Cin-UFPE
IF826 - Robótica
Projeto 1: Braço robótico

Este projeto visa criar e implementar um sistema de controle para um braço robótico que realize tarefas de "Pick and Place"ou rastreamento de trajetória. O robô possui quatro graus de liberdade, conforme mostrado na figura. Cada grau de liberdade corresponde a uma rotação e é obtido por meio de servomotores, modelo MG996R. Os motores são controlados por um Raspberry Pi 4, que é um computador de placa única equipado com uma interface GPIO (General Purpose Input Output).

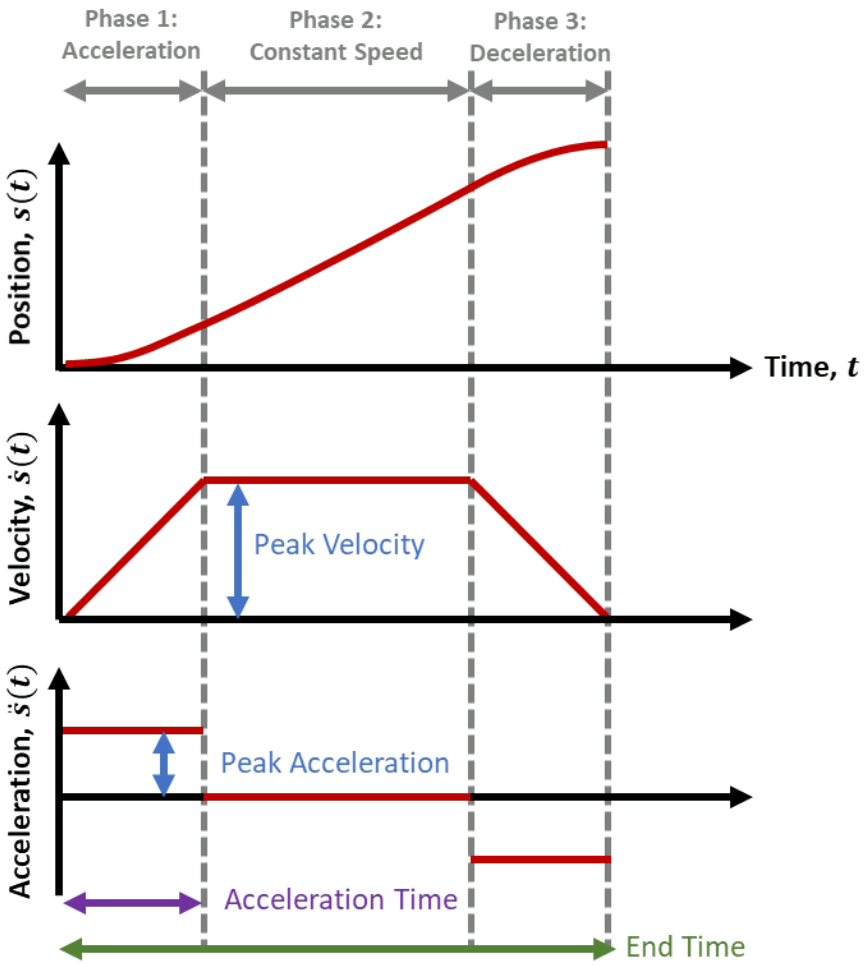
Objetivos do projeto:

- O sistema de controle deve ser implementado utilizando a linguagem Python e o middleware ROS2.
- O sistema contém dois nós:
  - Um nó localizado no computador do grupo, que calcula trajetórias no espaço de configuração e no espaço de tarefas.
  - Um nó localizado no Raspberry Pi 4, que executa as trajetórias por meio de um serviço.
- A comunicação entre os dois nós deve utilizar estruturas ROS.
- Trajetórias que façam com que a ferramenta saia do seu espaço de trabalho ou violem os limites dos atuadores não devem ser executadas.

1 Modelagem

Para iniciar o projeto, é necessário primeiro modelar o braço robótico, ou seja, calcular os modelos cinemáticos **direto** e **inverso**. Em primeiro lugar, recomenda-se definir a referência base, bem como as de cada juntas. Para estas últimas, isso significa definir a origem e o sentido de rotação.

2 Planejamento de trajetória



Para controlar o braço robótico é necessário implementar a geração de trajetória. As trajetórias podem ser calculadas no espaço de configuração ou no espaço Euclidiano.

- O planejamento no espaço de configuração deve utilizar o perfil trapezoidal apresentado na figura e respeitar os limites de velocidade V_{max} e de aceleração A_{max}.
- O planejamento do espaço Euclidiano consiste em uma interpolação linear entre os pontos de partida e chegada. A trajetória calculada no espaço Euclidiano deve ser convertida no espaço de configuração.

As trajetórias devem conter o mesmo número de pontos para todas as dimensões. Portanto, recomenda-se calcular primeiro a duração das trajetórias de cada junta. A maior duração será então usada para calcular as trajetórias de todas as juntas.

ROS2

- Criar um nó `planner`
- Implementar a função `p = FK(q)`
- Implementar a função `q = IK(q)`
- Implementar a função `move(p)` (planejamento no espaço das juntas)
- Implementar a função `moveS(p)` (planejamento no espaço do efetor terminal)

Comentários:

- q: vetor das posições angulares das juntas
- p: pose do efetor terminal
- `move` e `moveS` devem chamar um serviço cuja requesta contém um único campo do tipo `JointTrajectory`

3 Servomotor

3.1 Hardware

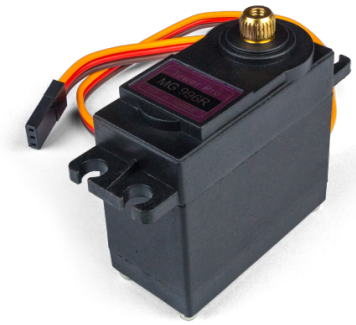


Figura 1: Servomotor TowerPro MG996R

O robô está equipado com um servomotor TowerPro MG996R que permite orientar a câmera. Estas características e o código colorido dos fios são dados nas Tab. 1 e 2

Tensão de operação	4,8 a 6,0 V
Tipo de Engrenagem	Metálica
Modulação	Digital
Velocidade de operação	0,19 seg/60 graus (4,8 V sem carga)
Velocidade de operação	0,15 seg/60 graus (6 V sem carga)
Torque (stall)	9,4 kgf.cm (4,8 V) e 11,0 kgf.cm (6 V)

Tabela 1: Código de cores dos fios do par motor-codificador

Marão	Fonte de alimentação negativa do motor(-) (0 V)
Vermelho	Fonte de alimentação positiva do motor(+) (4,8 - 6 V)
Amarelo	Sinal de controle PWM

Tabela 2: Código de cores dos fios do par motor-codificador

Os servos são controlados pela largura do pulso, a largura do pulso determina o ângulo. Uma largura de pulso de 1500 μs move o servo para o ângulo 0. Cada aumento de 10 μs na largura de pulso normalmente move o servo 1 grau mais no sentido horário. Cada diminuição de 10 μs na largura de pulso normalmente move o servo 1 grau mais no sentido anti-horário. Servos normalmente recebem 50 pulsos por segundo (50 Hz).
Alguns cálculos em 50 Hz para larguras de pulso de amostra:

$$500/20000 = 0.025 \text{ ou } 2.5\ \% \text{ dutycycle}$$

$$1000/20000 = 0.05 \text{ ou } 5.0\ \% \text{ dutycycle}$$

$$1500/20000 = 0.075 \text{ ou } 7.5\ \% \text{ dutycycle}$$

$$2000/20000 = 0.1 \text{ ou } 10.0\ \% \text{ dutycycle}$$

$$2500/20000 = 0.125 \text{ ou } 12.5\ \% \text{ dutycycle}$$

Os sinais são gerados por um Raspberry Pi através dos pinos GPIO. O mapa dos pinos encontra-se abaixo.

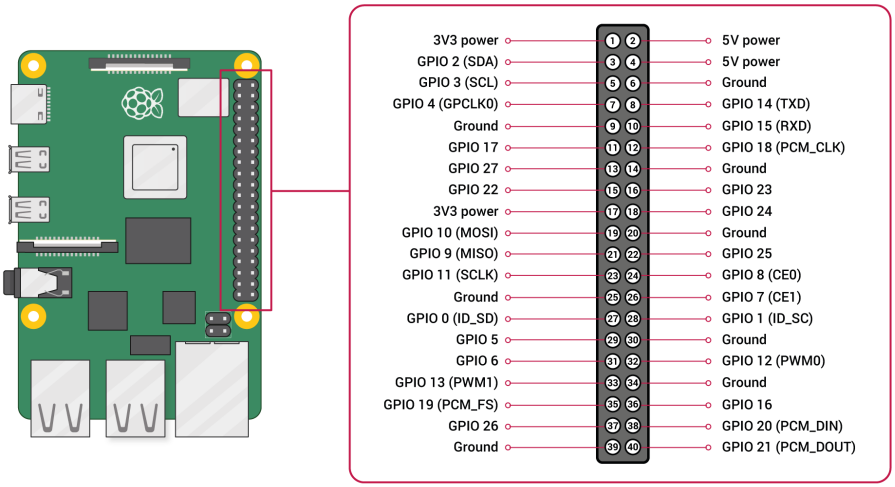


Figura 2: Mapa GPIO

3.2 Software

O sinal de controle PWM é gerado pelo Raspberry Pi4 usando a biblioteca **PIGPIO** `https://abyz.me.uk/rpi/pigpio/`.
Antes de usar a biblioteca, é necessário iniciar o daemon com o seguinte comando:
`sudo pigpiod`
Aqui está um exemplo de uso.

Listing 1: Output

```
1 # Import the pigpio library
2 import pigpio
3 import time
```

```
4
5  # Connect to the pigpio deamon
6  # First you have to execute the following command:
7  # sudo pigpiod
8
9  pi  = pigpio.pi()
10 if not pi.connected:
11   print("Could not connect to the pigpio deamon")
12   exit()
13
14 # Select a pin
15 servo_pin = 12
16
17 # Move to 0 deg
18 pi.set_servo_pulsewidth(servo_pin, 1500)
19 # Give some time to finish the movement
20 time.sleep(1)
21
22 # Move to 90 deg
23 pi.set_servo_pulsewidth(servo_pin, 2400)
24 time.sleep(1)
25
26 # Turn off the servo
27 pi.set_servo_pulsewidth(servo_pin, 0)
28 pi.stop()
```

ROS2

- Criar um nó `driver`
- Implementar a função `moveToInitialPose()`
- Implementar serviço `executeTrajectory` cuja requesta contém um único campo do tipo `JointTrajectory`