UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA
ATIVIDADE EXTRACURRICULAR REFERENTE À EQUIPE DE COMPETIÇÃO
TERRA COMPETITION

HEBERT ALAN KUBIS

CONTROLE DOS MOTORES

Joinville
2024

HEBERT ALAN KUBIS

CONTROLE DOS MOTORES

Atividade apresentada como estudo sobre a competição ROBOSUB que a equipe Terra Competition participará em 2024, do Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina.

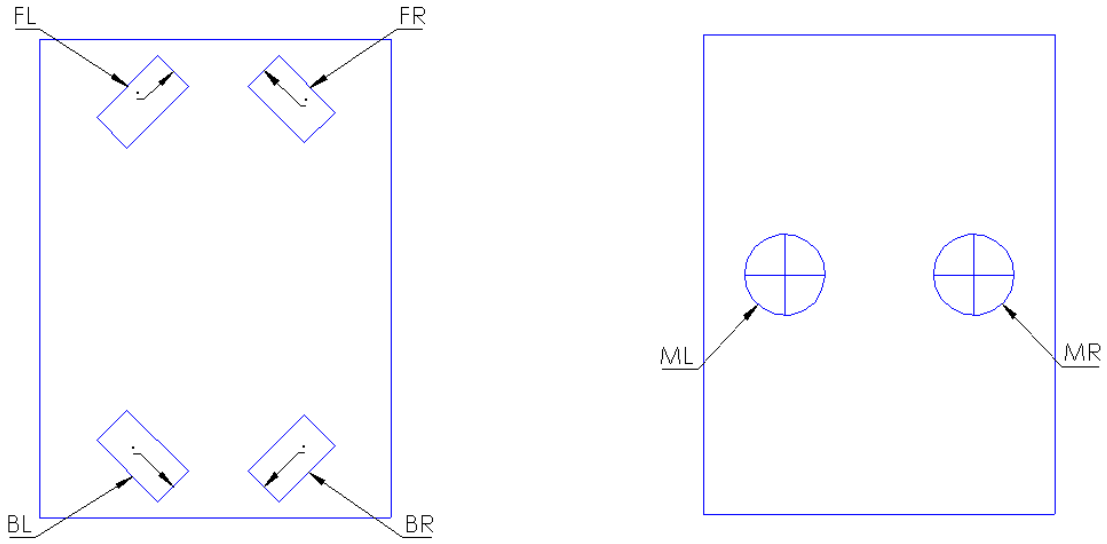
Joinville
2024

SUMÁRIO

1. DISPOSIÇÃO DOS MOTORES

Semana do dia 21 de Janeiro de 2024:

Figura 1 - Diagrama da disposição dos motores e suas respectivas frentes



Fonte: Lucas Barcaro

1. FL - Motor da frente lado esquerdo
2. FR - Morto da frente lado direito
3. ML - Motor do meio lado esquerdo
4. BR - Motor de tras lado direito
5. BL - Motor de tras lado esquerdo
6. MR - Motor do meio lado direito

Setas da primeira imagem indicando a frente dos motores e na vista superior, na segunda imagem, motores para baixo.

2. MOVIMENTAÇÕES

Para que o AUV se movimente para as direções que serão especificadas é necessário que certos motores sejam acionados ao receber cada instrução. As movimentações básicas são as de subir (UP), descer (DOWN), frente (FORWARD), trás (BACK), direita (RIGHT) e esquerda (LEFT), além de virar à direita (TURN RIGHT) e virar à esquerda (TURN LEFT).

O acionamento dos motores para cada instrução deve ser:

- **UP** - ML e MR com fluxo normal
- **DOWN** - ML e MR com fluxo inverso
- **FORWARD** - BL e BR com fluxo normal; FL e FR com fluxo reverso
- **BACK** - FL e FR com fluxo normal; BL e BR com fluxo reverso
- **RIGHT** - FR e BR com fluxo normal; FL e BL com fluxo reverso
- **LEFT** - FL e BL com fluxo normal; FR e BR com fluxo reverso
- **TURN RIGHT** - FR e BL com fluxo normal; FL e BR com fluxo reverso
- **TURN LEFT** - FL e BR com fluxo normal; FR e BL com fluxo reverso

3. IMPLEMENTAÇÕES

3.1. CÓDIGO BÁSICO

Para uma primeira implementação os códigos foram baseados em alguns códigos anteriores do drive, onde uma função toma uma decisão de movimentação e envia uma instrução do tipo *string* para o código de controle dos motores para que o AUV se movimente.

No código foram criadas constantes com os pinos usados para cada motor e uma constante para a frequência que é usada nos ESCs, assim facilitando a troca de pinos dos motores e frequência do sinal PWM caso necessário.

```
import RPi.GPIO as GPIO

# Constants for pins and frequency
FREQUENCY_PWM = 400

MR_PIN = 1
ML_PIN = 2
FR_PIN = 3
FL_PIN = 4
BR_PIN = 5
BL_PIN = 6
```

O próximo passo é a declaração de como os pinos estão sendo referenciados, que no caso é usado a numeração da placa, e em seguida definir cada pino dos motores como saída. Também é necessário configurar a frequência do PWM para cada saída e iniciar os pinos com a função *start()*.

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

for pin in [MR_PIN, ML_PIN, FR_PIN, FL_PIN, BR_PIN, BL_PIN]:
    GPIO.setup(pin, GPIO.OUT)

mr = GPIO.PWM(MR_PIN, FREQUENCY_PWM)
ml = GPIO.PWM(ML_PIN, FREQUENCY_PWM)
fr = GPIO.PWM(FR_PIN, FREQUENCY_PWM)
fl = GPIO.PWM(FL_PIN, FREQUENCY_PWM)
br = GPIO.PWM(BR_PIN, FREQUENCY_PWM)
bl = GPIO.PWM(BL_PIN, FREQUENCY_PWM)

mr.start(50)
ml.start(50)
fr.start(50)
fl.start(50)
br.start(50)
bl.start(50)
```

Em seguida foi definido uma função que interpreta o valor *action* (string), que é passado por parâmetro, convertendo-o na movimentação correspondente. Os motores ligados para cada movimentação são os descritos no começo desse documento, e o código basicamente ativa os motores com potência máxima se a instrução passar na condição ou desativa se não for a instrução, isso porque o comportamento dos motores após o término do código até que a função seja chamada novamente ainda não é conhecida, eles podem ficar ativados ou não.

```
def motors_control(action):
    if action == "UP":
        mr.ChangeDutyCycle(100)
        ml.ChangeDutyCycle(100)
    elif action == "DOWN":
        mr.ChangeDutyCycle(0)
        ml.ChangeDutyCycle(0)
    else:
        mr.ChangeDutyCycle(50)
        ml.ChangeDutyCycle(50)
    if action == "FORWARD":
        br.ChangeDutyCycle(100)
        bl.ChangeDutyCycle(100)
```

```
    else:
        br.ChangeDutyCycle(50)
        bl.ChangeDutyCycle(50)
    if action == "BACK":
        fr.ChangeDutyCycle(100)
        fl.ChangeDutyCycle(100)
    else:
        fr.ChangeDutyCycle(50)
        fl.ChangeDutyCycle(50)
    if action == "RIGHT":
        fr.ChangeDutyCycle(100)
        br.ChangeDutyCycle(100)
    else:
        fr.ChangeDutyCycle(50)
        br.ChangeDutyCycle(50)
    if action == "LEFT":
        fl.ChangeDutyCycle(100)
        bl.ChangeDutyCycle(100)
    else:
        fl.ChangeDutyCycle(50)
        bl.ChangeDutyCycle(50)
    if action == "TURN RIGHT":
        fr.ChangeDutyCycle(100)
        bl.ChangeDutyCycle(100)
    else:
        fr.ChangeDutyCycle(50)
        bl.ChangeDutyCycle(50)
    if action == "TURN LEFT":
        fl.ChangeDutyCycle(100)
        br.ChangeDutyCycle(100)
    else:
        fl.ChangeDutyCycle(50)
        br.ChangeDutyCycle(50)
```

Esse é um código básico que em teoria deve funcionar. Provavelmente existem problemas e coisas que podem ser melhoradas, principalmente na questão de quanto vai ser ativado cada motor, já que nesse caso o código simplesmente ativa ao máximo para frente ou para trás.

3.2. BIBLIOTECA PIGPIO

Semana do dia 28 de janeiro de 2024:

Com base no que foi discutido na reunião do dia 24 de janeiro de 2024 e com base em algumas questões que foram encontradas no fórum da Blue Robotics o código anterior será alterado.

A primeira alteração é na forma de configurar os pulsos do PWM, onde no primeiro código isto era feito usando *changeDutyCycle* da biblioteca GPIO, mas após a reunião esse método se tornou pouco interessante. Desta forma, o PWM será configurado usando microssegundos como parâmetros através da função *set_servo_pulsewidth(pino_gpio, pulse width (µs))* da biblioteca PIGPIO .

Outra alteração é que a pinagem dos motores será configurada no vetor PINS, que está servindo somente como exemplo, pois ainda não foram determinados os pinos que serão usados. Também foi criada uma função para inicializar os motores para deixar o código mais modular e fácil de fazer alterações caso necessário e, além disso, também foi criada uma função para finalizar a execução dos motores. Em relação a frequência do PWM, após pesquisas em fóruns da Blue Robotics e em alguns outros sites foram encontrados códigos com utilizações de 50, 60 e 100 Hz.

Como a biblioteca foi alterada é necessário criar um objeto *pi* com *pigpio.pi()*, que é um serviço em segundo plano que permite o controle dos pinos GPIO do RaspBerry, para poder controlar os motores.

```
import pigpio
import time

FREQUENCY = 50

# Pins motors
PINS = [1, 2, 3, 4, 5, 6]
# 0 - Front left
# 1 - Front right
# 2 - Middle right
# 3 - Back right
# 4 - Back left
# 5 - Middle left

pi = pigpio.pi()
```

Dando seguimento ao código, a função de inicialização dos motores configura cada pino do vetor PINS como saída, usando *pigpio.OUTPUT*, define a frequência do PWM e inicia cada motor com um pulso de 1500 µs, que segundo a documentação da Blue Robotics é o valor de repouso dos motores. Após isso espera 7 segundos para que os ESCs sejam configurados.

```
def inicialize_pins():
    for pin in PINS:
        pi.set_mode(pin, pigpio.OUTPUT)
        pi.set_PWM_frequency(pin, FREQUENCY)
        pi.set_servo_pulsewidth(pin, 1500)
    time.sleep(7)
```

Após, vem a função de controle dos motores, que recebe como parâmetro a ação que deve ser tomada e a partir disso os motores são ligados de acordo. As ações são:

- “UP” - Move para cima ligando os motores 2 e 5 para frente
- “DOWN” - Move para baixo ligando os motores 2 e 5 em reverso
- “FORWARD” - Move para frente ligando os motores 0 e 1 em reverso e os motores 3 e 4 para frente
- “BACK” - Move para trás ligando os motores 0 e 1 para frente e os motores 3 e 4 em reverso
- “RIGHT” - Move para a direita ligando os motores 1 e 3 para frente e os motores 0 e 4 em reverso
- “LEFT” - Move para esquerda ligando os motores 0 e 4 para frente e os motores 1 e 3 em reverso
- “TURN RIGHT” - Vira para à direita ligando os motores 1 e 4 para frente e os motores 0 e 3 em reverso
- “TURN LEFT” - Vira para à esquerda ligando os motores 0 e 3 para frente e os motores 1 e 4 em reverso.

```
def motors_control(action):
    if action == "UP":
        pi.set_servo_pulsewidth(PINS[2], 1900)
        pi.set_servo_pulsewidth(PINS[5], 1900)
    elif action == "DOWN":
```

```
        pi.set_servo_pulsewidth(PINS[2], 1100)
        pi.set_servo_pulsewidth(PINS[5], 1100)
    else:
        pi.set_servo_pulsewidth(PINS[2], 1500)
        pi.set_servo_pulsewidth(PINS[5], 1500)
    if action == "FORWARD":
        pi.set_servo_pulsewidth(PINS[0], 1100)
        pi.set_servo_pulsewidth(PINS[1], 1100)
        pi.set_servo_pulsewidth(PINS[3], 1900)
        pi.set_servo_pulsewidth(PINS[4], 1900)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
    if action == "BACK":
        pi.set_servo_pulsewidth(PINS[0], 1900)
        pi.set_servo_pulsewidth(PINS[1], 1900)
        pi.set_servo_pulsewidth(PINS[3], 1100)
        pi.set_servo_pulsewidth(PINS[4], 1100)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
    if action == "RIGHT":
        pi.set_servo_pulsewidth(PINS[0], 1100)
        pi.set_servo_pulsewidth(PINS[1], 1900)
        pi.set_servo_pulsewidth(PINS[3], 1900)
        pi.set_servo_pulsewidth(PINS[4], 1100)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
    if action == "LEFT":
        pi.set_servo_pulsewidth(PINS[0], 1900)
        pi.set_servo_pulsewidth(PINS[1], 1100)
        pi.set_servo_pulsewidth(PINS[3], 1100)
        pi.set_servo_pulsewidth(PINS[4], 1900)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
```

```
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
    if action == "TURN RIGHT":
        pi.set_servo_pulsewidth(PINS[0], 1100)
        pi.set_servo_pulsewidth(PINS[1], 1900)
        pi.set_servo_pulsewidth(PINS[3], 1100)
        pi.set_servo_pulsewidth(PINS[4], 1900)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
    if action == "TURN LEFT":
        pi.set_servo_pulsewidth(PINS[0], 1900)
        pi.set_servo_pulsewidth(PINS[1], 1100)
        pi.set_servo_pulsewidth(PINS[3], 1900)
        pi.set_servo_pulsewidth(PINS[4], 1100)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
```

Ao fim é definida a função de finalização do PWM, onde o *dutycycle* é definido como 0 para cada um dos pinos e o objeto *pi* é parado com a função *stop()*.

```
def finish():
    for pin in PINS:
        pi.set_PWM_dutycycle(pin, 0)
    pi.stop()
```

3.3. DIVISÃO ENTRE EIXO Z E PLANO XY

Semana do dia 11 de fevereiro de 2024:

Depois da reunião do dia 06 de fevereiro de 2024 ficou determinado que a função de controle dos motores deveria receber dois argumentos, um para controle do eixo Z e outro para o plano XY. Deste modo, as únicas alterações no código são o recebimento de mais um argumento e modificações nos ifs para separar as movimentações.

```
def motors_control(action_z, action_xy):
    if action_z == "UP":
        pi.set_servo_pulsewidth(PINS[2], 1900)
        pi.set_servo_pulsewidth(PINS[5], 1900)
    elif action_z == "DOWN":
        pi.set_servo_pulsewidth(PINS[2], 1100)
        pi.set_servo_pulsewidth(PINS[5], 1100)
    else:
        pi.set_servo_pulsewidth(PINS[2], 1500)
        pi.set_servo_pulsewidth(PINS[5], 1500)

    if action_xy == "FORWARD":
        pi.set_servo_pulsewidth(PINS[0], 1100)
        pi.set_servo_pulsewidth(PINS[1], 1100)
        pi.set_servo_pulsewidth(PINS[3], 1900)
        pi.set_servo_pulsewidth(PINS[4], 1900)
    else:
        pi.set_servo_pulsewidth(PINS[0], 1500)
        pi.set_servo_pulsewidth(PINS[1], 1500)
        pi.set_servo_pulsewidth(PINS[3], 1500)
        pi.set_servo_pulsewidth(PINS[4], 1500)
    if action_xy == "BACK":
      . . . . (Continua)
```

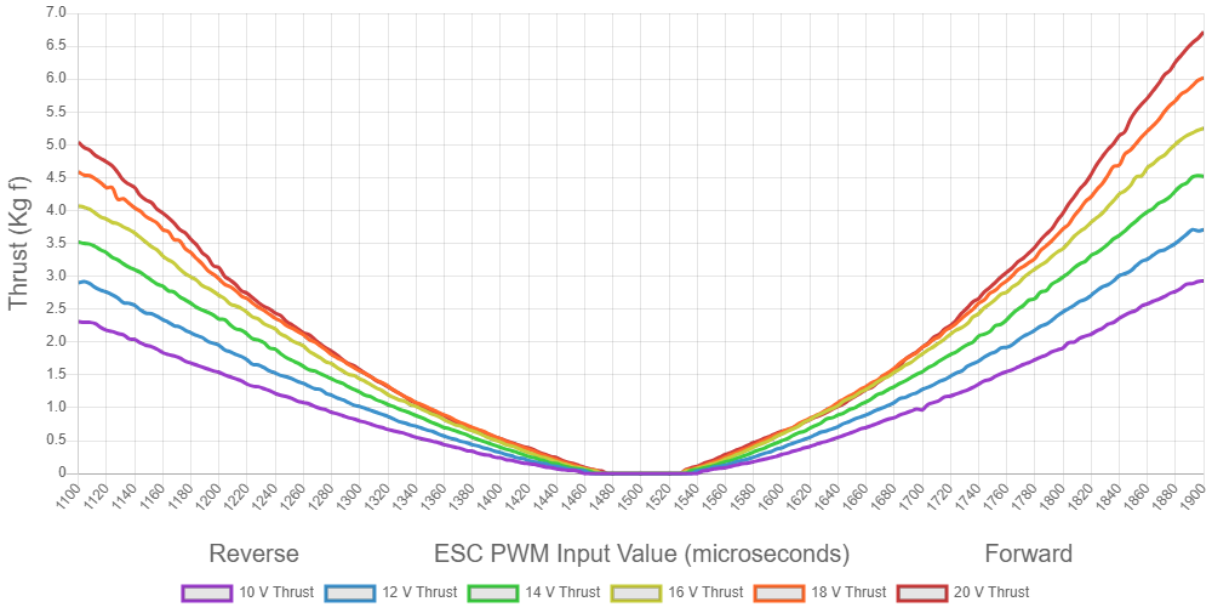
3.4. CONTROLE DA INTENSIDADE DOS MOVIMENTOS

Essa atualização tem por objetivo receber um dicionário como argumento na função de controle dos motores e nesse dicionário a *chave* representa a decisão, como “subir” ou “virar a esquerda”, e o *valor* é a intensidade que esse movimento vai ocorrer, sendo passado de forma percentual. Deste modo, não só é possível passar a instrução, mas também como ela será executada, podendo ser devagar ou com a força máxima dos motores. Além disso, também é possível passar várias instruções por vez, mas este caso ainda precisa ser estudado, pois se são instruções que agem nos mesmos motores a última vai sobrepor as instruções antecessoras.

Desta forma, para fazer a implementação duas funções foram criadas, a *convert_forward()* e a *convert_reverse()*. Essas funções tem como objetivo transformar a intensidade (percentual) no valor em microsegundos correspondente, mas existe um pequeno problema aqui, o valor em porcentagem pode se referir ao intervalo de valores em microsegundos, como 50% de 1100 a 1500 (para trás) que seria 1300, ou 50% da força do

motor, que se torna mais complicado de implementar já que a relação *Pulso PWM - Força* não é linear.

Figura 2 - Relação pulso PWM - Força



Fonte: https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/

Assim, o código de controle possui duas diferenças em relação ao anterior. A primeira é que é necessário pegar a chave e o valor em variáveis separadas usando *actions.items()*, onde *actions* é o dicionário recebido por parâmetro. E a outra alteração é que o valor do pulso passado para o motor é antes calculado pelas funções citadas acima.

```
def motors_control(actions):
      for action, value in actions.items():
              forward_value = convert_forward(value)
              reverse_value = convert_reverse(value)

              if action == "UP":
                  pi.set_servo_pulsewidth(PINS[2], forward_value)
                  pi.set_servo_pulsewidth(PINS[5], forward_value)
              elif action == "DOWN":
                  pi.set_servo_pulsewidth(PINS[2], reverse_value)
                  pi.set_servo_pulsewidth(PINS[5], reverse_value)
              else:
                  pi.set_servo_pulsewidth(PINS[2], rest_value)
                  pi.set_servo_pulsewidth(PINS[5], rest_value)
```

```
            if action == "FORWARD":
                pi.set_servo_pulsewidth(PINS[0], reverse_value)
                pi.set_servo_pulsewidth(PINS[1], reverse_value)
                pi.set_servo_pulsewidth(PINS[3], forward_value)
                pi.set_servo_pulsewidth(PINS[4], forward_value)
    . . . . (continua)
```

Além das funções, também foi adicionada uma variável com o valor de repouso para substituir nas partes do código onde é utilizado 1500 µs.

```
# value for the motor to stop
rest_value = 1500
```

OBS: Um problema no código foi encontrado enquanto as funções eram testadas. O objetivo era que o os motores parassem se o comando enviado não entrasse no if, mas se o comando é "FRONT", ao passar pelos ifs o código liga os motores de acordo ao passar pela condição em que "FRONT" é verdadeiro e quando passar pelos próximos ifs com as execuções de outras instruções os motores param, pois caem no else que desliga os mesmos motores que estavam executando a instrução "FRONT". Sendo assim, os elses foram retirados para resolver o problema e um if que recebe "STOP" foi adicionado para desligar os motores do AUV.

```
def motors_control(actions):
    for action, value in actions.items():
        forward_value = convert_forward(value)
        reverse_value = convert_reverse(value)

        if action == "UP":
            pi.set_servo_pulsewidth(PINS[2], forward_value)
            pi.set_servo_pulsewidth(PINS[5], forward_value)
        elif action == "DOWN":
            pi.set_servo_pulsewidth(PINS[2], reverse_value)
            pi.set_servo_pulsewidth(PINS[5], reverse_value)

        if action == "FORWARD":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
```

```
        elif action == "BACK":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        elif action == "RIGHT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        elif action == "LEFT":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        elif action == "TURN RIGHT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        elif action == "TURN LEFT":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        elif action == "STOP":
            pi.set_servo_pulsewidth(PINS[0], rest_value)
            pi.set_servo_pulsewidth(PINS[1], rest_value)
            pi.set_servo_pulsewidth(PINS[2], rest_value)
            pi.set_servo_pulsewidth(PINS[3], rest_value)
            pi.set_servo_pulsewidth(PINS[4], rest_value)
            pi.set_servo_pulsewidth(PINS[5], rest_value)
```

3.4.1. Controle em relação ao intervalo de microsegundos

A implementação usando o intervalo em microssegundos é simples, basicamente o percentual é convertido em microssegundos sabendo que para cada sentido de rotação do motor existe um intervalo de 400 µs seguindo a relação em que 0% é 0 µs e 100% é 400 µs.

$$Intervalo = 1900 - 1100 = 800\,\mu s$$

$$Intervalo\ "frente\ /\ trás"\ = \frac{Intervalo}{2}\ =\ 400\ \mu s$$

$$PWM\ "frente"\ =\ (1500\ +\ \frac{percent\_value}{100} \cdot 400)\ \mu s$$

$$PWM\ "trás"\ =\ (1500\ -\ \frac{percent\_value}{100} \cdot 400)\ \mu s$$

Assim, a diferença entre as duas funções é que uma soma esse valor a 1500 µs (valor de repouso) para que o motor "se mova para frente", e a outra subtrai o valor obtido de 1500 µs para que o motor "se mova para trás".

```
def convert_forward(value):
    value_micro = value * 4

    return (1500 + value_micro)

def convert_reverse(value):
    value_micro = value * 4

    return (1500 - value_micro)
```

3.4.2. Implementação em relação a força usada

Semana do dia 19 de maio de 2024:

Utilizando conceitos da disciplina de cálculo numérico será utilizado o método da Spline Cúbica para encontrar um PWM correspondente a um percentual de força do motor. Desta forma, o algoritmo primeiro encontra a matriz de coeficientes das splines usando *ipo.spline(pwm_values, kgf_values)* e, dentro das funções que convertem o percentual para o PWM correspondente, é feito o cálculo de qual é a força buscada, sendo que a potência máxima para frente é 3.71 Kgf e para trás é 2.91 Kgf.

$$Kgf_{frente}\ =\ value \cdot \frac{3.71}{100}$$

$$Kgf_{trás}\ =\ value \cdot \frac{2.91}{100}$$

```
import pigpio
import time
import interpolacao as ipo

FREQUENCY = 50

# Pins motors
PINS = [1, 2, 3, 4, 5, 6]
# 0 - Front left
# 1 - Front right
# 2 - Middle right
# 3 - Middle left
# 4 - Back right
# 5 - Back left

# value for the motor to stop
rest_value = 1500

pwm_values = [1100, 1120, 1140, 1160, 1180, 1200, 1220, 1240, 1260,
1280, 1300, 1320, 1340, 1360, 1380, 1400, 1420, 1440, 1460, 1480, 1500,
1520, 1540, 1560, 1580, 1600, 1620, 1640, 1660, 1680, 1700, 1720, 1740,
1760, 1780, 1800, 1820, 1840, 1860, 1880, 1900]

kgf_values = [2.9, 2.76, 2.56, 2.34, 2.14, 1.95, 1.73, 1.53, 1.37,
1.19, 1.02, 0.87, 0.72, 0.57, 0.44, 0.32, 0.2, 0.11, 0.04, 0, 0, 0,
0.04, 0.13, 0.25, 0.39, 0.55, 0.71, 0.89, 1.08, 1.28, 1.48, 1.71, 1.92,
2.18, 2.46, 2.71, 3.01, 3.26, 3.5, 3.71]

coef = ipo.spline(pwm_values, kgf_values)

. . . (continua)
```

Feito isto é encontrado o índice *i* do valor de força correspondente ao requisitado, percorrendo o vetor com as forças amostradas em *kgf_values*, e gerado um vetor com 20 valores de PWM a partir do valor em *pwm_values* com o mesmo índice *i* encontrado inicialmente. Com o valor de PWM no índice encontrado por *kgf_values*, que é o primeiro valor da spline que vai ser usada nos cálculos, o vetor *x* com os 20 valores de PWM e os coeficientes das splines no índice *i* encontrado, pode ser feito o cálculo dos valores de força pelo método das splines, sendo que o retorno será os valores de força para cada um dos 20

PWMs enviados por parâmetro. Com isso é verificado o índice do valor de Kgf calculado que corresponde a força desejada e é retornado o valor de PWM do vetor *x* com o mesmo índice.

```
def convert_forward(value):
    kgf = value * 0.0371
    i = 20

    while kgf_values[i] < kgf:
        i += 1
    if i != 20:
        i -= 1

    x = []

    for j in range(21):
        x.append(pwm_values[i] + j)

    y = ipo.calc_spline(pwm_values[i], x, coef[i])

    i = 0

    while y[i] < kgf:
        i += 1

    return x[i]

def convert_reverse(value):
    kgf = value * 0.0289

    i = 20

    while kgf_values[i] < kgf:
        i -= 1

    x = []

    for j in range(21):
        x.append(pwm_values[i] + j)

    y = ipo.calc_spline(pwm_values[i], x, coef[i])

    i = 0
```

```
    while y[i] > kgf:
        i += 1

    return x[i]
```

Depois de aplicada essa implementação surgiu a indagação de que calcular todos esses valores em tempo de execução poderia tornar o código lento, logo uma alteração nesta implementação foi feita. Na modificação, os cálculos de todos os valores de PWM correspondentes a cada percentual, de 0 a 100 para frente e para trás, são feitos e salvos em um vetor, assim os valores utilizam somente armazenamento, armazenando 202 valores de PWM, sem fazer cálculos em execução.

Fazendo isso o código ficou da seguinte maneira:

```
import pigpio
import time

FREQUENCY = 50

# Pins motors
PINS = [1, 2, 3, 4, 5, 6]
# 0 - Front left
# 1 - Front right
# 2 - Middle right
# 3 - Middle left
# 4 - Back right
# 5 - Back left

# value for the motor to stop
rest_value = 1500

pwm_values  = [1500, 1464, 1455, 1447, 1439, 1432, 1426, 1420, 1415,
1410, 1405, 1401, 1396, 1391, 1386, 1382, 1377, 1372, 1368, 1363, 1359,
1355, 1352, 1348, 1344, 1340, 1336, 1333, 1329, 1325, 1321, 1317, 1313,
1309, 1305, 1302, 1298, 1294, 1291, 1288, 1284, 1281, 1278, 1275, 1272,
1268, 1265, 1262, 1258, 1255, 1251, 1247, 1244, 1240, 1237, 1234, 1231,
1228, 1225, 1223, 1220, 1218, 1215, 1212, 1210, 1207, 1205, 1202, 1199,
1196, 1193, 1190, 1187, 1184, 1181, 1178, 1175, 1172, 1169, 1166, 1163,
1160, 1158, 1155, 1152, 1150, 1147, 1145, 1142, 1139, 1137, 1134, 1131,
1128, 1125, 1122, 1119, 1115, 1111, 1107, 1102, 1500, 1540, 1549, 1557,
1564, 1570, 1576, 1582, 1588, 1593, 1598, 1603, 1608, 1612, 1617, 1621,
1626, 1631, 1635, 1640, 1644, 1648, 1653, 1657, 1661, 1665, 1668, 1672,
```

```
1676, 1680, 1684, 1688, 1691, 1695, 1699, 1702, 1706, 1710, 1714, 1717,
1721, 1724, 1727, 1731, 1734, 1737, 1740, 1744, 1747, 1751, 1754, 1758,
1761, 1764, 1767, 1770, 1773, 1776, 1779, 1781, 1784, 1786, 1789, 1791,
1794, 1797, 1800, 1803, 1806, 1809, 1812, 1815, 1818, 1820, 1823, 1825,
1828, 1830, 1833, 1835, 1838, 1840, 1843, 1846, 1849, 1852, 1855, 1858,
1861, 1864, 1867, 1870, 1873, 1876, 1879, 1883, 1886, 1890, 1893, 1897,
1900]

pi = pigpio.pi()
# #
# # while(not pi.connect()):
# #     pi = pigpio.pi()

def inicialize_pins():
    for pin in PINS:
        pi.set_mode(pin, pigpio.OUTPUT)
        pi.set_PWM_frequency(pin, FREQUENCY)
        pi.set_servo_pulsewidth(pin, rest_value)
    time.sleep(7)

def motors_control(actions):
    for action, value in actions.items():
        forward_value = convert_forward(value)
        reverse_value = convert_reverse(value)

        if action == "UP":
            pi.set_servo_pulsewidth(PINS[2], forward_value)
            pi.set_servo_pulsewidth(PINS[5], forward_value)
        elif action == "DOWN":
            pi.set_servo_pulsewidth(PINS[2], reverse_value)
            pi.set_servo_pulsewidth(PINS[5], reverse_value)

        if action == "FRONT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        if action == "BACK":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        if action == "RIGHT":
```

```
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        if action == "LEFT":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        if action == "TURN RIGHT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        if action == "TURN LEFT":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        if action == "STOP":
            pi.set_servo_pulsewidth(PINS[0], rest_value)
            pi.set_servo_pulsewidth(PINS[1], rest_value)
            pi.set_servo_pulsewidth(PINS[2], rest_value)
            pi.set_servo_pulsewidth(PINS[3], rest_value)
            pi.set_servo_pulsewidth(PINS[4], rest_value)
            pi.set_servo_pulsewidth(PINS[5], rest_value)

def convert_forward(value):
    return pwm_values[101 + value]

def convert_reverse(value):
    return pwm_values[value]

def finish():
    for pin in PINS:
        pi.set_PWM_dutycycle(pin, 0)
    pi.stop()
```

Para testar se calcular os valores antes realmente faz diferença um teste foi feito exibindo o valor percentual e a correspondência em PWM nos dois códigos, o que calcula esses valores toda vez que um percentual é enviado e o que tem esses valores armazenados em um vetor.

Código de teste:

```
for i in range(1000):
        for i in range(101):
            print(f"{i}: {convert_reverse(i)}")
        for i in range(101):
            print(f"{i}: {convert_forward(i)}")
```

Conseguindo como resultado para o código que calcula os valores:

```
real    0m37.284s
user    0m0.000s
sys     0m0.091s
```

E para o código com os valores armazenados:

```
real    0m24.741s
user    0m0.015s
sys     0m0.076s
```

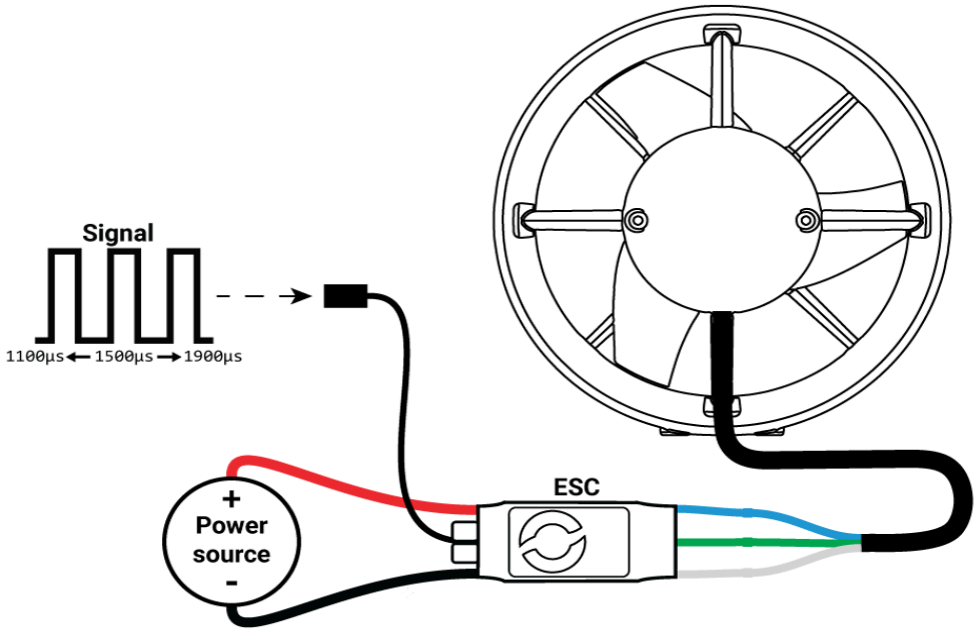
Assim, percebe-se que o código com os valores armazenados tem um melhor desempenho, e portanto será mantido se nenhum problema for encontrado posteriormente.

Em relação ao método usando se, por exemplo, for requerido 10% da potência do motor para trás, sendo que nessa direção a potência máxima é 2.9 KGf, obtem-se 0.29 KGf através de uma regra de 3 simples, e ao buscar por 10% usando a função *convert_reverse* obtem-se um PWM de 1405 µs, que é 1 µs acima do PWM relacionado a força de 0.29 KGf na planilha de valores da Blue Robotics que relaciona PWM com Kgf.

4. TESTES

Após o desenvolvimento dos códigos de controle, foram executados testes com o motor para garantir o funcionamento esperado. Para isso, foi seguido o esquema de ligação fornecido pela Blue Robotics para a ligação do motor com o ESC e com a Raspberry Pi.

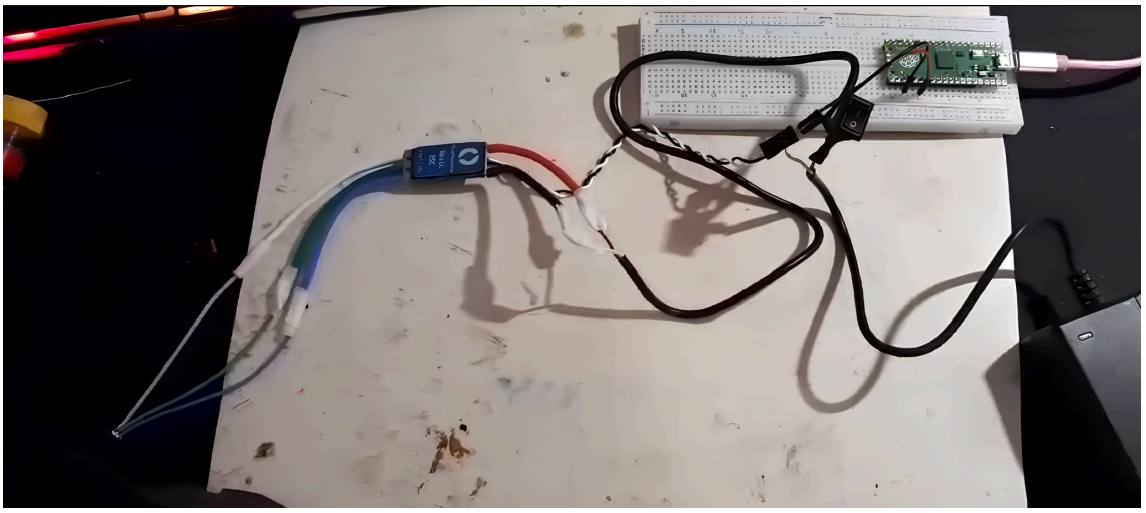
Figura 3 - Esquema de ligação do motor



Fonte: https://bluerobotics.com/learn/thruster-usage-guide/

O objetivo era fazer os testes na Raspberry Pi 4, mas no momento em que os testes seriam feitos a mesma estava sem sistema, então foi usada a Raspberry Pi Pico. Assim, a ligação dos componentes ficou a seguinte:

Imagem 1 - Ligação do motor, ESC e Raspberry Pi Pico



Fonte: Hebert Alan Kubis

Imagem 2 - Ambiente de teste do motor

Fonte: Hebert Alan Kubis

Como fonte de alimentação foi usada uma fonte de notebook de 12 V e 3 A, o que não atende às especificações do motor, que exige uma fonte de energia que possa fornecer 30 A, mas limitando a potência máxima em 30% a corrente máxima que a fonte pode fornecer não é ultrapassada.

Semana do dia 26 de maio de 2024:

Como consequência de trocar de Raspberry, o código teve que ser adaptado para rodar na RaspBerry Pi Pico. Desta forma, foi feito um código para testar apenas um motor, mas seguindo a mesma lógica do outro.

No código são usados algumas constantes que surgem de alguns cálculos feitos anteriormente. Inicialmente, deve-se compreender que a função *duty_u16* recebe um valor entre 0 e 65536 (*duty*) que representa valores entre 0 % e 100 %. Em seguida é necessário encontrar o valor de *duty* que representa o ponto inicial e final do intervalo útil para os motores, que no caso são 1100 µs e 1900 µs, respectivamente, com base no período da onda. O período é calculado abaixo:

$$T = \frac{1}{F} \cdot 1000000 = (\frac{1000000}{F})\ \mu s$$

Onde T é o período, em microssegundos, em função de F, que é a frequência usada. O valor 1000000 é usado para transformar segundos em microssegundos e a conta é deixada em termos função de F para o código se adequar a qualquer frequência usada.

Com o período podemos encontrar o percentual equivalente aos pontos final e inicial, ou seja, quanto tempo a onda está em estado ativado em relação ao seu período. Sendo esses valores encontrados da seguinte forma:

$$X_0 = \frac{1100}{T} = \frac{1100}{\frac{1000000}{F}} = \frac{1100 \cdot F}{1000000} = 0.0011 \cdot F$$

$$X_1 = \frac{1900}{T} = \frac{1900}{\frac{1000000}{F}} = \frac{1900 \cdot F}{1000000} = 0.0019 \cdot F$$

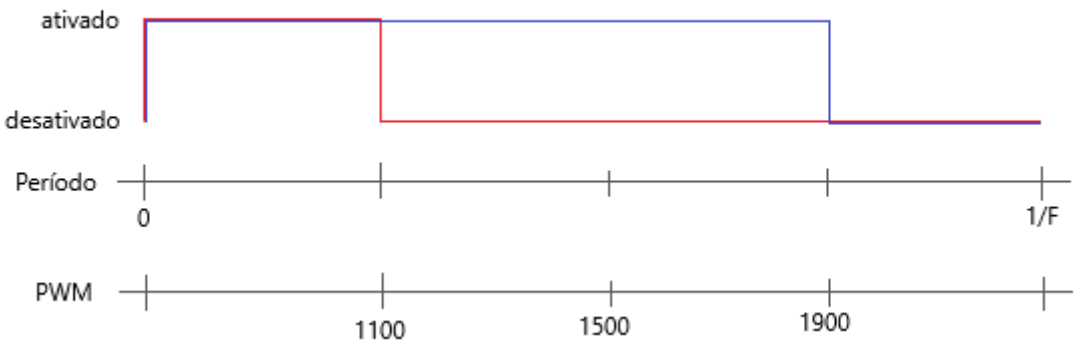
Onde X_0 é o percentual do período equivalente ao começo do intervalo de PWM do motor (1100 μs) e X_1 é o percentual do período equivalente ao final do intervalo de PWM do motor (1900 μs).

Com esses valores será definido um intervalo que será dividido por 2 para poder se referir em movimentos para frente e para trás, que fica da seguinte maneira:

$$Intervalo = \frac{X_1 - X_0}{2} = \frac{0.0008 \cdot F}{2} = 0.0004 \cdot F$$

O que está sendo feito pode parecer um pouco complicado de compreender somente com os cálculos e explicações, mas visualmente foi feito o seguinte:

Figura 4 - relação onda, período e PWM



Fonte: Hebert Alan Kubis

Em vermelho é quanto tempo o pulso deve estar ativado para que o reverso do motor esteja em potência máxima e em azul é quanto o pulso deve estar ativado para ativar o motor em potência máxima para frente. Cada um desses pontos, 1100 e 1900, representa um percentual do período de onda, isto que foi obtido em X_0 e X_1, e a diferença entre esses valores é um intervalo percentual, mas um percentual para frente e para trás. Como o intervalo é simétrico, com 1500, que é o repouso, exatamente na metade do intervalo, é possível obter "quantos por cento" do período tem para cada lado do valor de repouso.

Com isto, ao receber um valor de *percent_value* a função vai encontrar esse percentual em relação a metade do intervalo total de valores de PWM, referente assim a movimentação para frente ou para trás do motor. Este valor percentual é calculado abaixo:

$$percentual\_intervalo = \frac{intervalo \cdot percent\_value}{100} = \frac{0.0004 \cdot F \cdot percent\_value}{100}$$

Onde o $percentual\_intervalo$ é dividido por 100 pois *percent_value* é passado como, por exemplo, 50 para se referir a 50% ou 0.5.

Com esses valores é possível prosseguir e calcular o *duty* para o *convert_forward* e para o *convert_reverse*. Calculando primeiro para o *convert_forward*, $percentual\_intervalo$ deve ser somado ao percentual que representa 1500 μs, isto pode ser feito somando o percentual que representa o início do intervalo maior a *intervalo*, que é a metade do intervalo maior. O resultado dessa conta é um percentual em função de *F* e *percent_value*, que será multiplicado por 65536, que é o valor máximo do *duty*.

$$duty = (0.000004 \cdot F \cdot percent\_value + (X_0 + intervalo)) \cdot 65536$$

$$duty = 0.262144 \cdot F \cdot percent\_value + (\frac{X_0 + X_1}{2}) \cdot 65536$$

$$duty = 0.262144 \cdot F \cdot percent\_value + (\frac{0.0011 \cdot F + 0.0019 \cdot F}{2}) \cdot 65536$$

$$duty = 0.262144 \cdot F \cdot percent\_value + 98.304 \cdot F$$

Para a função *convert_reverse* é a mesma conta, a diferença é que o *percentual_intervalo* é subtraído do percentual que representa o meio do intervalo.

$$duty = ((X_0 + intervalo) - 0.000004 \cdot F \cdot percent\_value) \cdot 65536$$

$$duty = 98.304 \cdot F - 0.262144 \cdot F \cdot percent\_value$$

O valor de 98.304 foi arredondado para 98.35 no código para melhores resultados.

No código também existe uma variável *value_init* que é o valor de 1500 µs para qualquer frequência usada, e consiste em usar *percent_value* como sendo 0.

$$value\_init = duty = 0.262144 \cdot F \cdot 0 + 98.304 \cdot F$$

$$value\_init = 98.304 \cdot F$$

```
from machine import Pin, PWM
from utime import sleep

FREQ = 200
PIN = 20

motor1 = PWM(Pin(PIN), freq = FREQ)

def init_esc():
    print("Starting . . .")
    value_init = int(98.3144 * FREQ) # Equivalente a 1500 us
    print(int((15.2587890625 * value_init) / FREQ))
    motor1.duty_u16(value_init)
    sleep(7)

def convert_forward(percent_value):
     duty = int(FREQ * (0.262144 * percent_value + 98.35)) # Encontra o
valor do duty cicle correspondente
      print(int((15.2587890625 * duty) / FREQ)) # Exibe a frequência
usada
    motor1.duty_u16(duty)

def convert_reverse(percent_value):
     duty = int(FREQ * (98.35 - 0.262144 * percent_value)) # Encontra o
valor do duty cicle correspondente
      print(int((15.2587890625 * duty) / FREQ)) # Exibe a frequência
usada
```

```
    motor1.duty_u16(duty)

def finalize_esc():
    print("Finalizando . . .")
    value_fin = int(98.3144 * FREQ)
    motor1.duty_u16(value_fin)
    sleep(2)
    motor1.duty_u16(0)
    sleep(2)

# Test
init_esc()

# NÃO PASSAR DE 30% COM A FONTE DE 3A

convert_forward(30)
sleep(10)
convert_forward(0)
sleep(3)
convert_reverse(30)
sleep(10)

finalize_esc()
```

A diferença deste código para o outro é que não há um vetor com os motores, pois como a ideia era somente testar se o motor funcionaria, o código acima age diretamente no objeto motor, mas pode ser adaptado para que funcione como o último código de controle desenvolvido.

Em relação aos testes, estes foram iniciados com 5% da potência, a qual não mantém o motor ligado constantemente, nessa faixa de potência o motor é ligado e desligado em ciclos. Após isso, a potência foi sendo aumentada gradativamente até chegar a 30%, que era o máximo com a fonte usada. Com isso os testes ocorreram bem, e garantiram que os códigos de controle estão funcionando.

5. CONCLUSÃO

Semana do dia 09 de junho de 2024:

Com tudo isso, abaixo segue o código de controle final usando a biblioteca *pigpio*:

```
import pigpio
import time

FREQUENCY = 200

# Pins motors
PINS = [1, 2, 3, 4, 5, 6]
# 0 - Front left
# 1 - Front right
# 2 - Middle right
# 3 - Middle left
# 4 - Back right
# 5 - Back left

# value for the motor to stop
rest_value = 1500

pwm_values = [1500, 1464, 1455, 1447, 1439, 1432, 1426, 1420, 1415,
1410, 1405, 1401, 1396, 1391, 1386, 1382, 1377, 1372, 1368, 1363, 1359,
1355, 1352, 1348, 1344, 1340, 1336, 1333, 1329, 1325, 1321, 1317, 1313,
1309, 1305, 1302, 1298, 1294, 1291, 1288, 1284, 1281, 1278, 1275, 1272,
1268, 1265, 1262, 1258, 1255, 1251, 1247, 1244, 1240, 1237, 1234, 1231,
1228, 1225, 1223, 1220, 1218, 1215, 1212, 1210, 1207, 1205, 1202, 1199,
1196, 1193, 1190, 1187, 1184, 1181, 1178, 1175, 1172, 1169, 1166, 1163,
1160, 1158, 1155, 1152, 1150, 1147, 1145, 1142, 1139, 1137, 1134, 1131,
1128, 1125, 1122, 1119, 1115, 1111, 1107, 1102, 1500, 1540, 1549, 1557,
1564, 1570, 1576, 1582, 1588, 1593, 1598, 1603, 1608, 1612, 1617, 1621,
1626, 1631, 1635, 1640, 1644, 1648, 1653, 1657, 1661, 1665, 1668, 1672,
1676, 1680, 1684, 1688, 1691, 1695, 1699, 1702, 1706, 1710, 1714, 1717,
1721, 1724, 1727, 1731, 1734, 1737, 1740, 1744, 1747, 1751, 1754, 1758,
1761, 1764, 1767, 1770, 1773, 1776, 1779, 1781, 1784, 1786, 1789, 1791,
1794, 1797, 1800, 1803, 1806, 1809, 1812, 1815, 1818, 1820, 1823, 1825,
1828, 1830, 1833, 1835, 1838, 1840, 1843, 1846, 1849, 1852, 1855, 1858,
1861, 1864, 1867, 1870, 1873, 1876, 1879, 1883, 1886, 1890, 1893, 1897,
1900]

pi = pigpio.pi()
```

```
# #
# # while(not pi.connect()):
# #     pi = pigpio.pi()

def inicialize_pins():
    for pin in PINS:
        pi.set_mode(pin, pigpio.OUTPUT)
        pi.set_PWM_frequency(pin, FREQUENCY)
        pi.set_servo_pulsewidth(pin, rest_value)
    time.sleep(7)

def motors_control(actions):
    """
      Function that takes an argument with movement instruction and
decides how the motors will be activated

    Parameters:

    - action       : action that should be execute

    actions:
     - "UP"          : move the AUV up, turning on the motors 3 and 6 in
the forward direction.
     - "DOWN"        : move the AUV down, turning on the motors 3 and 6
in the reverse direction.
     - "FRONT"       : move the AUV front, turning on the motors 1 and 2
in the reverse direction and the motors 4 and 5 in the forward
direction.
     - "BACK"        : move the AUV back, turning on the motors 1 and 2
in the forward direction and the motors 4 and 5 in the reverse
direction.
     - "RIGHT"       : move the AUV right, turning on the motors 2 and 4
in the forward direction and the motors 1 and 5 in the reverse
direction.
     - "LEFT"        : move the AUV left, turning on the motors 1 and 5
in the forward direction and the motors 2 and 4 in the reverse
direction.
     - "TURN RIGHT" : turn the AUV right, turning on the motors 2 and 5
in the forward direction and the motors 1 and 4 in the reverse
direction.
     - "TURN LEFT"  : turn the AUV left, turning on the motors 1 and 4
in the forward direction and the motors 2 and 5 in the reverse
direction.
```

```
    Return:
    None
    """

    for action, value in actions.items():
        forward_value = convert_forward(value)
        reverse_value = convert_reverse(value)

        if action == "UP":
            pi.set_servo_pulsewidth(PINS[2], forward_value)
            pi.set_servo_pulsewidth(PINS[5], forward_value)
        elif action == "DOWN":
            pi.set_servo_pulsewidth(PINS[2], reverse_value)
            pi.set_servo_pulsewidth(PINS[5], reverse_value)

        if action == "FRONT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        if action == "BACK":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        if action == "RIGHT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        if action == "LEFT":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        if action == "TURN RIGHT":
            pi.set_servo_pulsewidth(PINS[0], reverse_value)
            pi.set_servo_pulsewidth(PINS[1], forward_value)
            pi.set_servo_pulsewidth(PINS[3], reverse_value)
            pi.set_servo_pulsewidth(PINS[4], forward_value)
        if action == "TURN LEFT":
            pi.set_servo_pulsewidth(PINS[0], forward_value)
```

```
            pi.set_servo_pulsewidth(PINS[1], reverse_value)
            pi.set_servo_pulsewidth(PINS[3], forward_value)
            pi.set_servo_pulsewidth(PINS[4], reverse_value)
        if action == "STOP":
            pi.set_servo_pulsewidth(PINS[0], rest_value)
            pi.set_servo_pulsewidth(PINS[1], rest_value)
            pi.set_servo_pulsewidth(PINS[2], rest_value)
            pi.set_servo_pulsewidth(PINS[3], rest_value)
            pi.set_servo_pulsewidth(PINS[4], rest_value)
            pi.set_servo_pulsewidth(PINS[5], rest_value)

def convert_forward(value):
    return pwm_values[101 + value]

def convert_reverse(value):
    return pwm_values[value]

def finish():
    """
    Finish the motors
    """
    for pin in PINS:
        pi.set_PWM_dutycycle(pin, 0)
    pi.stop()

# Test
if __name__ == "__main__":
    # while True:
    #     motors_control({"DOWN": 80, "UP": 60})
    #     time.sleep(1)

    for i in range(0, 10):
        for i in range(101):
            print(f"{i}: {convert_reverse(i)}")
        for i in range(101):
            print(f"{i}: {convert_forward(i)}")
```

E, por último, o código de controle que age exatamente do mesmo modo que o anterior, mas feito para rodar na RaspBerry Pi Pico:

```
from machine import Pin, PWM
from utime import sleep
```

```
FREQ = 200 # 50 - 400 Hz

PINS = [16, 17, 18, 19, 20, 21]
# 0 - Front left
# 1 - Front right
# 2 - Middle right
# 3 - Middle left
# 4 - Back right
# 5 - Back left

motores = []

rest_value = int(98.3144 * FREQ) # Equivalente a 1500 us

def init_esc():
    """
     Inicializa o ESC, passa um PWM de 1500 us e mantém por 7 segundos, que é o tempo que o ESC
    precisa, para então o motor estar pronto para ser usado
    """

    print("Starting . . .")

    for i in PINS:
        motor = PWM(Pin(i), freq = FREQ)
        motor.duty_u16(rest_value)
        motores.append(motor)

      print(int((15.2587890625 * rest_value) / FREQ)) # shows the PWM used

    sleep(7)

def motors_control(actions):
    """
      Function that takes an argument with movement instruction and decides how the motors will be activated

    Parameters:

    - action       : action that should be execute
```

```
    actions:
     - "UP"          : move the AUV up, turning on the motors 3 and 6 in
the forward direction.
     - "DOWN"        : move the AUV down, turning on the motors 3 and 6
in the reverse direction.
     - "FRONT"       : move the AUV front, turning on the motors 1 and 2
in the reverse direction and the motors 4 and 5 in the forward
direction.
     - "BACK"        : move the AUV back, turning on the motors 1 and 2
in the forward direction and the motors 4 and 5 in the reverse
direction.
     - "RIGHT"       : move the AUV right, turning on the motors 2 and 4
in the forward direction and the motors 1 and 5 in the reverse
direction.
     - "LEFT"        : move the AUV left, turning on the motors 1 and 5
in the forward direction and the motors 2 and 4 in the reverse
direction.
     - "TURN RIGHT" : turn the AUV right, turning on the motors 2 and 5
in the forward direction and the motors 1 and 4 in the reverse
direction.
     - "TURN LEFT"  : turn the AUV left, turning on the motors 1 and 4
in the forward direction and the motors 2 and 5 in the reverse
direction.

    Return:
    None
    """

    for action, value in actions.items():
        forward_value = convert_forward(value)
        reverse_value = convert_reverse(value)

        if action == "UP":
            motores[2].duty_u16(forward_value)
            motores[5].duty_u16(forward_value)
        elif action == "DOWN":
            motores[2].duty_u16(reverse_value)
            motores[5].duty_u16(reverse_value)

        if action == "FRONT":
            motores[0].duty_u16(reverse_value)
            motores[1].duty_u16(reverse_value)
            motores[3].duty_u16(forward_value)
```

```
            motores[4].duty_u16(forward_value)
        if action == "BACK":
            motores[0].duty_u16(forward_value)
            motores[1].duty_u16(forward_value)
            motores[3].duty_u16(reverse_value)
            motores[4].duty_u16(reverse_value)
        if action == "RIGHT":
            motores[0].duty_u16(reverse_value)
            motores[1].duty_u16(forward_value)
            motores[3].duty_u16(forward_value)
            motores[4].duty_u16(reverse_value)
        if action == "LEFT":
            motores[0].duty_u16(forward_value)
            motores[1].duty_u16(reverse_value)
            motores[3].duty_u16(reverse_value)
            motores[4].duty_u16(forward_value)
        if action == "TURN RIGHT":
            motores[0].duty_u16(reverse_value)
            motores[1].duty_u16(forward_value)
            motores[3].duty_u16(reverse_value)
            motores[4].duty_u16(forward_value)
        if action == "TURN LEFT":
            motores[0].duty_u16(forward_value)
            motores[1].duty_u16(reverse_value)
            motores[3].duty_u16(forward_value)
            motores[4].duty_u16(reverse_value)
        if action == "STOP":
            motores[0].duty_u16(rest_value)
            motores[1].duty_u16(rest_value)
            motores[2].duty_u16(rest_value)
            motores[3].duty_u16(rest_value)
            motores[4].duty_u16(rest_value)
            motores[5].duty_u16(rest_value)

def convert_forward(percent_value):
    """
      Recebe um valor de 0% a 100% e converte em uma valor de PWM
equivalente entre 1500us e 1900us para que o
    motor "gire para frente"

    Input:
    - percent_value: valor entre 0 e 100 %
    """
```

```
    duty = int(FREQ * (0.262144 * percent_value + 98.35)) # Encontra o
valor do duty cicle correspondente
    print(int((15.2587890625 * duty) / FREQ)) # Exibe o valor do pulso
    return duty

def convert_reverse(percent_value):
    """
      Recebe um valor de 0% a 100% e converte em uma valor de PWM
equivalente entre 1500us e 1100us para que o
    motor "gire para trás"

    Input:
    - percent_value: valor entre 0 e 100 %
    """

    duty = int(FREQ * (98.35 - 0.262144 * percent_value)) # Encontra o
valor do duty cicle correspondente
    print(int((15.2587890625 * duty) / FREQ)) # Exibe o valor do pulso
    return duty

def finalize_esc():
    """
     Finaliza a execução do ESC, passa um PWM de 1500 us para que o
motor pare
    """

    print("Finishing . . .")

    for motor in motores:
        motor.duty_u16(rest_value)
        motor.duty_u16(0)

if __name__ == "__main__":
    """
    Algoritmo para testar o motor usando micropython
    """

    init_esc()

    motors_control({"UP": 30})
    sleep(100)
    motors_control({"FRONT": 40})
```

```
    sleep(30)
    motors_control({"LEFT": 20})
    sleep(100)

    finalize_esc()
```

Todos os códigos feitos até o momento podem ser encontrados em: https://github.com/2005HAK/Terra

REFERÊNCIAS

T200 Thrusters. Blue Robotics, 2023. Disponível em: https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/. Acesso em: 21 de janeiro de 2024.

Basic ESC. Blue Robotics. Disponível em: https://bluerobotics.com/store/thrusters/speed-controllers/besc30-r3/. Acesso em: 21 de janeiro de 2024.

Current and PWM piloting T200. Blue Robotics, 2021. Disponível em: https://discuss.bluerobotics.com/t/current-and-pwm-piloting-t200/9913. Acesso em: 24 de janeiro de 2024.

Python Interface. PIGPIO Library, 2023. Disponível em: https://abyz.me.uk/rpi/pigpio/python.html. Acesso em: 24 de janeiro de 2024.

How can i run two t200 thrusters in raspberry pi4. Blue Robotics, 2021. Disponível em: https://discuss.bluerobotics.com/t/how-can-i-run-two-t200-thrusters-in-raspberry-pi4/10158. Acesso em: 24 de janeiro de 2024.

Operating Basic ESC from Raspberry Pi using RPi.GPIO. Blue Robotics, 2019. Disponível em: https://discuss.bluerobotics.com/t/operating-basic-esc-from-raspberry-pi-using-rpi-gpio/4704. Acesso em: 26 de janeiro de 2024.

How to run Brushless Motor with raspberry pi 4. RaspBerry Pi, 2023. Disponível em: https://raspberrypi.stackexchange.com/questions/142581/how-to-run-brushless-motor-with-raspberry-pi-4. Acesso em: 31 de janeiro de 2024.

Basic ESC with RaspBerry Pi. Blue Robotics, 2015. Disponível em: https://discuss.bluerobotics.com/t/basic-esc-with-raspberry-pi/137. Acesso em: 31 de janeiro de 2024.

Control T200, basic esc with python. Blue Robotics, 2020. Disponível em: https://discuss.bluerobotics.com/t/control-t200-basic-esc-with-python/7745. Acesso em: 31 de janeiro de 2024.

Controlling quadcopter brushless motors in Raspberry using pigpio. Raspberry Pi, 2018. Disponível em: https://forums.raspberrypi.com/viewtopic.php?t=220341. Acesso em: 31 de janeiro de 2024.

Thruster User Guide. Blue Robotics, 2024. Disponível em: https://bluerobotics.com/learn/thruster-usage-guide/. Acesso em: 21 de maio de 2024.

Operating Basic ESC from Raspberry PI using RPI.GPIO. Blue Robotics, 2019. Disponível em: https://discuss.bluerobotics.com/t/operating-basic-esc-from-raspberry-pi-using-rpi-gpio/4704/2. Acesso em: 21 de maio de 2024.