IAD - 2.º Trabalho - Telecomando e Mapeamento

Semião, Bruno (100292) | Assunção, Eduarda (100301) | Simenta, Sofia (100368) | Marques, Patrícia (100357)

1 Hardware: Componentes do Carro

- **Breadboard**
- **Arduino Uno R3**
- **Módulo Ponte H L298N**
- **2 pilhas 9 V**: Uma para alimentar o Arduino, outra para o motor.
- **Elásticos fixadores da roda traseira**: Para eliminar erros do movimento, colocando a roda no sentido de movimento quando está a andar em frente.

2 Interface: Secções da Aplicação

- **Time Step e Time Range**
- **Variável no eixo vertical**

  (**Nota:** Estas secções estão especificadas no relatório do 1º trabalho)
- **Modos de funcionamento:** Mapeamento com acelerómetro, sem acelerómetro, mapeamento de só rotação e controlo livre.

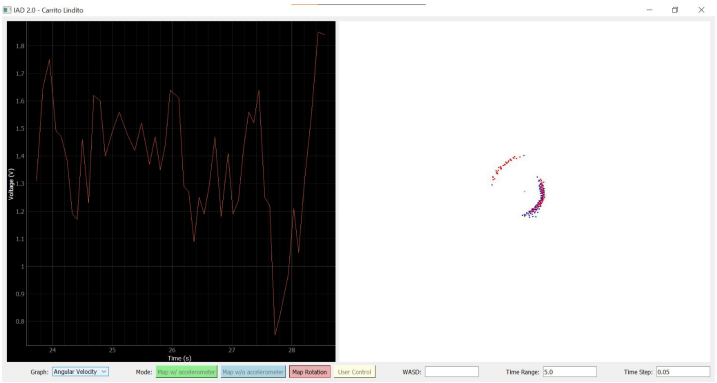


Figura 1: *Interface Gráfica do Utilizador atualizada em Tempo Real*

3 Comunicação Bluetooth

Para pôr o módulo a funcionar, foi necessário um divisor de tensão, pois o módulo lê 3.3 V e não 5 V como seriam enviados a partir de um "HIGH". Usamos o módulo Bluetooth HC-05 para tornar o carro portátil. Para o módulo **enviamos** velocidade e sentido das rodas esquerda *L* e direita *R*. Do módulo Bluetooth **recebemos** a distância detetada pelos sensores da frente e da direita, medições tanto do acelerómetro como do giroscópio.

```
#Setup Bluetooth connection at beggining of execution
def setupBluetooth(self):
    #Connect
    try:
        self.bluetooth = serial.Serial(self.BT_PORT,
    9600, timeout=1)
        self.bluetooth.write_timeout = 0.1
        print(f"Connected successfully to {self.BT_PORT
    }\n\n")
    except:
        print(f"OOPSIE - Could not connect to bluetooth
    port {self.BT_PORT}!\nExiting program.")
        exit()

    #Clear input buffer
    try:
        self.bluetooth.flushInput()
        self.bluetooth.flushOutput()
    except:
        print("OOPSIE - Couldn't clear Input Buffer!")
```

4 Envio de Dados

Dependendo do sentido escolhido para a roda, serão enviados 0 V ou 5 V para um dos pinos. Um outro terá um valor entre 0 V e 5 V, determinado pela velocidade escolhida. Haverá então uma diferença de voltagem positiva ou negativa, causando a rotação da roda num sentido ou noutro.

```
//With Arduino, control speed and direction of the
    motors

speedL = command.substring(0,3).toInt()-100;
speedR = command.substring(3,6).toInt()-100;
invertL = command.substring(6,7).toInt();
invertR = command.substring(7,8).toInt();

analogWrite(PinPWML, speedL);
digitalWrite(PinInvL, invertL);
analogWrite(PinPWMR, speedR);
digitalWrite(PinInvR, invertR);
```

```
#Send command
self.bluetooth.write(f"{self.speedL}{self.speedR}{self.
    invL}{self.invR}\n".encode('utf-8'))
```

Como seria de esperar, andar para a frente e para trás é fornecer a mesma voltagem a ambos os motores, quer num sentido, quer noutro. Nas rotações, para virar sobre si próprio, colocamos uma roda a rodar para a frente e outra para trás, ambas com a mesma velocidade. O comando é seguido até dadas novas instruções.

5 Receção de Dados

5.1 Sensores

Usamos 2 sensores de distância ultrassónicos HC-SR04, um na frente e outro à direita. Convertemos linearmente os valores de voltagem dos sensores para distância em *mm* ao centro das rodas por via de calibrações. Usamos ainda o módulo acelerómetro giroscópio MPU-6050. O giroscópio tem bastante precisão. Já o acelerómetro não, devido ao impacto da gravidade, que é muitas ordens de grandeza superior às acelerações que o carro atinge.

```
//Receive and assign data (same for SensorR)
pinMode(PinSensorF, OUTPUT);
digitalWrite(PinSensorF, LOW);
delayMicroseconds(2);
digitalWrite(PinSensorF, HIGH);
delayMicroseconds(10);
digitalWrite(PinSensorF, LOW);
pinMode(PinSensorF, INPUT);
state = String(pulseIn(PinSensorF, HIGH)) + "!";

mpu.getEvent(&a, &g, &temp);

state += String(a.acceleration.x) + "!";
state += String(a.acceleration.y) + "!";
state += String(a.acceleration.z) + "!";
state += String(g.gyro.x) + "!";
state += String(g.gyro.y) + "!";
state += String(g.gyro.z) + "!";

bluetooth.println(state);
```

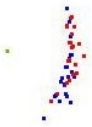
```
#Get and decode data
def Read_BT(self):
    while self.bluetooth.in_waiting <= 0 and self.isRun:
          #Wait until data is available
        time.sleep(0.005)
    try:
        self.line = self.bluetooth.readline().decode('
    utf-8').split("!")
    except:
        print("Failed to Read")
```

5.2 Calibração e Desvios

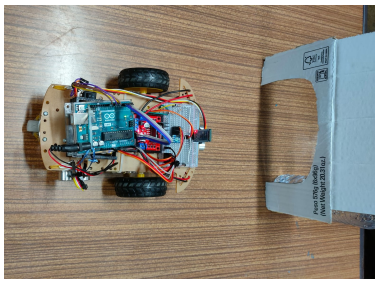
A calibração dos **sensores ultrassónicos** foi feita manualmente. Colocada uma régua com a sua origem no ponto entre as rodas, variamos e medimos a distância de um objeto ao sensor. Relacionamos esta com a voltagem adquirida para obter uma relação proporcional entre ambas. Quando nos referimos à calibração da **velocidade angular e aceleração**, referimo-nos ao desconto dos **offsets**. Estes são obtidos a partir das aquisições dos valores do acelerómetro e do giroscópio com o carro em repouso. Retiramos o *offset* para contrariar a sua inclinação inicial e usamos a *standard deviation* para ignorar valores que sejam ruído em futuras medições.

5.3 Limitações e Alternativas

Substituímos os sensores infravermelhos inicialmente utilizados pelos ultrassónicos, já que os primeiros dependiam da refletividade dos objetos e tinham baixo alcance (< 30 cm), não impedindo a travagem sem choque. Verificamos que o sensor tem um ângulo de efeito de 15º. Se, por um lado, é uma vantagem caso não haja paralelismo entre o carro e a parede, evitando colisões, por outro, dificulta a deteção de buracos pequenos 2.



2.1 Mapeamento



2.2 Setup

Figura 2: *Limitação advindo do ângulo de efeito*

6 Processamento de Dados

As medições recebidas do módulo Bluetooth são utilizadas para atualizar a posição do carro, o ângulo da sua trajetória e a posição de objetos próximos. Para calcular o ângulo de rotação, fazemos uma aproximação linear com base na velocidade angular e no período de tempo decorrido.

```
#Get angular velocity
self.velocityW = float(self.line[7]) - self.offsetW

#Calculate angle
self.phi += self.velocityW * self.timeDelta * self.
    rotationK

#Get accelerations
self.acceleration = -self.offset_acc + (float(self.line
    [2])*1000)
if(abs(self.acceleration) < 2*self.sd_acc): #Remove
    small random variations
    self.acceleration = 0

#Get position after calculating velocity
self.position[0] += int(self.positionK*self.velocity *
    self.timeDelta * math.cos(self.phi) )
self.position[1] += int(self.positionK*self.velocity *
    self.timeDelta * math.sin(self.phi) )

#Get distance in mm considering the center of the wheels
self.sensorF = int((int(self.line[0]) + 580)*0.1754 )
self.sensorR = int((int(self.line[1]) +  333)*0.1754 )
```

7 Modos de Funcionamento

7.1 Controlo livre

Há 5 opções de movimento no caso do controlo livre: frente (*W*), trás (*S*), virar no sentido horário (*D*), no sentido antihorário (*A*) ou parar (*q*).

```
# Example for backwards movement
def MovementHandler(self, val):
        match val:
            case -1: # Move Backwards
                self.speedL = 100
                self.speedR = 100
                self.invL = 1
                self.invR = 1
#Later on
match self.key:
    case 's':
        self.MovementHandler(-1)
        self.key  = ' '
```

Para evitar colisões com obstáculos, implementamos um mecanismo de controlo, ignorando os comandos inseridos que o fariam colidir.

```
#If too close to front, cannot move forward
match self.key:
    case 'w':
        if self.sensorF > 350:
            self.MovementHandler(11,0)
        else:
            self.MovementHandler(0,0)
```

7.2 Condução Automática e Mapeamento

A construção do mapa é baseada nas posições obtidas do carro e dos obstáculos próximos. A posição destes últimos é calculada a partir dos sensores de distância ultrassónicos e dos ângulos da trajetória do carro.

```
#Found object on the right
if self.sensorR < 400:
    # Position of obstacle based on angle and distance
    position_x_obj = math.floor( self.position[0] + self
    .sensorR * math.cos(self.phi - math.pi * 0.5) - 85. *
     math.cos(self.phi))
    position_y_obj = math.floor( self.position[1] + self
    .sensorR * math.sin(self.phi - math.pi * 0.5) - 85. *
     math.sin(self.phi))
    self.draw_position(position_x_obj,position_y_obj
    ,0,0,255)

#Found object on the front
if self.sensorF < 550:
    # Position of obstacle based on angle and distance
    position_x_obj = math.floor( self.position[0] + self
    .sensorF * math.cos(self.phi) )
    position_y_obj = math.floor( self.position[1] + self
    .sensorF * math.sin(self.phi) )
    self.draw_position(position_x_obj,position_y_obj
    ,255,0,0)
```

A condução automática tem instruções para que o carro se movimente em frente até identificar um objeto à sua frente. Nesse caso, gira aproximadamente 90º para a esquerda sobre o seu próprio eixo. Caso não esteja a andar exatamente a um ângulo reto da sua posição inicial, tem mecanismos de correção quando iniciar o movimento para a frente. Tem também a possibilidade de girar 90º para a direita, caso já tenha virado à esquerda e não detete uma parede ao seu lado, de forma a contornar um objeto e seguir em frente.

```
#Get diference to nearest multiple of pi/2
correction = int(400*(self.phi - 1.57075 * round(self.
    phi/1.57075, 0)))
if correction > 100:
    correction = 100
elif correction < -100:
```

```
    correction = -100

self.speedL   = 255 + correction
self.speedR   = 255 - correction
self.invL     = 0
self.invR     = 0
self.isMoving = 1
```

7.2.1 Com Acelerómetro

Existindo uma inclinação variável do chão, não basta determinar um *offset* inicial. Então, neste método, sempre que há uma rotação do carro, ocorre nova calibração da aceleração. A figura 33.1 representa o mapeamento do setup representado na figura 33.2.

3.1 Mapeamento

3.2 Setup

Figura 3: *Situação com Acelerómetro*

7.2.2 Sem Acelerómetro

Para evitar perdas de exatidão, este método modela a velocidade, segundo a figura 4, do carro ao invés de a calcular através dos valores adquiridos pelo acelerómetro. Para podermos assumir que a velocidade fica constante, é dada a mesma voltagem sempre que é decidido o movimento em frente.
Consideramos que a velocidade é acelerada uniformemente, constante (v_0) e, depois, desacelerada uniformemente até parar (figura 4). De modo a obter v_0, é feita uma calibração inicial: o carro é colocado com a frente perpendicular a uma parede, avança uma pequena distância e são registadas automaticamente as variações de distância e tempo. Apesar de ser uma boa modelação da velocidade e o mapeamento funcionar melhor, caso o carro fique preso ou embata num objeto, o programa não se apercebe por não estar a ser utilizada a aceleração real, provocando mapeamentos incorretos.

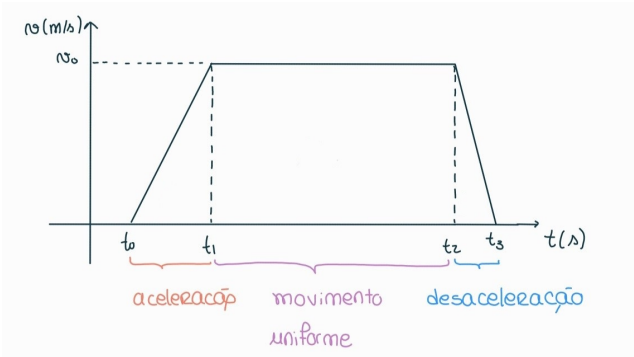
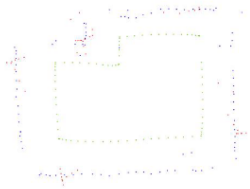


Figura 4: *Modelação da velocidade*

Observando as figuras 33.1 e 55.1 verifica-se, tal como era de esperar, que o mapeamento funciona melhor sem o acelerómetro.



5.1 Mapeamento

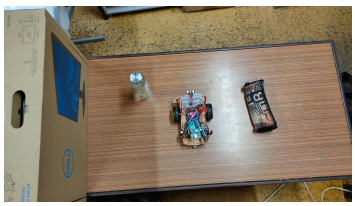
5.2 Setup

Figura 5: *Situação sem acelerómetro*

7.3 Só Com Rotação

Neste caso, não são necessárias calibrações, pois consideramos inalterada a posição do carro, sendo que este gira sobre o seu centro. Para isso, as rodas são colocadas a girar em sentidos opostos com o mesmo número de rotações. Para o *offset* da velocidade angular do carro, utilizamos o valor que foi consistentemente obtido ($-0.01 rad/s$). Para efetuar o mapeamento, basta ter em conta as distâncias adquiridas pelos sensores ultrassónicos e o ângulo do carro calculado com os valores do giroscópio. A figura 66.1 representa o mapeamento do setup representado em 66.2.

6.1 Mapeamento



6.2 Setup

Figura 6: *Situação com apenas rotação*