Apêndice D

Experimento 5: Como construir um controlador Avanço Atraso usando python

D.1 Introdução

Nesse experimento vamos descrever como o usuário pode construir um controlador incorporando a lei de controle vista no Capítulo 4. A lei de controle abordada no Capítulo 4, descrita na equação 4.13 e replicada como a equação D.1, tem como entrada o erro de posição e como saída (sinal de controle), uma referência de velocidade para as juntas do UR3.

$$u[k] = 0.958 \cdot u[k-1] + 0.30067 \cdot e[k] - 0.2108 \cdot e[k-1] \tag{D.1}$$

Para um melhor entendimento de como se dá o fluxo de dados entre o robô UR3, a Interface de Comunicação e o controlador proposto, foram esquematizadas duas figuras, a fim de esclarecer a fonte de cada dado que transita no sistema para a implementação de um controlador qualquer.

A Figura D.1 mostra, com uma separação de hardware (Linux PC e UR3), como é transmitido os dados entre a Interface de Comunicação e o robô UR3 e, também, de como é feita a comunicação entre o controlador proposto nesse experimento e a Interface de Comunicação. Já a Figura D.2 mostra, em malha fechada, a posição do controlador e do UR3 e quais suas entradas e saídas. Além disso, A Figura D.2 será usada para fazer a relação entre os sinais apresentados na própria Figura D.2 e as variáveis mostradas nos códigos das seções D.3.1 e D.1 (Fique atento a essa observação, por favor).

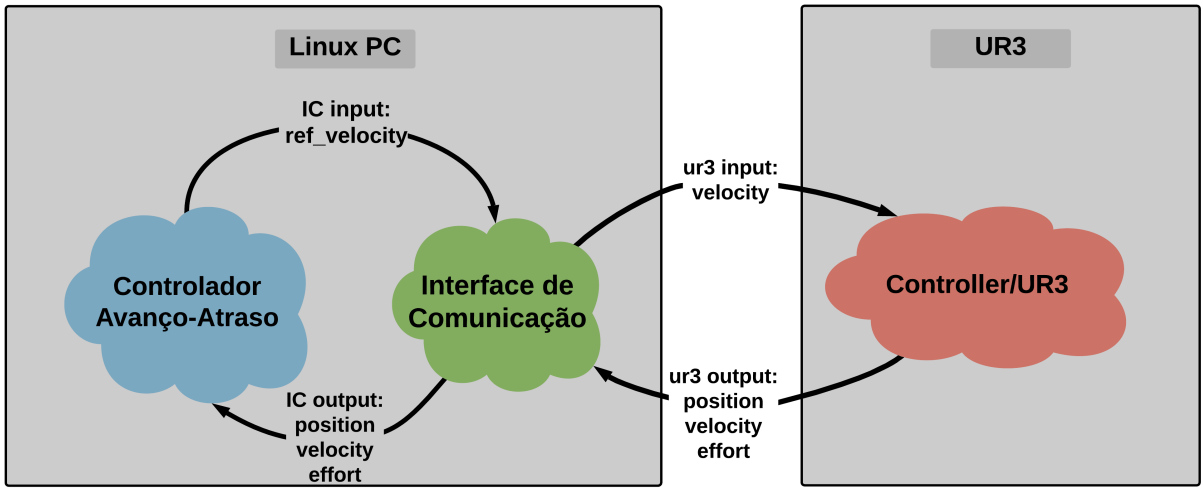

Figura D.1: Fluxo de dados entre o controlador a Interface de Comunicação e o UR3

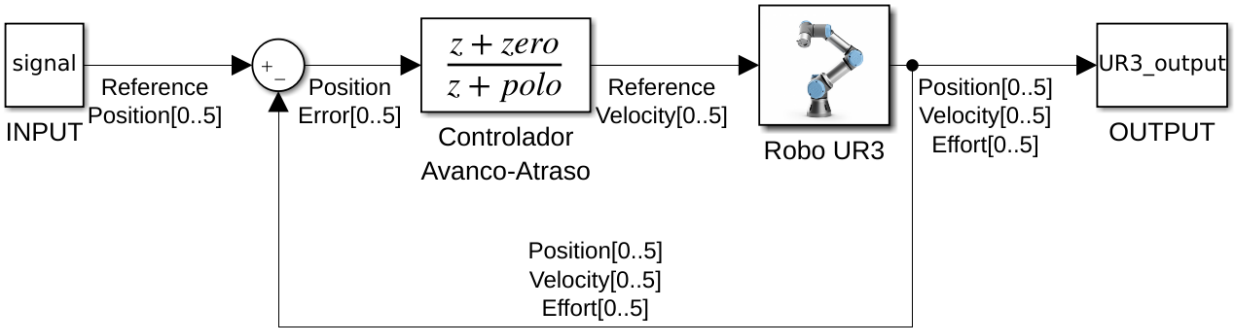

Figura D.2: Esquemático da malha de controle

D.2 Pacote ur3_controls

Pensando na organização dos códigos do UR3 dentro do **workspace catkin_ur3_ws** para o sistema de organização de arquivos ROS, demostrado na seção 2.2.2.1, foi criado um pacote chamado **ur3_controls**. Esse pacote tem como objetivo o armazenamento de todos os códigos onde serão escritos os controladores para o UR3.

A Figura D.3 mostra o pacote **ur3_controls** dentro do **workspace catkin_ur3_ws**.

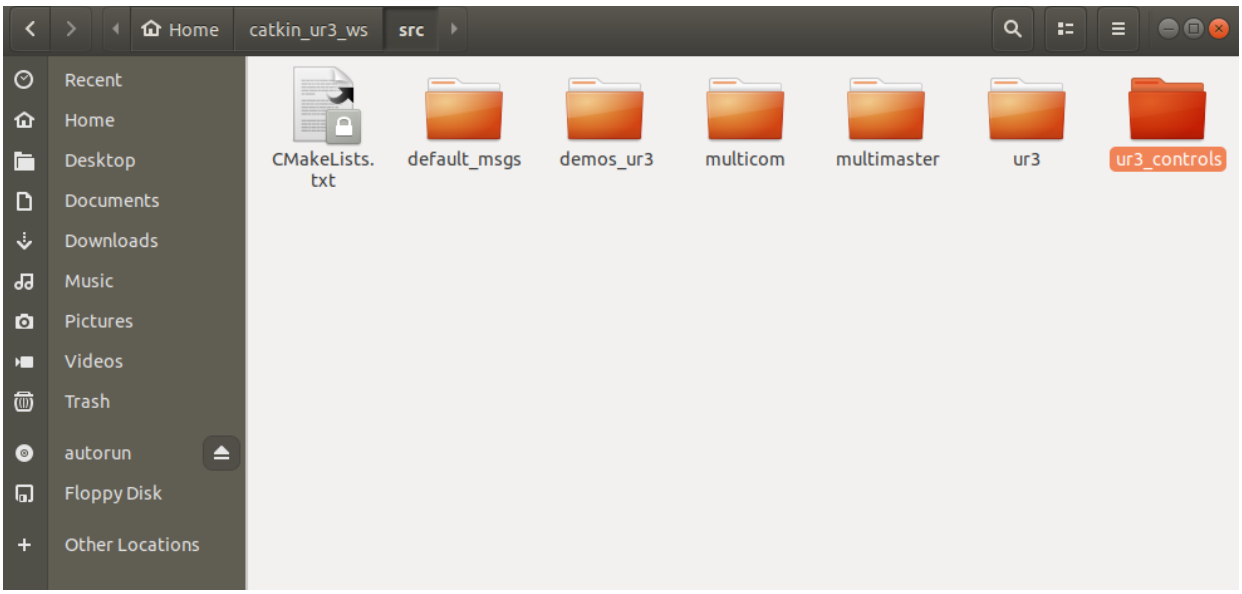

Figura D.3: Pasta selecionada mostrando o pacote ur3_controls

A Figura D.4 mostra a pasta **script** dentro do pacote **ur3_controls**. Os controladores serão armazenados dentro dessa pasta.

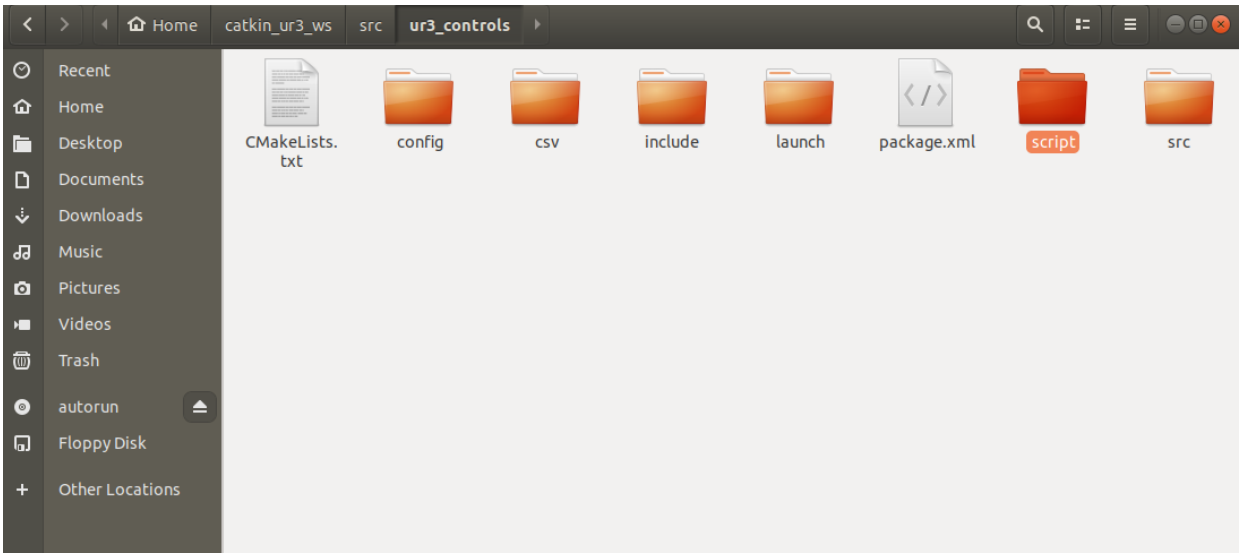

Figura D.4: Pasta script selecionada

A Figura D.5 mostra a pasta **config** dentro do pacote **ur3_controls**. Nela existe um arquivo muito importante para o entendimento desse experimento chamado **define_control.yaml**.

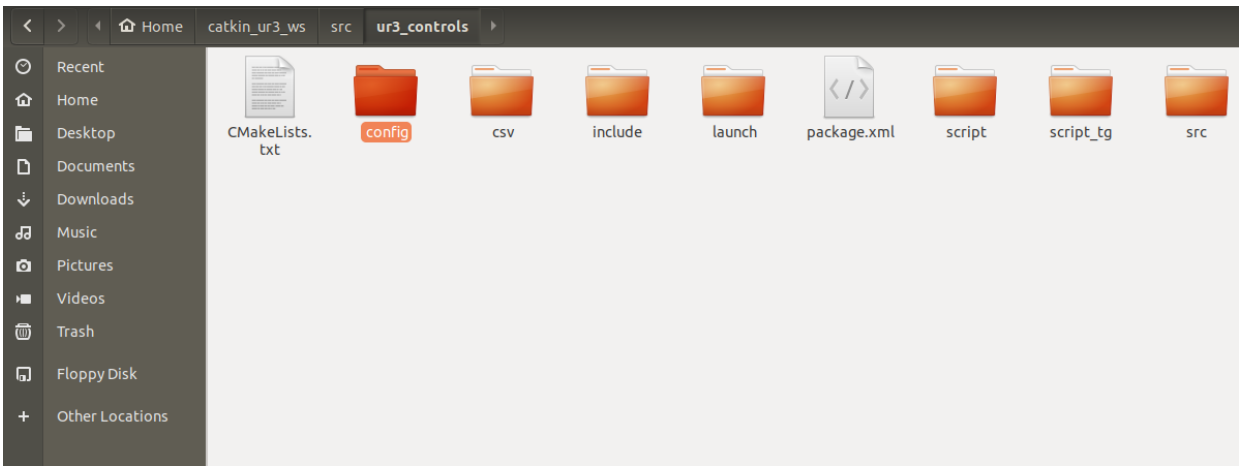

Figura D.5: Pasta config selecionada

A Figura D.6 mostra dentro da pasta **script**. Dentro dela há um arquivo que não pode ser alterado (o arquivo **main.py**). O arquivo chamado de **controle_avaco_atraso.py** será onde iremos implementar o controlador descrito na equação 4.13.

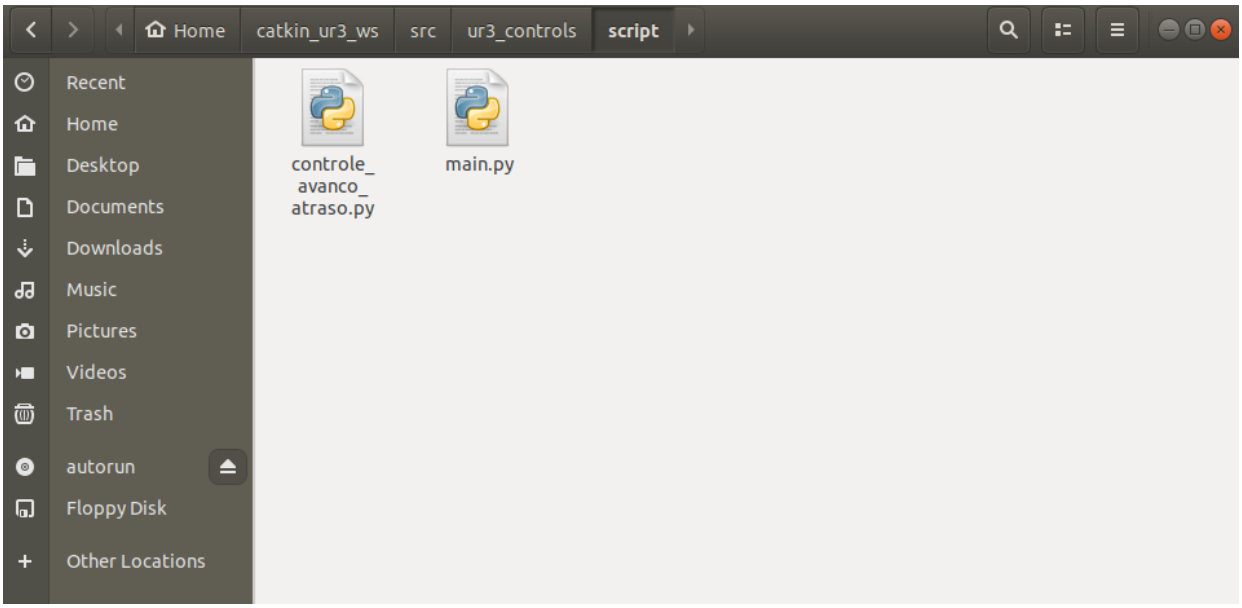

Figura D.6: Dentro Pasta script

A Figura D.7 mostra dentro da pasta **config**. Dentro dela existe um arquivo chamado de **define_control.yaml** que deve ser editado conforme a necessidade do experimento que será executado.

Entraremos em mais detalhes a respeito do arquivo **define_control.yaml** na seção D.4.

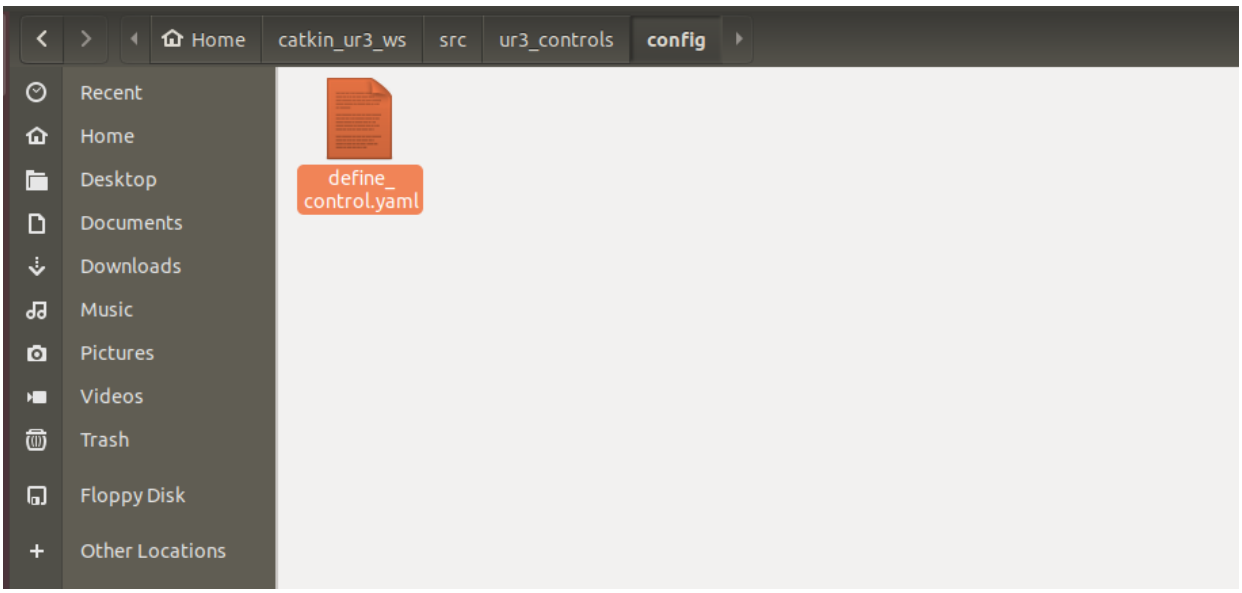

Figura D.7: Dentro Pasta script

D.3 Descrição do código Python controle_avaco_atraso.py usado para construir o controlador da equação D.1

Nessa seção, iremos abordar o passo-a-passo de como construir um controlador avanço-atraso para o UR3, nos moldes esquemáticos da Figura D.1, visando generalizar a construção para qualquer outro controlador buscando a máxima portabilidade dos controladores no sistema da Figura D.2. Para fazer a descrição, vamos nos basear nos códigos das sub-seções D.3.1, D.3.2 e D.4.

D.3.1 Inicio do código e a classe

Vamos descrever a parte do código correspondente a inicialização usando itens para facilitar o entendimento de cada linha escrita no código.

- **linha 1:** é descrito obrigatoriamente o interpretador python com **#!/usr/bin/env python**.
- **linha 2 a 5:** comentário a respeito do que se trata o código.
- **linha 6:** definição da classe com o tipo **class** e com o rótulo **MyControl**. O rótolo pode ter o nome que usuário quiser.
- **linha 7:** definição da o método de inicialição da classe **MyControl** com o tipo **def** e com o rótulo **___init___**. Esse método sempre recebe receberá o argumento **self**, ou seja, **___init___(self)**
- **linha 8 a 13:** comentário a respeito das variáveis de inicialização para o funcionamento do controlador.

- **linha 14:** definição de uma variável chamada de **self.Refence_Position** que é um vetor de 6 (seis) posições. Essa variável terá a função de representar as posições de referência para cada junta do UR3 como exemplificado na Figura D.2.

- **linha 15:** definição de uma variável chamada de **self.Position** que é um vetor de 6 (seis) posições. Essa variável terá a função de representar as posições lidas para cada junta do UR3 como exemplificado na Figura D.2.

- **linha 17:** definição de uma variável chamada de **self.Refence_Velocity** que é um vetor de 6 (seis) posições. Essa variável terá a função de representar os sinais de controle para cada junta do UR3 como exemplificado na Figura D.2.

- **linha 18:** definição de uma variável chamada de **self.Refence_Velocity_Old** que é um vetor de 6 (seis) posições. Essa variável terá a função de representar os sinais de controle, atrasados de uma amostra, para cada junta do UR3 como exemplificado na Figura D.2.

- **linha 21:** definição de uma variável chamada de **self.Position_Erro** que é um vetor de 6 (seis) posições. Essa variável terá a função de representar os sinais de erro de posição entre a referência e o sinal lido para cada junta do UR3, como exemplificado na Figura D.2.

- **linha 22:** definição de uma variável chamada de **self.Position_Erro_Old** que é um vetor de 6 (seis) posições. Essa variável terá a função de representar os sinais de erro de posição, atrasados de uma amostra, entre a referência e o sinal lido para cada junta do UR3, como exemplificado na Figura D.2.

```
#!/usr/bin/env python

# CONTROLADOR DE POSICAO AVANCO-ATRASO
# Esse codigo eh para ser usado como template para
# a construcao de outros controladores
class MyControl:
    def __init__(self):
        # Defina os parametros do controlador e variaveis
        # globais neste metodo (__init__)
        # Para todas as juntas do ur3, da junta da base
        # ate a junta do efetuador terminal, defina o estado
        # inicial aqui

        self.Refence_Position = [0, 0, 0, 0, 0, 0] # entrada de posicao
        self.Position = [0, 0, 0, 0, 0, 0] # saida de posicao

        self.Refence_Velocity = [0, 0, 0, 0, 0, 0] # sinal de controle
        self.Refence_Velocity_Old = [0, 0, 0, 0, 0, 0] # sinal de controle
        # atrasado de uma amostra

        self.Position_Erro = [0, 0, 0, 0, 0, 0] # sinal de erro
        self.Position_Erro_Old = [0, 0, 0, 0, 0, 0] # sinal de erro
        # atrasado de uma amostra
```

D.3.2 Lei de Controle

Vamos descrever a parte do código correspondente a **lei de controle** usando itens para facilitar o entendimento de cada linha escrita no código.

- **linha 1:** definição de uma função com o tipo **def** e com o rótulo **control_law**. O rótulo deve ter, obrigatoriamente, o nome **control_law**. Os argumentos dessa função serão listados abaixo.
  - **self**: argumento padrão da função
  - **robot_state**: variável com os estados das juntas do UR3 com a mensagem do tipo **JointState**. Esse formato de mensagem possui as sub-mensagens de posição (position), velocidade (velocity) e torque (effort).
    Para mais detalhes, consulte a documentação ROS da mensagem em `http://docs.ros.org/en/api/sensor_msgs/html/msg/JointState.html`
  - **ref_pose_msg**: mensagem de referência de posição para todas as juntas do UR3 com 6 (seis) posições de dados do tipo **Float64MultiArray**.
- **linha 16:** definição de um loop interativo do tipo **for**, que repete 6 (seis) vezes, de forma a gerar o sinal de controle para as seis juntas do UR3 usando a lei de controle da equação D.1. Vamos comentar as linhas de código que pertencem a esse loop levando em consideração que a variável **idx** começa com 0 (zero) e vai até 5 (cinco), totalizando 6 (seis) interações.
  - **linha 18:** passando a idx-ésima referência de posição, que vem de **ref_pose_msg.data[idx]**, para a idx-ésima posição no vetor **self.Reference_Position[idx]**.
  - **linha 19:** passando a idx-ésima posição lida da idx-ésima junta do UR3, que vem de **robot_state.position[idx]**, para a idx-ésima posição no vetor **self.Position[idx]**.
  - **linha 22:** passando o erro de posição para a variável **self.Position_Erro[idx]**.
  - **linha 27 a 29:** nessas linhas acontece a execução da lei de controle. Vamos fazer a relação de cada variável presente nessas linhas com as variáveis presente na equação D.1 e com as variáveis do esquemático da Figura D.2.
    - **linha 27:** Nessa linha temos duas variáveis.
      Temos a **self.Reference_Velocity[idx]**, que representa $u[k]$ na equação D.1 e representa com **Reference_Velocity[0..5]** no esquemático da Figura D.2.
      Temos, logo depois, a **self.Reference_Velocity_Old[idx]**, que representa $u[k-1]$ na equação D.1 e também representa a variável **Reference_Velocity[0..5]** no esquemático da Figura D.2 atrasado de uma amostra.
    - **linha 28:** Nessa linha temos **self.Position_Erro[idx]**, que representa $e[k]$ na equação D.1 e também representa a variável **Position_Erro[0..5]** no esquemático da Figura D.2.
    - **linha 29:** Nessa linha temos **self.Position_Erro_Old[idx]**, que representa $e[k-1]$ na equação D.1 e também representa a variável **Position_Erro[0..5]** no esquemático da Figura D.2 atrasado de uma amostra.

- **linha 31:** Esta linha tem a função de passar o valor de **self.Reference_Velocity[idx]** para **self.Reference_Velocity_Old[idx]** para construir a variável $u[k-1]$ representada na equação D.1.
- **linha 32:** Esta linha tem a função de passar o valor de **self.Position_Erro[idx]** para **self.Position_Erro_Old[idx]** para construir a variável $e[k-1]$ representada na equação D.1.

- **linha 35:** Nesta linha é feito a execução da instrução **return** que faz o retorno das variáveis de referência de velocidade usando **return self.Reference_Velocity**.

```
def control_law(self, robot_state, ref_pose_msg):
    ##################################################################
    # argumendo de control law:
    # self: indica que essa funcao eh acessada apenas pela classe
    # MyControl ou o nome que o usuario queira dar a classe
    # robot_state: mensagem ROS dos tipo JointState
    # mais detalhes da mensagem pode ser encontrado em
    # http://docs.ros.org/en/api/sensor_msgs/html/msg/JointState.html
    # ref_pose_msg: mensagem de referencia de posicao para todas as
    # juntas do robo do tipo Float64MultiArray() para
    # [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3]
    # mais detalhes da mensagem pode ser encontrado em
    # http://docs.ros.org/en/api/std_msgs/html/msg/Float64MultiArray.html
    ##################################################################
    # for que gere todas as variaveis do controlador
    for idx in range(6):
        # idx eh um interador que vai de 0 a 5:
        self.Refence_Position[idx] = ref_pose_msg.data[idx]
        self.Position[idx] = robot_state.position[idx]
        # self.ek[idx]: sinal de erro de posicao para
        # cada junta(entra - saida)
        self.Position_Erro[idx] = (self.Refence_Position[idx]
            - self.Position[idx][idx])
        ##################################################################
        # Lei de controle para o controlador proposto
        # para a junta 3 (Elbow Joint)
        if idx == 2:
            self.Reference_Velocity[2] = (0.958*self.Reference_Velocity_Old[2]
                + 0.30067*self.Position_Erro[2]
                - 0.2108*self.Position_Erro_Old[2])
        ##################################################################
        self.Reference_Velocity_Old[idx] = self.Reference_Velocity[idx]
        self.Position_Erro_Old[idx] = self.Position_Erro[idx]
    ##################################################################
    # sinal de controle parada junta do robo(sinal de velocidade)
    return self.Reference_Velocity
```

D.4 Estrutura do arquivo de configuração

Com o controlador implementado na seção D.3.2, agora vamos descrever, com um arquivo de configuração que é nomeado de **define_control.yaml**, como é feita o acoplamento do controlador na aplicação que gere o controlador descrita nos no Capitulo Resultado. Esse aquivo tem a função de fornecer o caminho onde **controle_avaco_atraso.py** se encontra, como é o nome da classe presente no arquivos e outros detalhes que serão melhor explorados na descrição.

vamos seguir a metodologia de descrição feita para o controlador **controle_avaco_atraso.py**, onde foi usado itens e subitens para mencionar cada linha do arquivo e qual sua função.

Primeiramente, vamos ignorar as linha em branco e as linha com comentários, pois os devidos esclarecimentos serão dados nos itens a seguir.

- **linha 3:** definido a categoria **control** e nela será define as seguintes variáveis:
  - **linha 4:** a variável **control_name** será onde o usuário deve nomear os tópicos de iniciação do experimento, de parada do experimento e envio de referência de posição.

    O nome escolhido pode ficar a critério do usuário. No exemplo do arquivo de configuração, mostrado em D.4.1, foi colocado o nome de **control_aa**.
  - **linha 5:** a variável **file_name** receberá o nome do arquivo onde foi implementado o controlador sem a extensão do arquivo **.py**.

    No exemplo do arquivo de configuração mostrado em D.4.1 foi atribuído a variável **file_name** o nome de **controle_avanco_atraso**.
  - **linha 6:** a variável **class_name** receberá o nome da classe que onde foi implementado o controlador.

    No exemplo do arquivo de configuração mostrado em D.4.1 foi atribuído a variável **class_name** o nome de **MyControl**.
- **linha 10:** definido a categoria **csv_name**, serão definidos nela as seguintes variáveis:
  - **linha 11:** a variável **sine** que não pode ser muda. Essa variável é usada para fazer demostrações onde a entrada é uma onda senoidal gerada previamente.
  - **linha 12:** a variável **square** que não pode ser muda. Essa variável é usada para fazer demostrações onde a entrada é uma onda quadrada gerada previamente.
  - **linha 13:** a variável **your_wave** é aquela que receberá o nome do arquivo **csv** gerado previamente pelo usuário. Caso tenha dúvida de como gerar um onda para o UR3, sugiro que estude o conteúdo do Apêndice B.
- **linha 18:** definido a categoria **reference_type**, será definido nela as seguintes variáveis:
  - **linha 19: online** é uma variável booleana que, se for atribuído o valor **False** o controlador usará como entrado a onda que é definida na variável **type** (linha 20). Caso

a variável **online** seja definida como **True**, controlador passará a receber referência de posição gerada de forma online por outra aplicação [1].

- **linha 20:** a variável **type** uma das variáveis pertencente a categoria **reference_type** (linha 18), podendo ser **sine, square** ou **your_wave**.

- **linha 23:** definido a categoria **activated_joints**, nela será definida variáveis booleanas que tem a capacidade de ativar e desativar as juntas do UR3. Caso queira que o controlador use uma determinada junta, defina a variável como **True**. Caso queira que o controlador não use uma determinada junta, defina a variável como **False**.

  - **linha 24:** variáveis booleana **Base**. poder definida como **True** ou **False**.
  - **linha 25:** variáveis booleana **Shoulder**. Poder ser definida como **True** ou **False**.
  - **linha 26:** variáveis booleana **Elbow**. Pode ser definida como **True** ou **False**.
  - **linha 26:** variáveis booleana **Wrist1**. Pode ser definida como **True** ou **False**.
  - **linha 26:** variáveis booleana **Wrist2**. Pode ser definida como **True** ou **False**.
  - **linha 26:** variáveis booleana **Wrist3**. Pode ser definida como **True** ou **False**.

[1] Outra aplicação pode ser entendido como outro robô ou algum sensor externo ao UR3

D.4.1 Arquivo de configuração define_control.yaml

```
# define the name of file and class
# where you wrote your control
control:
  control_name: control_aa #(nick name for control (whatever you want))
  file_name: controle_avanco_atraso #(python file name without ".py")
  class_name: MyControl #(class name that you wrote in
  #controle_avanco_atraso.py)

# Choose the csv fine for offline experiment
csv_name:
  sine: 'ref_sine.csv' # Default to demo (type sine)
  square: 'ref_square.csv' # Default to demo (type square)
  your_wave: None # Custom to your experiment (type your_wave)

#Choose if you want online or offline experiment (if offline
# type, choose the wave type)
# and type of reference
reference_type:
  online: False
  type: 'sine' # if online is define as True type is ignored

#choose which joints you want activated
activated_joints:
  Base: False
  Shoulder: False
  Elbow: False
  Wrist1: False
  Wrist2: False
  Wrist3: True
```

D.4.2 Código completo do controlador Avanço-Atraso

```
#!/usr/bin/env python

# CONTROLADOR DE POSICAO AVANCO-ATRASO
# Esse codigo eh para ser usado como template para
# a construcao de outros controladores
class MyControl:
    def __init__(self):
        # Defina os parametros do controlador e variaveis
        # globais neste metodo (__init__)
        ##################################################################
        # Para todas as juntas do ur3, da junta da base
        # ate a junta do efetuador terminal, defina o estado inicial
        # aqui
        self.Refence_Position = [0, 0, 0, 0, 0, 0] # entrada de posicao qr[k]
        self.Position = [0, 0, 0, 0, 0, 0] # saida de posicao qo[k]

        self.Reference_Velocity = [0, 0, 0, 0, 0, 0] # sinal de controle u[k]
        self.Reference_Velocity_Old = [0, 0, 0, 0, 0, 0] # sinal de controle u[k-

        self.Position_Erro = [0, 0, 0, 0, 0, 0] # sinal de erro e[k]
        self.Position_Erro_Old = [0, 0, 0, 0, 0, 0] # sinal de erro e[k-1]

    def control_law(self, robot_state, ref_pose_msg):
        ##################################################################
        # argumendo de control law:
        # self: indica que essa funcao eh acessada apenas pela classe
        # MyControl ou o nome que o usuario queira dar a classe
        # robot_state: mensagem ROS dos tipo JointState
        # mais detalhes da mensagem pode ser encontrado em
        # http://docs.ros.org/en/api/sensor_msgs/html/msg/JointState.html
        # ref_pose_msg: mensagem de referencia de posicao para todas as
        # juntas do robo do tipo Float64MultiArray() para
        # [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3]
        # mais detalhes da mensagem pode ser encontrado em
        # http://docs.ros.org/en/api/std_msgs/html/msg/Float64MultiArray.html
        ##################################################################
        # for que gere todas as variaveis do controlador
        for idx in range(6):
            # idx eh um interador que vai de 0 a 5:
            self.Refence_Position[idx] = ref_pose_msg.data[idx]
```

```
41              self.Position[idx] = robot_state.position[idx]
42              # self.ek[idx]: sinal de erro de posicao para
43              # cada junta(entra - saida)
44              self.Position_Erro[idx] = (self.Refence_Position[idx]
45               - self.Position[idx])
46              ##############################################################
47              # Lei de controle para o controlador proposto
48              # para a junta 3 (Elbow Joint)
49              if idx == 2:
50                  self.Reference_Velocity[2] = (0.958*self.Reference_Velocity_Old[2
51                      + 0.30067*self.Position_Erro[2]
52                      - 0.2108*self.Position_Erro_Old[2])
53              ##############################################################
54              self.Reference_Velocity_Old[idx] = self.Reference_Velocity[idx]
55              self.Position_Erro_Old[idx] = self.Position_Erro[idx]
56          ##################################################################
57          # sinal de controle parada junta do robo(sinal de velocidade)
58          return self.Reference_Velocity
59          ##################################################################
```

D.5 Como rodar o controlador proposto nesse experimento

Como esse experimento é apenas para uma demonstração de funcionamento, por questão de segurança, será habilitado apenas a junta chamada de **Wrist3**. Depois, quando o usuário estiver familiarizado com o UR3, poderá personalizar o arquivo de configuração descrito em D.4.

Assumindo que o leitor tenha feito o processo de setup do ambiente de experimento, como é demonstrado no Apêndice A da seção A.1.1 até a seção A.1.2. Feito os procedimentos da seção A, abra uma seção do terminal e vá para o **workspce** do UR3, que nada mais é a pasta **catkin_ur3_ws**.

Use o comando abaixo para ir para **workspce**:

cd ~/catkin_ur3_ws

Use o comando abaixo para fazer o diretório do **workspce** um diretório ROS.

source devel/setup.zsh.

D.5.1 Experimento offline

No experimento **offline** a variável **online** (linha 19 no arquivo de configuração D.4.1) precisa ser definida como **False**. Então, vá ao diretório (catkin_ur3_ws/src/ur3_controls/config), onde

se encontra o arquivo de configuração **define_control.yaml** e certifique-se de que a variável **online** está definida como **False**.

Para rodar o controlador descrito em D.3.2, rode o comando abaixo.

roslaunch ur3_controls main.launch.

Nesse momento, o controlador está esperando o usuário enviar o comando para a inicialização do experimento.

Abra uma seção do terminator e vá para o **workspce** do UR3, que nada mais é a pasta **catkin_ur3_ws**.

Use o comando abaixo para ir para **workspce**:

cd ∼/catkin_ur3_ws

Use o comando abaixo para fazer o diretório do **workspce** um diretório ROS.

source devel/setup.zsh.

rosservice call /ur3_controls/start_control_aa "data: true".

A junta **wrist3** começará a se mover

Caso queria parar o experimento, envie o comando abaixo ou espere o experimento chegar ao fim.

rosservice call /ur3_controls/start_control_aa "data: false".

Para desligar o controlador, vá na abata do terminal onde foi rodado controlador e pressione Ctrl+C.

D.5.2 Experimento online

No experimento **online** a variável **online** (linha 19 no arquivo de configuração D.4.1) precisa ser definida como **True**. Então, vá ao diretório (catkin_ur3_ws/src/ur3_controls/config), em que se encontra o arquivo de configuração **define_control.yaml** e certifique-se de que a variável **online** está definida como **True**.

Para rodar o controlador descrito em D.3.2, rode o comando abaixo.

roslaunch ur3_controls main.launch.

Nesse momento, o controlador está esperando o usuário enviar o comando para de inicialização do experimento.

Abra uma seção do terminator e vá para o **workspce** do UR3, que nada mais é a pasta **catkin_ur3_ws**.

Use o comando abaixo para ir para **workspce**:

cd ~/catkin_ur3_ws

Use o comando abaixo para fazer o diretório do **workspce** um diretório ROS.

source devel/setup.zsh.

Inicialize o controlador com o comando abaixo.

rosservice call /ur3_controls/start_control_aa "data: true".

Para enviar um referência de posição para a junta **Wrist3**, abra uma seção do terminator e vá para o **workspce** do UR3, que nada mais é que a pasta **catkin_ur3_ws**.

Use o comando abaixo para ir para **workspce**:

cd ~/catkin_ur3_ws

Use o comando abaixo para fazer o diretório do **workspce** um diretório ROS.

source devel/setup.zsh.

O comando abaixo foi projetado para ser de fácil entendimento para o usuário, ou seja, o nome de cada junta foi inserido no comando para facilitar a visualização da junta e seu respectivo valor de reverência de posição.

Podemos observar no comando abaixo, que a junta de interesse **(Wrist3)** recebe o valor de 0.2 rad.

O comando abaixo é apenas para visualização, pois se o usuário tentar copiar do pdf a cópia será mal sucedida. Para contornar esse problema, o comando foi adicionado ao repositório do controlador no github do LARA e pode ser acessado em `https://github.com/lara-unb/catkin_ur3_ws/blob/main/src/ur3_controls/cmd/comando_ref_pose.yaml`.

Com o comando presente no github do LARA o usuário pode copiar e colar na aba do terminator que foi aberta para esse procedimento.

Cole, pressione ENTER e a junta **Wrist3** começará a se movimentar para a posição de referênciua.

```
rostopic pub /ur3_controls/control_aa/target_position default_msgs/JointPosition
"header:
  seq: 0
  stamp: {secs: 0, nsecs: 0}
  frame_id: "
Base: 0.2
Shoulder: -1.570796
Elbow: 0.0
Wrist1: -1.570796
Wrist2: 0.0
Wrist3: 0.0"
```

Caso queria parar o experimento, envie o comando abaixo.

rosservice call /ur3_controls/start_control_aa "data: false".

Caso queira usar a interface gráfica do **ur3_contrls** (Figura D.8) para mover a juntas do UR3, abra outro seção no terminator e rode o comando a seguir.

rosrun rqt_gui rqt_gui -s reconfigure.

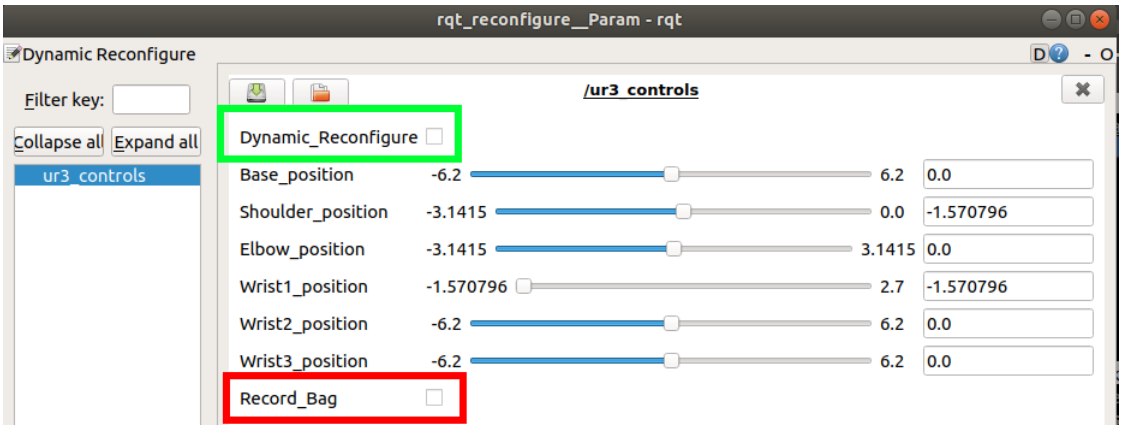

Figura D.8: Interface ur3_contros

Na Figura D.8, mostra, em destaque por um retângulo verde, o campo **Dynamic_Reconfigure** com opção para habilitar a interface para mover as juntas do UR3. Também mostra, em destaque por um retângulo vermelho, o campo **Record_Bag** para gravar os dados de entrada e saída do UR3 (Veja o Apêndice C).

Para desligar o controlador, vá na abata do terminal onde foi rodado controlador e pressione Ctrl+C.

FIM!