Logo de seguida sucede-se a reprodução do som de um sonar.

O programa termina quando o utilizador decide fechar a janela da GUI.



CONCLUSÃO

O controlo de velocidades de tráfego e’ um problema presente em todas as vias de comunicação terrestres. Como foi visto, este problema é facilmente resolvido com um sensor, um Arduino e a respectiva programação.

O ponto chave deste projecto não se trata simplesmente da sua resolução, mas sim na ideia de que existem inúmeros problemas actuais na sociedade que podem ser resolvidos com este tipo de soluções. Numa era em que a tecnologia de Internet Of Things (IoT) encontra-se em expansão, o Arduino e outro tipo de componentes electrónicos de pequena escala, como o Raspberry Pi, podem ser soluções de baixo custo. Sendo apenas necessário o conhecimento conceptual e criatividade.

ANEXO – CÓDIGO DE SIMULAÇÃO EM ARDUINO

```
#include <Wire.h>

// Addresses
#define RS 0x01
#define RW 0x02
#define EN 0x04
#define LIGHT 0x08
#define LCD_ADDRESS 0x3F

// Pins
#define PIN_BUTTON 7

// States
#define STATE_WAITING 0
#define STATE_GENERATE_TRIGGER 1
#define STATE_CALCULATE_DISTANCE 2
#define STATE_BUTTON_CLICKED 1

int sensor = STATE_GENERATE_TRIGGER;
int button = STATE_WAITING;

// Properties
#define PIN_SENSOR_ECHO 2
#define PIN_SENSOR_TRIGGER 4
#define SENSOR_TRIGGER_GAP 60
#define SENSOR_TRIGGER_DELAY 10
#define SENSOR_ECHO_LIMIT 38
#define BUTTON_PRESSING_INTERVAL 20

unsigned long sensor_trigger_timer;
volatile boolean sensor_echo;
volatile unsigned long sensor_echo_initial_time;
volatile unsigned long sensor_echo_final_time;

double object_distance[] = {-1, -1};
long object_time[] = {-1, -1};
float object_velocity;

unsigned long button_timer;
boolean button_status;

void setup() {
   Serial.begin(9600);
   // Sensor
   pinMode(PIN_SENSOR_TRIGGER, OUTPUT);
     attachInterrupt(digitalPinToInterrupt(PIN_SENSOR_ECHO),  on_echo_released,
RISING);
   interrupts();
   // LCD
   display_initialize();
   display_clear();
}

void loop() {
   sensor_state_machine();
   button_state_machine();
   read_input();
}

void read_input() {
   if (Serial.available()) {
       if (Serial.read() == 'R') {
           button = STATE_BUTTON_CLICKED;
       }
```

```
    }
}

/*
* Sensor
*/
void sensor_state_machine() {
    if (sensor == STATE_GENERATE_TRIGGER && button == STATE_BUTTON_CLICKED) {
        boolean trigger = generate_trigger();

        if (trigger) {
            sensor = STATE_WAITING;
        }

        return;
    }

    if (sensor == STATE_WAITING) {
        long timer = millis() - sensor_trigger_timer;

        if (timer > SENSOR_TRIGGER_GAP && !sensor_echo) {
            on_echo_lost();
            sensor = STATE_GENERATE_TRIGGER;
            return;
        }

        if (sensor_echo) {
            sensor_process();
            sensor = STATE_GENERATE_TRIGGER;
            return;
        }
    }
}

boolean generate_trigger() {
    long trigger_delta = millis() - sensor_trigger_timer;

    if (trigger_delta < SENSOR_TRIGGER_GAP) {
        return false;
    }

    sensor_trigger_timer = millis();

    digitalWrite(PIN_SENSOR_TRIGGER, HIGH);
    delayMicroseconds(SENSOR_TRIGGER_DELAY);
    digitalWrite(PIN_SENSOR_TRIGGER, LOW);

    return true;
}

void on_echo_released() {
    sensor_echo = false;
    sensor_echo_initial_time = micros();
      attachInterrupt(digitalPinToInterrupt(PIN_SENSOR_ECHO),  on_echo_received,
FALLING);
}

void on_echo_received() {
    sensor_echo = true;
    sensor_echo_final_time = micros();
      attachInterrupt(digitalPinToInterrupt(PIN_SENSOR_ECHO),  on_echo_released,
RISING);
}

void on_echo_lost() {
    object_time[0] = -1;
    object_time[1] = -1;
    object_distance[0] = -1;
    object_distance[0] = -1;
```

```
    Serial.println("Echo lost.");
    display_print_string("echo lost");
}

void sensor_process() {
    unsigned long delta = sensor_echo_final_time - sensor_echo_initial_time;
    float distance = delta / 58;

    if (distance < 2 || distance > 400) {
        return;
    }

    object_distance[0] = object_distance[1];
    object_time[0] = object_time[1];

    object_distance[1] = distance;
    object_time[1] = sensor_echo_final_time;

    if (object_distance[0] == -1 || object_time[0] == -1) {
        return;
    }

    float delta_distance = fabs(object_distance[1] - object_distance[0]);
    float delta_time = abs(object_time[1] - object_time[0]);

    object_velocity = delta_distance / delta_time;
    object_velocity *= pow(10, 4);
}

/*
* Button
*/
void button_state_machine() {
    boolean button_status_previous = button_status;
    button_status = digitalRead(PIN_BUTTON);

    if (button == STATE_WAITING) {
        on_button_waiting();
        long timer = millis() - button_timer;

                if  (!button_status_previous  &&  button_status  &&  timer  >
BUTTON_PRESSING_INTERVAL) {
            button = STATE_BUTTON_CLICKED;
        }
        return;
    }

    if (button == STATE_BUTTON_CLICKED) {
        on_button_clicked();

        if (true) {
            button = STATE_WAITING;
        }
        return;
    }
}

void on_button_waiting() {

}

void on_button_clicked() {
    display_print_string("Speed (m/s):" + String(object_velocity));
    Serial.println(String(object_velocity)+" m/s");
}

/*
* LCD
*/
```

```
void display_write_data_4(byte data) {
   Wire.beginTransmission(LCD_ADDRESS);
   Wire.write((data << 4) | LIGHT | RS);
   Wire.endTransmission();

   Wire.beginTransmission(LCD_ADDRESS);
   Wire.write((data << 4) | LIGHT | RS | EN );
   Wire.endTransmission();

   delayMicroseconds(1); // enable ativo >450ns

   Wire.beginTransmission(LCD_ADDRESS);
   Wire.write((data << 4) | LIGHT | RS);
   Wire.endTransmission();

   delayMicroseconds(40);
}

void display_write_command_4(byte data) {
   Wire.beginTransmission(LCD_ADDRESS);
   Wire.write((data << 4) | LIGHT );
   Wire.endTransmission();
   Wire.beginTransmission(LCD_ADDRESS);
   Wire.write((data << 4) | LIGHT | EN );
   Wire.endTransmission();
   delayMicroseconds(1); // enable ativo >450ns
   Wire.beginTransmission(LCD_ADDRESS);
   Wire.write((data << 4) | LIGHT);
   Wire.endTransmission();
   delayMicroseconds(40);
}

void display_write_command_8(byte data) {
   display_write_command_4(data >> 4);
   display_write_command_4(data);
}

void display_write_data_8(byte data) {
   display_write_data_4(data >> 4);
   display_write_data_4(data);
}

void display_initialize() {
   Wire.begin();
   delay(50);

   display_write_command_4(0x03);
   delay(5);
   display_write_command_4(0x03);

   delayMicroseconds(200);

   display_write_command_4(0x03);
   display_write_command_4(0x02);
   display_write_command_4(0x02);
   display_write_command_4(0x08);

   delayMicroseconds(120);

   display_write_command_4(0x00);
   display_write_command_4(0x0F);

   delayMicroseconds(120);

   display_write_command_4(0x08);
   display_write_command_4(0x00);

   delayMicroseconds(120);
}
```

```
void display_clear() {
   display_write_command_8(0x01);
   delay(5);
}

void display_set_cursor(byte line, byte column) {
   display_write_command_8(0x80 | line << 6 | column);
}

void display_print_char(char data) {
   display_write_data_8(data);
   delayMicroseconds(120);
}

void display_print_string(String data) {
   display_clear();
   display_set_cursor(0,0);

   for (int i = 0; i < data.length(); i++) {
       if(i == 16) {
           display_set_cursor(1,0);
       }

       display_print_char(data[i]);
   }
}

void display_print_int(int data) {
   String s = String(data);
   display_print_string(s);
}
```

ANEXO - CÓDIGO DA INTERFACE GRÁFICA DE UTILIZADOR

```
import pygame
import serial

com = 'COM1'
baudrate = 9600

antialias = True


def Button(Picture, coords, surface):
   print("buttonify")
   image = pygame.image.load(Picture)
   imagerect = image.get_rect()
   imagerect.center = coords
   surface.blit(image, imagerect)
   return (image, imagerect)

def freeText(screen, text, x, y, color, size, font):
   print("Entrou")
   fontType = pygame.font.SysFont(font, size)
   textDisplay = fontType.render(text, antialias, color)
   textpos = textDisplay.get_rect()
   textpos.center = (x, y)
   screen.blit(textDisplay, textpos)

    # Componentes adicionais, não necessárias para o funcionamento da função apenas para ajuda na composição do jogo de modo a conhecer a posição ocupada pelo texto
   rect_coordinates = [textpos.topleft, textpos.bottomright]
   return rect_coordinates

def GUI():
    array_GUI = []
    pygame.init()
    screen = pygame.display.set_mode((1000, 750))
    background_image=pygame.image.load("sonar1.jpg").convert_alpha()
    screen.fill((255, 255, 255))
    screen.blit(background_image, (0, 0))

    #window icon
    icon = pygame.image.load("icon.png")
    pygame.display.set_icon(icon)

    x = Button("sonar_or_radar_screen_mini_button.png", (500, 600), screen)

    textLabel(screen, "")

    pygame.display.update()

    array_GUI.append(screen)
    array_GUI.append(x[1])

    return array_GUI #x[1]

def textLabel(screen, text):
    # initialize font; must be called after 'pygame.init()' to avoid 'Font not Initialized' error
   myfont = pygame.font.SysFont("monospace", 50)

   # render text
   label = myfont.render(text, 1, (255,255,0))
   screen.blit(label, (375, 300))
```

```
def button_pressed(m_x, m_y, b_coords):
   x_left=b_coords[0]
   y_top=b_coords[1]
   width=b_coords[2]
   x_rigth=x_left+width
   height=b_coords[3]
   y_bot=y_top+height

   if m_x > x_left and m_x < x_rigth and m_y > y_top and m_y < y_bot:

       print("botão foi clicado")
       return True

   return False


def resetLabel(screen):
   pass


def main():
   Serie = comInit(com, baudrate)
   array_GUI = GUI()
   # button_coords = array_GUI[1]
   running = True

   while running:
       msg = stringReceive(Serie)
       if (msg):
           print(msg)

       pygame.display.update()

       for event in pygame.event.get():

           if event.type == pygame.QUIT:
               print("quiting")
               running = False
               pygame.quit()


# Função de inicialização
def comInit(com, baudrate):

   try:
       Serie = serial.Serial(com, baudrate)
       print ('Sucesso na ligacao ao Arduino.')
       print ('Ligado ao ' + Serie.portstr)
       return Serie
   except Exception as e:
       print ('Insucesso na ligacao ao Arduino.')
       print (e)
   return None

def caracterReceive(Serie):
   try:
       return Serie.read()
   except:
       print ('Erro na comunicacao.')
   Serie.close()

def caracterSend(Serie, info):

   print(info)
   try:
       Serie.write(info)
```

```
    except:
        print ('Erro de comunicacao2.')
        Serie.close()

def stringReceive(Serie):
    try:
        #decodes the byte type into string with the designated encoding
        string_decoded=Serie.readline().decode("UTF-8")

        #selects only the value of velocity from the decoded String
        space_index=string_decoded.index(" ");
        value=string_decoded[:space_index]

        return value
    except:
        print ('Erro na comunicacao1.')
        Serie.close()


if __name__ == '__main__':
    main()
```