Documentação de Programação de Robôs

Versão traduzida

Rev.2
PTM268S9136M

Índice de Conteúdo

Noções básicas de Python

Padrões básicos de Python

Esta seção explica alguns conceitos básicos de programação em Python que podem ser usados no editor de aplicações. Trata-se apenas de um resumo dos padrões mais importantes do Python (3.13); naturalmente, existem muitas outras funções integradas disponíveis. Tenha em mente que uma aplicação robótica não é apenas uma aplicação Python, portanto algumas funcionalidades do Python são restritas ou não podem ser utilizadas. Para usuários avançados de Python ou tutoriais de programação, recomendamos a Documentação oficial do Python 3.13.

Comentários

As linhas de código a seguir representam comentários e não são executadas em tempo de execução.

```
# this is a commentary and will not be considered as executable code
"""
This is a multiline commentary
that will also not be executed.
"""
```

Definições de variáveis

O Python suporta muitos tipos diferentes de variáveis e não diferencia definição de variável e declaração de variável.

```
my_boolean = True            # defining "my_boolean" as a Boolean with value True
my_integer = 3               # defining "my_integer" as an integer number with value 3
my_float = 3.1416            # defining "my_float" as a floating-point number 3.1416
my_string = "Hello World"    # defining "my_string" as a string with value "Hello
World"
```

Conversões de variáveis

As variáveis podem ser facilmente convertidas de um tipo para outro, desde que os valores sejam compatíveis.

```
x = bool(x)    # converts x to a boolean
x = int(x)     # converts x to an integer number
x = float(x)   # converts x to a floating-point number
x = str(x)     # converts x to a string
```

Estruturas de dados

O Python suporta quatro tipos diferentes de estruturas de dados:

- Uma lista contém uma sequência ordenada de elementos.

```
my_list = list()            # create an empty list
my_list = [1, 2, 3, 4, 5]   # create a list containing some numbers
my_list.append(6)           # add element to existing list
my_list[0] = 7              # sets the first list element to be 7
```

- Uma tupla contém uma sequência ordenada, porém imutável (inalterável), de elementos.

```
my_tuple = tuple()           # create an empty tuple
my_tuple = (1, 2, 3, 4, 5)   # create a tuple containing some numbers
```

- Um conjunto (set) contém uma coleção não ordenada de elementos sem duplicatas.

```
my_set = set()                          # create an empty set
my_set = {1, 2, 3, 4, 5}                # create a set containing some numbers
my_set.intersection({1, 3, 5, 7, 9})    # intersection of two sets
my_set.union({1, 3, 5, 7, 9})           # union of two sets
```

- Um dicionário contém uma combinação não ordenada de chaves e valores.

```
my_dictionary = dict()              # create an empty dictionary
my_dictionary = {"a": 1, "b": 2}    # create a dictionary containing key-value pairs
my_dictionary["a"] = 3              # sets the element "a" to be 3
```

Bibliotecas e importações

O Python fornece um conjunto de módulos integrados (a biblioteca padrão). Os seguintes módulos são suportados:

- Operações numéricas e aleatoriedade – útil para cálculos, funções matemáticas e geração de valores aleatórios

```
import math, random, statistics
```

- Gestão de tempo e data – trabalho com timestamps, atrasos e valores de data e hora

```
import time, datetime
```

- Armazenamento de dados e formatos de ficheiros – leitura e escrita de dados estruturados como JSON ou CSV

```
import json, csv
```

- Processamento de texto – manipulação de strings e correspondência de padrões com expressões regulares

```
import re, string
```

- Ferramentas avançadas de iteração e funcionais – loops eficientes, combinações e funções de ordem superior

```
import itertools, functools
```

Além da biblioteca padrão, estão disponíveis as seguintes bibliotecas de terceiros:

- Computação numérica – processamento eficiente de arrays e dados numéricos

```
import numpy
```

- Matemática de orientação 3D – representação de quaternions e transformações (relacionado com robótica)

```
import pyquaternion
```

- Comunicação HTTP – envio de pedidos para serviços web e APIs REST

```
import requests
```

- Configuração e serialização de dados – trabalho com dados em formato YAML

```
import yaml
```

Os seguintes módulos padrão do Python não estão disponíveis em aplicações de robôs:

- Acesso ao sistema operativo (ex.: `os`, `shutil`, `subprocess`, `sys`)
- Paralelização personalizada (ex.: `multiprocessing`, `threading`)
- Inspeção e depuração personalizadas (ex.: `inspect`, `traceback`)

Além disso, a instalação de bibliotecas de terceiros personalizadas não é suportada no ambiente Python do robô.

Funções básicas de Python

Argumentos obrigatórios e opcionais de funções

Vamos considerar que uma função foi definida com os seguintes argumentos:

```
def function(arg1, arg2, arg3="x", arg4="y"):
    ...
```

Os argumentos da função arg1 e arg2 são argumentos padrão que devem ser fornecidos ao chamar a função. Os argumentos arg3 e arg4 possuem valores padrão e, portanto, não precisam necessariamente ser fornecidos. Todas as chamadas de função a seguir são válidas:

```
function("a", "b")                                # -> function("a", "b", "x", "y")
function("a", "b", "c")                           # -> function("a", "b", "c", "y")
function("a", "b", "c", "d")                      # -> function("a", "b", "c", "d")
function("a", "b", arg4="d")                      # -> function("a", "b", "x", "d")
function(arg1="a", arg2="b", arg3="c", arg4="d")  # -> function("a", "b", "c", "d")
function(arg3="c", arg2="b", arg1="a")            # -> function("a", "b", "c", "y")
```

Argumentos de função desempacotados

Algumas funções mencionadas na documentação do código utilizam argumentos desempacotados. Esses argumentos são úteis quando o número de argumentos de uma função pode variar. Vamos considerar os seguintes argumentos de função:

```
function1(*arguments)    # function with unpacked list argument
function2(**arguments)   # function with unpacked dictionary argument
```

Essas funções podem então ser utilizadas da seguinte forma:

```
function1(value1, value2, ...)             # function with unpacked list argument
function2(arg1=value1, arg2=value2, ...)   # function with unpacked dict argument
```

Operadores básicos de Python

Operadores de cálculo

```
x = 3 + 5    # addition:        the value of x is now 8
x = 7 - 2    # subtraction:     the value of x is now 5
x = 4 * 8    # multiplication:  the value of x is now 32
x = 54 / 6   # division:        the value of x is now 9
x = 16 % 7   # modulo:          the value of x is now 2
x = 2 ** 3   # exponential:     the value of x is now 8
```

Operadores de comparação

Aqui, x e y podem ser de qualquer tipo.

```
x == y   # returns true if x is equal to y
x != y   # returns true if x is not equal to y
```

Aqui, x e y são números.

```
x > y    # returns true if x is greater than y
x < y    # returns true if x is smaller than y
x >= y   # returns true if x is greater than or equal to y
x <= y   # returns true if x is smaller than or equal to y
```

Operadores lógicos

Aqui, x e y são valores booleanos.

```
x and y   # logical AND returns True if x and y are both True
x or y    # logical OR returns True if either x or y is True
not x     # logical NOT returns True if x is False
```

Operadores bit a bit

Aqui, x e y são números inteiros.

```
x & y   # bitwise AND returns an integer resulting from bitwise logical AND of x and y
x | y   # bitwise OR returns an integer resulting from bitwise logical OR of x and y
```

Programação condicional

Condição If–Else

As condições `if`, `elif` (else if) e `else` podem ser combinadas de acordo com as seguintes regras:

- O bloco `if` é executado apenas se a sua condição retornar True.
- Um bloco `elif` é opcional e é executado apenas se todas as condições anteriores `if` ou `elif` retornarem False e a sua própria condição retornar True.
- Um bloco `else` é opcional e é executado apenas se todas as condições anteriores `if` ou `elif` retornarem False.

O código geral if-elif-else tem a seguinte aparência:

```
if condition_1:
    # the following code is executed if 'condition_1' is True
    ...
elif condition_2:
    # the following code is executed if 'condition_1' is False
    # and 'condition_2' is True
    ...
elif condition_3:
    # the following code is executed if 'condition_1' and 'condition_2' are False
    # and 'condition_3' is True
    ...
else:
    # the following code is executed if all previous conditions leading up
    # to this 'else' statement are False
    ...
```

Exemplos:

Verificando se a variável `my_integer` é igual a 5, 10 ou a nenhum desses números.

```
if my_integer == 5:
    print("your integer is 5.")
    print("have a nice day.")
elif my_integer == 10:
    print("your integer is 10.")
    print("have a nice day.")
else:
    print("your integer is neither 5 nor 10.")
    print("have a nice day anyway.")
```

Laço While

Um laço while é executado e repetido de acordo com as seguintes regras:

- O bloco `while` é executado repetidamente enquanto sua condição retornar True.
- A palavra-chave `continue` é opcional e pode ser usada para pular para a próxima iteração do laço, ignorando o código restante dentro do while.
- A palavra-chave `break` é opcional e pode ser usada para sair deste laço.
- Um bloco `else` é opcional (e raramente usado) e é executado se a condição do laço retornar False. Em outras palavras, este bloco não é executado se o while for encerrado com um break.

```
while condition:
    # the following code is executed repeatedly as long as 'condition' is True
    ...
    if condition_1:
        # the 'continue' keyword skips the remaining code of this loop
        # and jump to the next iteration
        continue
    if condition_2:
        # the 'break' keyword breaks out of this loop, no matter the loop condition
        break
else:
    # the following code is executed only if the while loop exits
    # by setting 'condition' to False
    ...
```

Exemplos:

Imprimindo todos os números de 1 a 10.

```
index = 1
while index <= 10:
    print(index)
    index += 1
```

Laço Infinito: Laços infinitos são úteis quando você quer que o robô continue repetindo uma tarefa até que você pare manualmente ou saia do laço usando uma condição ou função específica. Se o laço tende a executar muito rápido, recomenda-se adicionar comandos de espera para dar algum tempo de respiro ao processador.

```
import time
while True:
    ...
    time.sleep(0.1)
```

Laço For

Um laço for é executado e repetido de acordo com as seguintes regras:

- O bloco `for` é executado repetidamente, uma vez para cada item em uma sequência iterável (por exemplo, lista, conjunto, dicionário ou string).
- A palavra-chave `continue` é opcional e pode ser usada para pular para a próxima iteração do laço, ignorando o código restante dentro do for.
- A palavra-chave `break` é opcional e pode ser usada para sair deste laço.
- Um bloco `else` é opcional (e raramente usado) e é executado após o último item da sequência do laço for ter sido processado. Em outras palavras, este bloco não é executado se o for for encerrado com um break.

```
for item in sequence:
    # the following code is executed once for each 'item' in 'sequence'
    ...
    if condition_1:
        # the 'continue' keyword skips the remaining code of this loop
        # and jump to the next iteration
        continue
    if condition_2:
        # the 'break' keyword breaks out of this loop, no matter the loop condition
        break
else:
    # the following code is executed only if the for loop exits
    # by finding no more 'item' in 'sequence'.
    ...
```

Exemplos:

Imprimindo todos os números de 1 a 5.

```
for index in range(5):
    print(index+1)
```

Imprimindo minhas frutas favoritas.

```
for fruit in ["apple", "banana", "orange"]:
    print(f"one of my favorite fruits is {fruit}")
```

Contando as letras em `python`.

```
index = 1
for letter in "python":
    print(f"letter number {index} in 'python' is {letter}")
    index += 1
```

Funções principais

class Core

Nota

Esta é uma coleção de todas as funções principais. Estas funções estão diretamente disponíveis e podem ser chamadas no código das aplicações.

move_joints(*joints*, *joint_position*, *joint_velocity=None*, *joint_acceleration=None*, *relative=False*, *block=True*)

Mover uma ou mais articulações do robô para um conjunto específico de ângulos articulares. Este movimento sincroniza todas as articulações para que alcancem a posição alvo ao mesmo tempo.

Parâmetros:

- **joints** (*int, str, list*) –
  Os IDs dos atuadores a serem movidos. Especifique da seguinte forma:
  - o ID de uma única articulação (ex. *6*)
  - uma lista de IDs para várias articulações (ex. *[1, 4, 6]*)
  - a constante *ALL_KIN_JOINTS* para mover todas as seis articulações
- **joint_position** (*float, list of float*) –
  As posições angulares das articulações a mover em [°]. Especifique da seguinte forma:
  - um único número para mover todas as articulações especificadas para a mesma posição (ex. *-30*)
  - uma lista de números para mover cada articulação especificada para uma posição diferente (ex. *[90, -45, 30]*)
- **joint_velocity** (*float or None*) – A velocidade máxima em percentagem do limite de velocidade das articulações. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *45* %). Se nenhum valor for especificado, utiliza-se a velocidade global das articulações.
- **joint_acceleration** (*float or None*) – A aceleração máxima em percentagem do limite de aceleração das articulações em movimento. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *60* %). Se nenhum valor for especificado, utiliza-se a aceleração global das articulações.
- **relative** (*bool*) – Indica se as posições das articulações são consideradas como um deslocamento relativo em relação às posições atuais. Por padrão, são usados valores de posição absoluta.
- **block** (*bool*) – Indica se deve aguardar a conclusão deste movimento antes de continuar a execução do programa. Apenas movimentos sincronizados usando as mesmas articulações podem ser executados consecutivamente de forma não bloqueante.

Exemplos:

Movendo todas as juntas do robô sincronizadas para 0°, que é a posição ereta do robô:

```
# these are equivalent
move_joints(ALL_KIN_JOINTS, 0)
move_joints([1, 2, 3, 4, 5, 6], [0, 0, 0, 0, 0, 0])
```

Movendo a junta 6 em -30° a partir de sua posição atual:

```
move_joints(6, -30, relative=True)
```

Movendo as juntas 2, 3 e 5 para 20°, -40° e 20°, respectivamente. O movimento será realizado de forma que cada junta termine seu movimento ao mesmo tempo. Este movimento usará a aceleração máxima permitida para as juntas, mas apenas 50% da velocidade máxima permitida.

```
move_joints([2, 3, 5], [20, -40, 20], joint_velocity=50,
joint_acceleration=100)
```

move_coordinates(*coordinates*, *configuration=None*, *joint_velocity=None*, *joint_acceleration=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *relative=False*, *block=True*)

Movendo o TCP (ponto central da ferramenta) do robô para um conjunto específico de coordenadas cartesianas. O robô se move ao longo de uma trajetória temporalmente ótima, resultando em perfis de velocidade trapezoidais em cada junta.

Parâmetros:

- **coordinates** (*Coordinates, list or dict*) –
  As coordenadas alvo do TCP. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}
- **configuration** (*str or None*) – A configuração da pose alvo. Isto é uma string de “c1” até “c8”.
- **joint_velocity** (*float or None*) – A velocidade máxima em percentagem do limite de velocidade das articulações. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *45* %). Se nenhum valor for especificado, utiliza-se a velocidade global das articulações.
- **joint_acceleration** (*float or None*) – A aceleração máxima em percentagem do limite de aceleração das articulações em movimento. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *60* %). Se nenhum valor for especificado, utiliza-se a aceleração global das articulações.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

- **relative** (*bool*) – Se as coordenadas são consideradas como um deslocamento relativo às coordenadas atuais. Por padrão, são usados valores de coordenadas absolutos.
- **block** (*bool*) – Se deve aguardar a continuação da execução do programa até que este movimento seja concluído. Por padrão, esta função é bloqueante.

Exemplos:

As coordenadas de destino podem ser fornecidas de várias formas. Todos os comandos especificados movem o TCP para a posição x = 700 mm, y = -125 mm, z = 500 mm e ângulos de orientação roll = 180°, pitch = 0° e yaw = -90° no sistema de coordenadas de trabalho global.

```
move_coordinates([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])
move_coordinates({"x": 700.0, "y": -125.0, "z": 500.0,
                  "roll": 180.0, "pitch": 0.0, "yaw": -90.0})
move_coordinates(Coordinates([700.0, -125.0, 500.0, 180.0, 0.0, -90.0]))
```

Se nem todas as coordenadas forem especificadas, as coordenadas atuais do TCP serão usadas para as restantes. Por exemplo, alterar apenas a componente x e roll pode ser feito da seguinte forma:

```
move_coordinates([500.0, None, None, 150.0, None, None])
move_coordinates({"x": 500.0, "roll": 150.0})
```

Nos exemplos acima, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função:

```
move_coordinates({"y": -100.0, "yaw": -60.0},
                 joint_velocity=25, joint_acceleration=100,
                 relative=True, tool_offset=[20, 0, 0, 0, 0, 0])
```

que move adicionalmente -100 mm na direção y e -60° na direção yaw com 50% da velocidade máxima e com a aceleração máxima. Além disso, o TCP é deslocado 20mm na direção x pelo deslocamento da ferramenta.

Existem várias opções para definir o movimento no espaço da ferramenta, uma delas é:

```
move_coordinates(get_reference_coordinates(), tool_offset=[0.0, 0.0, 100.0,
0.0, 0.0, 0.0])
```

que se move ao longo da direção z da ferramenta.

move_pose(*pose*, *joint_velocity=None*, *joint_acceleration=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *block=True*)

Movendo o TCP (ponto central da ferramenta) do robô para uma pose específica. O robô se move ao longo de uma trajetória de tempo ótimo, resultando em perfis de velocidade trapezoidais em cada junta.

Parâmetros:

- **pose** (*Pose, str, or int*) –
  A pose alvo do TCP. Especifique o valor da seguinte forma:
  - um objeto Pose
  - o nome de uma pose pré-definida (salva no banco de dados)
  - o ID de uma pose pré-definida (salva no banco de dados)
- **joint_velocity** (*float or None*) – A velocidade máxima em percentagem do limite de velocidade das articulações. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *45* %). Se nenhum valor for especificado, utiliza-se a velocidade global das articulações.
- **joint_acceleration** (*float or None*) – A aceleração máxima em percentagem do limite de aceleração das articulações em movimento. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *60* %). Se nenhum valor for especificado, utiliza-se a aceleração global das articulações.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates

- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.
- **block** (*bool*) – Se deve aguardar a continuação da execução do programa até que este movimento seja concluído. Por padrão, esta função é bloqueante.

Exemplos:

Movendo para uma pose do banco de dados:

```
move_pose("fancy_pose")
move_pose(10)   # ID of the pose in the database
```

Uma pose personalizada também pode ser especificada no script usando a função `create_pose()`:

```
pose = create_pose([700.0, -125.0, 500.0, 180.0, 0.0, -90.0], "c8")
move_pose(pose)
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
move_pose("final_pose", joint_velocity=45, joint_acceleration=60,
          work_offset=[0, 0, 50, 0, 0, 0], block=False)
```

Isso move o robô para a «final_pose» do banco de dados com 45% da velocidade máxima das juntas e 60% da aceleração máxima das juntas. Além disso, um deslocamento é aplicado ao referencial de trabalho, deslocando a pose alvo 50 mm na direção z.

Existem várias opções para definir o movimento no espaço da ferramenta, uma delas é:

```
move_pose(get_reference_pose(), tool_offset=[0.0, 0.0, 100.0, 0.0, 0.0, 0.0])
```

que se move ao longo da direção z da ferramenta.

move_linear(*coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *relative=False*, *block=True*)

Movendo o TCP (ponto central da ferramenta) do robô da pose atual para uma pose alvo em linha reta. Note que tanto a posição quanto a orientação são interpoladas linearmente ao longo da trajetória. Este movimento só pode ser executado se cada pose ao longo do trajeto for alcançável.

Parâmetros:

- **coordinates** (*Coordinates, list, or dict*) –
  As coordenadas alvo do TCP. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}
- **linear_velocity** (*float or None*) – A velocidade linear máxima da ferramenta [mm/s] para todas as partes móveis do robô.
- **linear_acceleration** (*float or None*) – A aceleração linear máxima da ferramenta [mm/s^2] para todas as partes móveis do robô.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates

- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.
- **relative** (*bool*) – Se as coordenadas são consideradas como um deslocamento relativo às coordenadas atuais. Por padrão, são usados valores de coordenadas absolutos.
- **block** (*bool*) – Se deve aguardar a continuação da execução do programa até que este movimento seja concluído. Por padrão, esta função é bloqueante.

Exemplos:

As coordenadas alvo podem ser fornecidas de várias formas. Todos os comandos especificados movem o TCP linearmente para a posição x = 600 mm, y = -125 mm, z = 500 mm e ângulos de orientação roll = 180°, pitch = 0° e yaw = -90° no referencial de trabalho global.

```
move_linear([600.0, -125.0, 500.0, 180.0, 0.0, -90.0])
move_linear({"x": 600.0, "y": -125.0, "z": 500.0,
             "roll": 180.0, "pitch": 0.0, "yaw": -90.0})
move_linear(Coordinates([600.0, -125.0, 500.0, 180.0, 0.0, -90.0]))
```

Se nem todas as coordenadas forem especificadas, as coordenadas atuais do TCP são usadas para as coordenadas restantes. Por exemplo, os componentes x e roll podem ser alterados da seguinte forma:

```
move_linear([500.0, None, None, 150.0, None, None])
move_linear({"x": 500.0, "roll": 150.0})
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
move_linear({"x": 100.0}, linear_velocity=500, linear_acceleration=1000,
            relative=True, work_frame=[20, 0, 0, 0, 0, 0])
```

Isso move adicionalmente 100 mm na direção x com uma velocidade de 500 mm/s e uma aceleração de 1000 mm/s^2. Além disso, o referencial de trabalho é sobrescrito com as coordenadas fornecidas.

Existem várias opções para definir o movimento no espaço da ferramenta, uma delas é:

```
move_linear(get_reference_coordinates(), tool_offset=[0.0, 0.0, 100.0, 0.0,
0.0, 0.0])
```

Isso move linearmente ao longo da direção z da ferramenta.

move_circular(*intermediate_position*, *coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *relative=False*, *block=True*)

Movendo o TCP (ponto central da ferramenta) do robô da pose atual para uma pose alvo ao longo de uma trajetória circular, passando por uma posição intermediária. Observe que a orientação é interpolada linearmente ao longo da trajetória. Este movimento só pode ser executado se cada pose ao longo do trajeto for alcançável.

Parâmetros:

- **intermediate_position** (*list*) – Uma posição intermediária no caminho para as coordenadas alvo. A posição intermediária determina a curvatura do movimento circular.
- **coordinates** (*Coordinates, list, or dict*) –
  As coordenadas alvo do TCP. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}
- **linear_velocity** (*float or None*) – A velocidade linear máxima da ferramenta [mm/s] para todas as partes móveis do robô.
- **linear_acceleration** (*float*) – A aceleração linear máxima da ferramenta [mm/s^2] para todas as partes móveis do robô.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –

A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

- **relative** (*bool*) – Se as coordenadas são consideradas como um deslocamento relativo às coordenadas atuais. Por padrão, são usados valores de coordenadas absolutos.
- **block** (*bool*) – Se deve aguardar a continuação da execução do programa até que este movimento seja concluído. Por padrão, esta função é bloqueante.

Exemplos:

O robô pode ser movido circularmente para a posição alvo x = 0 mm, y = -500 mm, z = 500 mm e ângulos de orientação roll = 180°, pitch = 0° e yaw = -90° no referencial de trabalho global. Durante o movimento, passará pela posição intermediária x = 0 mm, y = -500 mm, z = 500 mm.

```
move_circular([0.0, -500.0, 500.0], [-500.0, -125.0, 500.0, 180.0, 0.0,
-90.0])
```

Se nem todas as coordenadas forem especificadas, as coordenadas atuais do TCP são usadas para as coordenadas restantes.

```
move_circular([0.0, -500.0, None], [-500.0, -125.0, None, None, None, None])
move_circular([0.0, -500.0, None], {"x": -500.0, "y": -125.0})
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
move_circular([500.0, -300.0, 100.0], [-1000.0, 0.0, 0.0, 20.0, -10.0, 0.0],
              linear_velocity=500, linear_acceleration=1000,
              tool_offset={"x": 20}, relative=True)
```

Isso move o TCP para as coordenadas especificadas com uma velocidade de 500 mm/s e uma aceleração de 1000 mm/s². Além disso, o TCP é deslocado 20 mm na direção x pelo offset da ferramenta.

move_orientation(*tool_orientation*, *angular_velocity=None*, *angular_acceleration=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *relative=False*, *block=True*)

Movendo o TCP (ponto central da ferramenta) de modo que apenas a orientação mude. O robô então gira linearmente de acordo com a orientação desejada da ferramenta, mantendo a posição constante. Este movimento só pode ser executado se cada pose ao longo do trajeto for alcançável.

Parâmetros:

- **tool_orientation** (*list*) – Orientação desejada da ferramenta como ângulos náuticos (roll, pitch, yaw) [°]
- **angular_velocity** (*float or None*) – Velocidade angular máxima da ferramenta [°/s] do TCP.
- **angular_acceleration** (*float or None*) – Aceleração angular máxima da ferramenta [°/s²] do TCP.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.
- **relative** (*bool*) – Se as coordenadas são consideradas como um deslocamento relativo às coordenadas atuais. Por padrão, são usados valores de coordenadas absolutos.
- **block** (*bool*) – Se deve aguardar a continuação da execução do programa até que este movimento seja concluído. Por padrão, esta função é bloqueante.

Exemplos:

A orientação alvo pode especificar todos os ângulos ou apenas alguns deles, usando "None" para os demais. Neste caso, a orientação atual é usada para os ângulos que faltam.

```
move_orientation([150.0, -20.0, -60.0])
move_orientation([150.0, None, -60.0])
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
move_orientation([-60, 0, 0], angular_velocity=250, angular_acceleration=3000,
                 tool_frame=[0, 0, 0, 0, 20, 0], relative=True)
```

O que corresponde a um movimento relativo de -60° na direção de rotação com uma velocidade de 250°/s e aceleração de 3000°/s². Além disso, o quadro da ferramenta é sobrescrito com as coordenadas fornecidas.

Existem várias opções para definir o movimento no espaço da ferramenta, uma delas é:

```
move_orientation(get_reference_coordinates().orientation, tool_offset=[0.0,
0.0, 0.0, 10.0, 0.0, 0.0])
```

Que gira em torno da direção z da ferramenta.

move_waypoints(*poses*, *joint_velocity=None*, *joint_acceleration=None*, *blend_radius=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *block=True*)

Move o TCP (ponto central da ferramenta) através dos waypoints definidos, começando na pose atual do robô.

Parâmetros:

- **poses** (*list of str, int or Pose*) – Uma lista de waypoints.
- **joint_velocity** (*float or None*) – A velocidade máxima em percentagem do limite de velocidade das articulações. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *45* %). Se nenhum valor for especificado, utiliza-se a velocidade global das articulações.
- **joint_acceleration** (*float or None*) – A aceleração máxima em percentagem do limite de aceleração das articulações em movimento. Este é um único número para limitar todas as articulações ao mesmo valor máximo (ex. *60* %). Se nenhum valor for especificado, utiliza-se a aceleração global das articulações.
- **blend_radius** (*float or None*) – Raio de mistura permitido [mm] definindo a máxima desvição dos waypoints no espaço da ferramenta durante a transição entre waypoints.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates

- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

- **block** (*bool*) – Indica se deve aguardar a conclusão deste movimento antes de continuar a execução do programa. Apenas movimentos sincronizados usando as mesmas articulações podem ser executados consecutivamente de forma não bloqueante.

Exemplos:

Os waypoints podem ser fornecidos em diferentes formas (`Pose` objeto, nome da pose na base de dados, ID da pose na base de dados):

```
first_waypoint = create_pose([700.0, -125.0, 500.0, 180.0, 0.0, -90.0], "c8")
move_waypoints([first_waypoint, "pick_pose", 6])
```

Nos exemplos acima, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função:

```
move_waypoints([first_waypoint, "pick_pose", 6], joint_velocity=25,
               joint_acceleration=100, blend_radius=50)
```

Que se move ao longo dos waypoints permitindo uma desviação máxima de 50mm. Usa 50% da velocidade máxima e a aceleração máxima.

move_segments(*segments*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *block=True*)

Move o TCP (ponto central da ferramenta) de acordo com os segmentos definidos. Começa na pose atual do robô.

Parâmetros:

- **segments** (*list*) – Uma lista de segmentos que compõem o caminho.
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.
- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]

- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

- **block** (*bool*) – Indica se deve aguardar a conclusão deste movimento antes de continuar a execução do programa. Apenas movimentos sincronizados usando as mesmas articulações podem ser executados consecutivamente de forma não bloqueante.

Exemplos:

Os segmentos podem ser definidos no script usando os objetos Segment (`LinearSegment` , `CircularSegment` , ...):

```
lin_seg = LinearSegment([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])
pose_seg = PoseSegment("place_2")

move_segments([lin_seg, pose_seg])
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
lin_seg = LinearSegment([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])
pose_seg = PoseSegment("place_2")

move_segments([lin_seg, pose_seg], tool_offset=[20, 0, 0, 0, 0, 0],
block=False)
```

Que executa os segmentos usando um pequeno deslocamento no TCP definido pelo offset da ferramenta. Além disso, o movimento é configurado como não bloqueante.

run_path(*path*, *previous_angles=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*, *block=True*)

Move o TCP (ponto central da ferramenta) de acordo com o caminho especificado. Primeiro, o robô é movido para a pose inicial do caminho usando as velocidades e acelerações padrão. A partir daí, os segmentos são executados um após o outro de acordo com sua especificação.

Parâmetros:

- **path** (*Path, str, or int*) – O caminho que o TCP deve seguir.
- **previous_angles** (*list or None*) –
  Os ângulos para os quais será calculada a solução mais próxima para a pose inicial. Especifique o valor da seguinte forma:
  - None, para usar os ângulos atuais do robô
  - Uma lista de ângulos das articulações [deg]
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
    - um objeto Coordinates
    - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
    - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

- **block** (*bool*) – Indica se deve aguardar a conclusão deste movimento antes de continuar a execução do programa. Apenas movimentos sincronizados usando as mesmas articulações podem ser executados consecutivamente de forma não bloqueante.

Exemplos:

Executar um caminho da base de dados:

```
run_path("fancy_path")
run_path(10)   # ID of the path in the database
```

Um caminho personalizado também pode ser especificado no script usando a função `create_path()`:

```
path = create_path(start_pose, [LinearSegment(...), ...])
run_path(path)
```

Outra opção é usar o objeto `Path` para criar um caminho e executá-lo:

```
path = Path("start_pose", [])
path.add_linear([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])

run_path(path)
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
run_path("fancy_path", tool_frame=[20, 0, 0, 0, 0, 0], block=False)
```

Que executa o «fancy_path» usando um pequeno deslocamento no TCP definido pelo quadro da ferramenta. Além disso, o movimento é configurado como não bloqueante.

wait_for_motor(*actuator_ids=None*)

Esperando que um ou mais motores de junta parem de se mover.

Parâmetros:

actuator_ids (*None, str, int, list, set*) –

Os IDs dos atuadores a aguardar. Especifique o valor da seguinte forma:

- o ID de uma única articulação (ex. *6*)
- uma lista de IDs para várias articulações (ex. *[1, 4, 6]*)
- um conjunto de nomes de juntas para várias juntas (ex.: *{«1», «4», «6»}*)
- a constante *ALL_KIN_JOINTS* ou *None* para ler de todas as juntas

Exemplos:

Este exemplo mostra que o tempo em que um robô se move pode ser usado para fins personalizados.

```
# move to the 'start pose'
move_pose("start_pose", block=True)
# start moving towards the 'end pose'
move_pose("end_pose", block=False)
# the idle time can be used e.g. for calculations, observations, etc.
#custom_function()
# wait for the non-blocking motion to end
wait_for_motor()
```

stop_motion()

Parando o movimento enviando um sinal de parada para todos os atuadores conectados.

Exemplos:

Este exemplo mostra que movimentos não bloqueantes podem ser abortados usando a função “stop_motion”. Aqui queremos abortar o movimento se uma entrada digital específica for alternada.

```
# move to the 'start pose'
move_pose("start_pose", block=True
# start checking for the value of a digital input
initial_value = read_digital_main_input(1)
# start moving towards the 'end pose'
move_pose("target_pose", block=False)
# stop the motion as soon as the value of the digital input toggles
while is_motor_moving() and read_digital_main_input(1) == initial_value:
    wait(0.1)
stop_motion()
```

is_motor_moving(*actuator_ids=None*)

Verificar se um ou vários motores estão se movendo

Parâmetros:

actuator_ids (*None, str, int, list, set*) –

Os IDs dos atuadores a verificar. Especifique o valor da seguinte forma:

- o ID de uma única articulação (ex. *6*)
- uma lista de IDs para várias articulações (ex. *[1, 4, 6]*)
- um conjunto de nomes de juntas para várias juntas (ex.: *{1, 4, 6}*)
- a constante *ALL_KIN_JOINTS* ou *None* para ler de todas as juntas

Retorno:

Se os motores especificados estão em movimento. Pode ser um dos seguintes:

- um booleano, se apenas um motor foi verificado
- uma lista de booleanos, se vários motores foram verificados

Tipo de retorno:

bool or list of bool

Exemplos:

Este exemplo mostra que movimentos não bloqueantes podem ser abortados usando a função “stop_motion”. Aqui queremos abortar o movimento se uma entrada digital específica for alternada.

```
# move to the 'start pose'
move_pose("start_pose", block=True
# start checking for the value of a digital input
initial_value = read_digital_main_input(1)
# start moving towards the 'end pose'
move_pose("target_pose", block=False)
# stop the motion as soon as the value of the digital input toggles
while is_motor_moving() and read_digital_main_input(1) == initial_value:
    wait(0.1)
stop_motion()
```

set_global_joint_velocity(*joint_velocity*)

Definindo um novo valor padrão global para o argumento `joint_velocity` usado nas funções de movimento ponto a ponto. O valor da velocidade da junta é dado em percentual do limite máximo de velocidade da junta.

Parâmetros:

joint_velocity (*float*) – A velocidade global da junta em [%] da velocidade máxima da junta.

Exemplos:

Executando dois movimentos consecutivos ponto-a-ponto usando os mesmos limites de velocidade.

```
set_global_joint_velocity(40)
move_pose("pose_1")
move_pose("pose_2")
# ... this is similar to ...
move_pose("pose_1", joint_velocity=40)
move_pose("pose_2", joint_velocity=40)
```

set_global_joint_acceleration(*joint_acceleration*)

Definindo um novo valor padrão global para o argumento `joint_acceleration` usado em funções de movimento ponto-a-ponto. O valor da aceleração da junta é dado em porcentagem do limite máximo de aceleração da junta.

Parâmetros:

joint_acceleration (*float*) – A aceleração global da junta em [%] da aceleração máxima da junta.

Exemplos:

Executando dois movimentos consecutivos ponto-a-ponto usando os mesmos limites de aceleração.

```
set_global_joint_acceleration(80)
move_pose("pose_1")
move_pose("pose_2")
# ... this is similar to ...
move_pose("pose_1", joint_acceleration=80)
move_pose("pose_2", joint_acceleration=80)
```

set_global_linear_velocity(*linear_velocity*)

Definindo um novo valor padrão global para o argumento `linear_velocity` usado em funções de movimento de ferramenta e planejador de caminho. O valor da velocidade linear limita a velocidade linear máxima da ferramenta e de todas as partes móveis do robô.

Parâmetros:

linear_velocity (*float*) – A velocidade linear global em [mm/s].

Exemplos:

Executando dois movimentos lineares consecutivos usando os mesmos limites de velocidade.

```
set_global_linear_velocity(150)
move_linear("pose_1")
move_linear("pose_2")
# ... this is similar to ...
move_linear("pose_1", linear_velocity=150)
move_linear("pose_2", linear_velocity=150)
```

set_global_linear_acceleration(*linear_acceleration*)

Definindo um novo valor padrão global para o argumento `linear_acceleration` usado em funções de movimento de ferramenta e planejador de caminho. O valor da aceleração linear limita a aceleração linear máxima da ferramenta e de todas as partes móveis do robô.

Parâmetros:

linear_acceleration (*float*) – A aceleração linear global em [mm/s^2].

Exemplos:

Executando dois movimentos lineares consecutivos usando os mesmos limites de aceleração.

```
set_global_linear_acceleration(600)
move_linear("pose_1")
move_linear("pose_2")
# ... this is similar to ...
move_linear("pose_1", linear_acceleration=600)
move_linear("pose_2", linear_acceleration=600)
```

set_global_angular_velocity(*angular_velocity*)

Definindo um novo valor padrão global para o argumento `angular_velocity` usado em funções de movimento rotacional. O valor da velocidade angular limita a velocidade de rotação da ferramenta.

Parâmetros:

angular_velocity (*float*) – A velocidade angular global em [°/s].

Exemplos:

Executando dois movimentos rotacionais consecutivos usando os mesmos limites de velocidade.

```
set_global_angular_velocity(20)
move_orientation("pose_1")
move_orientation("pose_2")
# ... this is similar to ...
move_orientation("pose_1", angular_velocity=20)
move_orientation("pose_2", angular_velocity=20)
```

set_global_angular_acceleration(*angular_acceleration*)

Definindo um novo valor padrão global para o argumento `angular_acceleration` usado em funções de movimento rotacional. O valor da aceleração angular limita a aceleração rotacional da ferramenta.

Parâmetros:

angular_acceleration (*float*) – A aceleração angular global em [°/s^2].

Exemplos:

Executando dois movimentos rotacionais consecutivos usando os mesmos limites de aceleração.

```
set_global_angular_acceleration(45)
move_orientation("pose_1")
move_orientation("pose_2")
# ... this is similar to ...
move_orientation("pose_1", angular_acceleration=45)
move_orientation("pose_2", angular_acceleration=45)
```

set_global_blend_radius(*blend_radius*)

Definindo um novo valor padrão global para o argumento `blend_radius` usado em funções de movimento e de trajetória.

Parâmetros:

blend_radius (*float*) – O raio de blend global em [mm].

Exemplos:

Executando dois movimentos consecutivos de waypoints usando o mesmo raio de blend.

```
set_global_blend_radius(100)
move_waypoints(["pose_1, "pose_2"])
# ... this is similar to ...
move_waypoints(["pose_1, "pose_2"], blend_radius=100)
```

set_global_work_frame(*work_frame*)

Definindo um novo valor padrão global para o argumento `work_frame` usado em funções de movimento e cinemática. O frame de trabalho é a transformação para alterar o quadro de referência do robô, dado em relação ao frame base.

Parâmetros:

work_frame (*Coordinates, list, or dict*) –

O referencial de trabalho global, fornecido em relação ao referencial base. Especifique o valor da seguinte forma:

- um objeto Coordinates
- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Exemplos:

Executando dois movimentos consecutivos usando o mesmo referencial de trabalho.

```
my_work_frame = Coordinates([450, -300, 0, 0, 0, 90])

set_global_work_frame(my_work_frame)
move_pose("pose_1")
move_linear("pose_2")
# ... this is similar to ...
move_pose("pose_1", work_frame=my_work_frame)
move_linear("pose_2", work_frame=my_work_frame)
```

set_global_tool_frame(*tool_frame*)

Definindo um novo valor padrão global para `tool_frame`, usado em funções de movimento e cinemática. O referencial da ferramenta é a transformação para alterar o TCP do robô, fornecido em relação ao flange.

Parâmetros:

tool_frame (*Coordinates, list, or dict*) –

O referencial da ferramenta global, fornecido em relação ao flange. Especifique o valor da seguinte forma:

- um objeto Coordinates
- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Exemplos:

Executando dois movimentos consecutivos usando o mesmo referencial da ferramenta.

```
my_tool_frame = Coordinates([0, 0, 65, 0, 0, 180])

set_global_tool_frame(my_tool_frame)
move_pose("pose_1")
move_linear("pose_2")
# ... this is similar to ...
move_pose("pose_1", tool_frame=my_tool_frame)
move_linear("pose_2", tool_frame=my_tool_frame)
```

get_current_pose(*tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Lendo a pose atual do TCP (ponto central da ferramenta) do robô. Se os frames não forem fornecidos, a pose é lida em relação aos frames globais.

Parâmetros:

- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.

- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

A pose atual do TCP do robô.

Tipo de retorno:

Pose

Exemplos:

Lendo as propriedades de uma pose de duas maneiras diferentes.

```
pose = get_current_pose()
coordinates = pose.coordinates
configuration = pose.configuration
# ... this is similar to ...
coordinates = get_current_coordinates()
configuration = get_current_configuration()
```

Observe que a pose atual é lida a partir de dados de sensores. Os sensores podem flutuar e nem sempre retornar valores precisos, portanto o exemplo a seguir de mover-se para frente e para trás em relação à pose atual levará a um desvio inesperado ao longo do tempo. Nesse caso, é melhor usar a função *get_reference_pose*.

```
while True:
    move_pose(get_current_pose(), tool_offset=[0.0, 0.0, 100.0, 0.0, 0.0,
0.0])
    move_pose(get_current_pose(), tool_offset=[0.0, 0.0, -100.0, 0.0, 0.0,
0.0])
```

get_current_coordinates(*tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Lendo as coordenadas atuais do TCP (ponto central da ferramenta) do robô em relação aos frames de coordenadas fornecidos. Se nenhum frame for especificado, retorna as coordenadas em relação aos frames globais.

Parâmetros:

- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.

- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]

- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

As coordenadas atuais do TCP do robô.

Tipo de retorno:

Coordinates

Exemplos:

Lendo as coordenadas atuais do TCP nos frames de coordenadas globais.

```
coordinates = get_current_coordinates()
```

Note que as coordenadas atuais são lidas a partir de dados de sensores. Os sensores podem flutuar e nem sempre retornar valores precisos, portanto o exemplo a seguir de mover-se para frente e para trás em relação à pose atual levará a um desvio inesperado ao longo do tempo. Nesse caso, é melhor usar a função *get_reference_coordinates*.

```
while True:
    move_coordinates(get_current_coordinates(), tool_offset=[0.0, 0.0, 100.0,
0.0, 0.0, 0.0])
    move_coordinates(get_current_coordinates(), tool_offset=[0.0, 0.0, -100.0,
0.0, 0.0, 0.0])
```

get_current_configuration()

Lê o nome da configuração atual da pose do robô. Observe que essa configuração é independente dos frames globais atuais.

Retorno:

A configuração atual (que é uma das opções de “c1” a “c8”).

Tipo de retorno:

str

Exemplos:

Lê a configuração da pose atual do robô de duas maneiras diferentes.

```
configuration = get_current_configuration()
# ... this is similar to ...
pose = get_current_pose()
configuration = f"c{pose.configuration + 1}"
```

get_reference_pose(*tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Obtém a pose de referência do TCP (tool center point) do robô. Se nenhum frame for fornecido, a pose é retornada em relação aos frames globais. Note que essa pose é calculada a partir dos ângulos de referência enviados aos acionamentos.

Parâmetros:

- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.

- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]

- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

A pose de referência do TCP do robô.

Tipo de retorno:

Pose

Exemplos:

Lê as propriedades de uma pose de referência de duas maneiras diferentes.

```
pose = get_reference_pose()
coordinates = pose.coordinates
configuration = pose.configuration
# ... this is similar to ...
coordinates = get_reference_coordinates()
configuration = get_reference_configuration()
```

get_reference_coordinates(*tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Obtém as coordenadas de referência do TCP (tool center point) do robô em relação aos frames de coordenadas fornecidos. Se nenhum frame for especificado, as coordenadas são retornadas em relação aos frames globais. Observe que essas coordenadas são calculadas a partir dos ângulos de referência enviados aos acionamentos.

Parâmetros:

- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.

- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]

- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

As coordenadas de referência do TCP do robô.

Tipo de retorno:

Coordinates

Exemplos:

Lê as coordenadas de referência do TCP nos frames de coordenadas globais.

```
coordinates = get_reference_coordinates()
```

get_reference_configuration()

Obtém o nome da configuração da pose de referência do robô. Essa configuração é independente dos frames globais de referência e é calculada a partir dos ângulos de referência enviados aos acionamentos.

Retorno:

A configuração de referência (que é uma das opções de “c1” a “c8”).

Tipo de retorno:

str

Exemplos:

Lê a configuração da pose de referência do robô de duas maneiras diferentes.

```
configuration = get_reference_configuration()
# ... this is similar to ...
pose = get_reference_pose()
configuration = f"c{pose.configuration + 1}"
```

create_pose(*coordinates*, *configuration*)

Cria um objeto Pose com base nas coordenadas e na configuração fornecidas.

Parâmetros:

- **coordinates** (*Coordinates, list, or dict*) –
  As coordenadas TCP da pose. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}
- **configuration** (*str*) – A configuração da pose. Esta é uma das opções de “c1” a “c8”.

Retorno:

O objeto de pose do robô.

Tipo de retorno:

Pose

Exemplos:

Cria uma pose a partir de uma lista de valores de coordenadas, em uma configuração específica.

```
pose = create_pose([350.0, 0.0, 550.0, 0.0, -30.0, 0.0], "c1")
```

get_pose(*pose*)

Lê o objeto de pose a partir de uma pose predefinida salva no banco de dados.

Parâmetros:

pose (*Pose, str, or int*) –

A pose do robô da qual as coordenadas serão lidas. Pode ser uma das seguintes:

- um objeto Pose
- o nome (str) de uma pose salva no banco de dados.
- o ID (int) de uma pose salva no banco de dados

Retorno:

O objeto de pose.

Tipo de retorno:

Pose

Exemplos:

Carregando uma pose do banco de dados com o nome «grasping_pose».

```
pose = get_pose("grasping_pose")
```

create_path(*initial_pose*, *segments*)

Cria um objeto Path com base na pose inicial e nos segmentos fornecidos.

Parâmetros:

- **initial_pose** (*Pose*) – A pose inicial do robô.
- **segments** (*list*) – Uma lista de segmentos dos quais o caminho é composto. Observe que segmentos também podem ser adicionados posteriormente.

Retorno:

O objeto de caminho do robô.

Tipo de retorno:

Path

Exemplos:

Criando um caminho a partir de uma pose inicial e vários segmentos.

```
path = create_path(start_pose, [LinearSegment(...), ...])
```

get_path(*path*)

Lendo o objeto de caminho a partir de um caminho predefinido salvo no banco de dados.

Parâmetros:

path (*Path, str, or int*) –

O caminho do robô a ser obtido. Pode ser um dos seguintes:

- um objeto Path
- o nome (str) de um caminho salvo no banco de dados
- o ID (int) de um caminho salvo no banco de dados

Retorno:

O objeto de caminho.

Tipo de retorno:

Path

Exemplos:

Carregando um caminho do banco de dados com o nome «grasping_path».

```
pose = get_path("grasping_path")
```

parse_linear_segment(*coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Criar um objeto LinearSegment.

Parâmetros:

- **coordinates** (*list, dict, or Coordinates*) – Coordenadas de destino.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto LinearSegment definindo todos os parâmetros:

```
linear_seg = parse_linear_segment([100, 200, 300, 15, -15, 30],
                                  linear_velocity=500,
linear_acceleration=1000,
                                  blend_radius=100)
```

ou usando os valores padrão:

```
linear_seg = parse_linear_segment([100, 200, 300, 15, -15, 30])
```

parse_circular_segment(*intermediate_position*, *coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Criar um objeto CircularSegment.

Parâmetros:

- **intermediate_position** (*list*) – Uma posição intermediária [x, y, z] no caminho até as coordenadas de destino. A posição intermediária determina a curvatura do movimento circular.
- **coordinates** (*list, dict or Coordinates*) – Coordenadas de destino.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto CircularSegment definindo todos os parâmetros:

```
circular_seg = parse_circular_segment([0, 300, 300], [100, 200, 300, 15, -15,
30],
                                      linear_velocity=500,
linear_acceleration=1000,
                                      blend_radius=100)
```

ou usando os valores padrão:

```
circular_seg = parse_circular_segment([0, 300, 300], [100, 200, 300, 15, -15,
30])
```

parse_orientation_segment(*orientation*, *angular_velocity=None*, *angular_acceleration=None*, *blend_radius=None*)

Criar um objeto OrientationSegment.

Parâmetros:

- **orientation** (*list*) – Orientação alvo. Observe que a posição permanece constante.
- **angular_velocity** (*float or None*) – Velocidade angular desejada [°/s]. Se nenhuma velocidade angular for definida, será usada a velocidade angular global.
- **angular_acceleration** (*float or None*) – Aceleração angular desejada [°/s²]. Se nenhuma aceleração angular for definida, será usada a aceleração angular global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Um raio de blend maior que 0 permite que o TCP (ponto central da ferramenta) gire antes que a posição deste segmento seja alcançada. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto OrientationSegment definindo todos os parâmetros:

```
orientation_seg = parse_orientation_segment([15, -15, 30],
                                            angular_velocity=250,
angular_acceleration=500,
                                            blend_radius=100)
```

ou usando os valores padrão:

```
orientation_seg = parse_orientation_segment([15, -15, 30])
```

parse_coordinates_segment(*coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Criar um objeto CoordinatesSegment.

Parâmetros:

- **coordinates** (*list, dict or Coordinates*) – Coordenadas de destino.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto CoordinatesSegment definindo todos os parâmetros:

```
coordinates_seg = parse_coordinates_segment([100, 200, 300, 15, -15, 30],
                                            linear_velocity=250,
linear_acceleration=500,
                                            blend_radius=100)
```

ou usando os valores padrão:

```
coordinates_seg = parse_coordinates_segment([100, 200, 300, 15, -15, 30])
```

parse_pose_segment(*pose*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Criar um objeto PoseSegment.

Parâmetros:

- **pose** (*Pose*) – Pose alvo.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto PoseSegment definindo todos os parâmetros:

```
pose_seg = parse_pose_segment(create_pose([100, 200, 300, 15, -15, 30], "c1"),
                              linear_velocity=250, linear_acceleration=500,
                              blend_radius=100)
```

ou usando os valores padrão e a pose do banco de dados:

```
pose_seg = parse_pose_segment("target_pose")
```

get_coordinates(*pose*)

Lendo as coordenadas da pose especificada. Observe que essas coordenadas são independentes dos frames globais atuais.

Parâmetros:

pose (*Pose, str, or int*) –

A pose do robô da qual as coordenadas serão lidas. Pode ser uma das seguintes:

- um objeto Pose
- o nome (str) de uma pose salva no banco de dados.
- o ID (int) de uma pose salva no banco de dados

Retorno:

As coordenadas desta pose.

Tipo de retorno:

Coordinates

Exemplos:

Lendo as coordenadas da pose salva anteriormente no banco de dados com o nome «grasping_pose».

```
coordinates = get_coordinates("grasping_pose")
```

Lendo as coordenadas da pose salva anteriormente no banco de dados com o ID 3.

```
coordinates = get_coordinates(3)
```

get_configuration(*pose*)

Lendo a configuração de uma pose específica.

Parâmetros:

pose (*Pose, str, or int*) –

A pose da qual a configuração será lida. Especifique o valor da seguinte forma:

- um objeto Pose
- o nome de uma pose pré-definida (salva no banco de dados)
- o ID de uma pose pré-definida (salva no banco de dados)

Retorno:

A configuração desta pose (que é uma das opções de “c1” a “c8”).

Tipo de retorno:

str

Exemplos:

Lendo a configuração de uma pose salva no banco de dados com o nome «grasping_pose».

```
configuration = get_configuration("grasping_pose")
```

Lendo a configuração de um objeto Pose de duas maneiras diferentes.

```
configuration = get_configuration(pose)
# ... this is similar to ...
configuration = f"c{pose.configuration + 1}"
```

parse_coordinates(*coordinates*)

Criar um objeto Coordinates.

Parâmetros:

coordinates (*list, dict, or Coordinates*) –

Um conjunto específico de coordenadas, que pode ser um dos seguintes:

- Uma lista na forma [x, y, z, roll, pitch, yaw] com valores de posição em [mm] e valores de orientação em [°]
- Um dicionário na forma {“x”: x, “y”: y, “z”: z, “roll”: roll, “pitch”: pitch, “yaw”: yaw} com valores de posição em [mm] e valores de orientação em [°]
- Um objeto Coordinates (copiando outro objeto Coordinates)

Construtor:

Cria um objeto Coordinates a partir de valores de coordenadas.

```
coordinates = parse_coordinates({"x": 100, "y": 200, "z": 300,
                                 "roll": 15, "pitch": -15, "yaw": 30})
coordinates = parse_coordinates([100, 200, 300, 15, -15, 30])
```

transform_coordinates(*coordinates*, *relative_tool_frame=None*, *relative_work_frame=None*)

Transformando as coordenadas do robô entre diferentes frames usando a diferença entre os frames relativos de ferramenta e de trabalho. Se nem o frame relativo da ferramenta nem o frame relativo de trabalho for especificado, as coordenadas não serão transformadas, retornando o objeto de coordenadas inalterado.

Parâmetros:

- **coordinates** (*Coordinates, list, or dict*) –
  As coordenadas TCP a transformar. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}
- **relative_tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **relative_work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

As coordenadas do robô transformadas.

Tipo de retorno:

Coordinates

Exemplos:

Transformar um conjunto de coordenadas para outro frame de trabalho. Se a origem do frame de trabalho estiver exatamente onde as coordenadas a transformar se encontram, a transformação retorna um objeto Coordinates com zeros para todos os valores.

```
coordinates = [150, -250, 600, 15, -60, -15]
transformed_coordinates = transform_coordinates(coordinates,
relative_work_frame=coordinates)
```

get_grid_points(*grid_name*)

Leitura das posições de todos os pontos da grelha a partir de uma grelha específica previamente guardada na base de dados.

Parâmetros:

grid_name (*str*) – O nome da grelha da qual extrair os pontos.

Retorno:

A lista de pontos da grelha, na ordem de iteração definida nas propriedades da grelha.

Tipo de retorno:

list

Exemplos:

Mover para todos os pontos numa grelha predefinida.

```
# defining the orientation with which to approach the grid points
orientation = [0.0, -85.0, 0.0]
# iterate through all grid points
for position in get_grid_points("sample_tray"):
    # move to each of the grid points
    coordinates = position + orientation
    move_coordinates(coordinates)
```

forward_kinematics(*joint_angles*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Cinemática direta é um conceito da cinemática robótica que calcula a pose do TCP (tool center point) a partir de um determinado conjunto de ângulos das articulações. O inverso desta função é a *cinemática inversa*, que calcula os ângulos das articulações a partir da pose do TCP do robô. Os cálculos cinemáticos são normalmente efetuados entre o frame base e o frame da flange integrados do robô. No entanto, o utilizador pode alterar os frames de referência definindo os frames de trabalho e de ferramenta. Por defeito, são utilizados os frames de trabalho e de ferramenta definidos globalmente.

Parâmetros:

- **joint_angles** (*list of float*) – A lista dos seis ângulos das articulações em [deg].
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.
- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:

- um objeto Coordinates
- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

O objeto de pose do robô.

Tipo de retorno:

Pose

Exemplos:

Numa singularidade, vários conjuntos de ângulos das articulações conduzem à mesma pose do TCP. Neste exemplo, ambos os conjuntos de ângulos resultam exatamente na mesma pose.

```
pose_1 = forward_kinematics([0, 0, 0, 0, 0, 0])
pose_2 = forward_kinematics([45, 0, 0, 45, 0, -90])
```

Leitura da pose atual do TCP utilizando duas abordagens diferentes.

```
pose = forward_kinematics(read_actuator_position())
# ... this is similar to ...
pose = get_pose()
```

inverse_kinematics(*pose*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Cinemática inversa é um conceito da cinemática robótica que calcula um conjunto de ângulos das articulações a partir da pose do TCP (tool center point). O inverso desta função é a *cinemática direta*, que calcula a pose do TCP a partir dos ângulos das articulações do robô. Note que a *cinemática inversa* geralmente tem múltiplas soluções, mas devolve apenas uma delas.

Parâmetros:

- **pose** (*Pose, str, or int*) –
  A pose para a qual devem ser calculados os ângulos das articulações correspondentes. Especifique o valor da seguinte forma:
  - um objeto Pose
  - o nome de uma pose pré-definida (salva no banco de dados)
  - o ID de uma pose pré-definida (salva no banco de dados)
- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.
- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.
- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

A lista dos seis ângulos das articulações em [deg].

Tipo de retorno:

list of float

Exemplos:

Calcular os ângulos das articulações de uma pose previamente guardada na base de dados.

```
inverse_kinematics("grasping_pose")
```

Calcular os ângulos das articulações que atingem um conjunto específico de coordenadas numa configuração específica.

```
inverse_kinematics(create_pose([350.0, 0.0, 550.0, 0.0, -30.0, 0.0], "c1"))
```

inverse_kinematics_nearest(*coordinates*, *previous_angles=None*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Esta função calcula a solução da *cinemática inversa* (ver a função `inverse_kinematics` para mais detalhes) que está mais próxima da pose atual do robô ou mais próxima de uma pose específica.

Parâmetros:

- **coordinates** (*Coordinates, list, or dict*) –
  As coordenadas TCP para as quais devem ser calculados os ângulos das articulações correspondentes. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

- **previous_angles** (*list or None*) –
  Os ângulos para os quais será calculada a solução mais próxima. Especifique o valor da seguinte forma:
  - None, para usar os ângulos atuais do robô
  - Uma lista de ângulos das articulações [deg]

- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.

- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates

- uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
- um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

A lista dos seis ângulos das articulações em [deg].

Tipo de retorno:

list of float

Exemplos:

A cinemática inversa pode ser usada para verificar se um conjunto específico de coordenadas retorna uma solução cinemática.

```
inverse_kinematics_nearest([600, 0, 300, 180, -90, 0])
```

Para obter múltiplas soluções para a mesma pose, a declaração da pose anterior pode ser ajustada. Neste exemplo, são calculadas duas soluções para um robô em pé, ligeiramente deslocado para frente e para baixo.

```
solution_1 = inverse_kinematics_nearest({"x": 485, "y": 188, "z": 1059,
"roll": 44, "pitch": -45, "yaw": -81},
                                        previous_angles=[-160, -20, -20, 20,
-20, -160])
solution_2 = inverse_kinematics_nearest([600, 0, 300, 180, -90, 0],
                                        previous_angles=[20, 15, 30, 30, 15,
10])
```

inverse_kinematics_complete(*coordinates*, *tool_offset=None*, *tool_frame=None*, *work_offset=None*, *work_frame=None*)

Esta função calcula a solução da *cinemática inversa* (ver a função `inverse_kinematics` para mais detalhes) que está mais próxima da pose atual do robô ou mais próxima de uma pose específica.

Parâmetros:

- **coordinates** (*Coordinates, list, or dict*) –
  As coordenadas TCP para as quais devem ser calculados os ângulos das articulações correspondentes. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

- **tool_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar o TCP do robô, fornecida em relação ao quadro da ferramenta. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, não há deslocamento da ferramenta, de modo que o TCP esteja na origem do quadro da ferramenta.

- **tool_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar o TCP do robô, fornecida em relação ao quadro da flange. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de ferramenta global é aplicado. Se o quadro da ferramenta contiver apenas coordenadas zero, ele coincide com o quadro da flange.

- **work_offset** (*Coordinates, list, dict, or None*) –
  A transformação para deslocar a referência do robô, fornecida em relação ao quadro de trabalho. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

Por padrão, não há deslocamento do trabalho, de modo que a referência fique na origem do quadro de trabalho.

- **work_frame** (*Coordinates, list, dict, or None*) –
  A transformação para alterar a referência do robô, dada em relação ao quadro base. Especifique o valor da seguinte forma:
  - um objeto Coordinates
  - uma lista de coordenadas cartesianas no formato [x, y, z, roll, pitch, yaw]
  - um dicionário de coordenadas cartesianas no formato {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}

  Por padrão, o quadro de trabalho global é aplicado. Se o quadro de trabalho contiver apenas zeros, ele coincide com o quadro base.

Retorno:

Listas de seis ângulos das juntas em [graus] e suas configurações no formato {“c1”: [...], “c2”: [...], ...}

Tipo de retorno:

dict

Exemplos:

Neste exemplo, o robô se move para todos os conjuntos de ângulos das juntas que alcançam as mesmas coordenadas TCP.

```
solutions = inverse_kinematics_complete([600, 0, 300, 180, -90, 0])
for joint_angles in solutions.values():
    move_joints([1, 2, 3, 4, 5, 6], joint_angles)
```

Se soubermos um conjunto de ângulos das juntas que atinge a pose desejada, podemos calcular todas as outras soluções que atingem a mesma pose, mas em diferentes configurações das juntas.

```
pose = forward_kinematics([0, 90, -90, 0, 0, 0])
solutions = inverse_kinematics_complete(pose.coordinates)
```

wait(*seconds*)

Aguardando e, portanto, bloqueando ativamente a execução posterior do código por um período definido.

Parâmetros:

seconds (*float*) – A duração para aguardar em [s].

Exemplos:

Este exemplo move o robô para poses predefinidas e espera 3 [s] entre os movimentos.

```
move_to_pose("pose_1")
wait(3)
move_to_pose("pose_2")
```

Este exemplo repete parte do seu código indefinidamente. Para garantir que a velocidade de execução seja limitada, o código espera 100 [ms] no final de cada iteração.

```
while True:
    # enter your code here
    wait(0.1)
```

Este exemplo é semelhante ao exemplo acima, mas em vez de aguardar um tempo constante, o tempo de espera é escolhido dinamicamente para permitir a execução do seu código em um intervalo constante a cada segundo.

```
import time
cycle_time = 1.0
time_start = time.perf_counter()
while True:
    # enter your code here
    time_now = time.perf_counter()
    elapsed_duration = time_now - time_start
    wait(cycle_time - elapsed_duration)
```

run_script(*script_name*, *arguments=None*)

Executando uma aplicação (bloqueante) dentro da aplicação principal.

Parâmetros:

- **script_name** (*str*) – O nome da aplicação a ser executada. Se a aplicação estiver em uma pasta, use o caminho completo da aplicação (ex.: *folder/subfolder/app*).
- **arguments** (*dict*) – Os argumentos passados para a aplicação a ser executada. Você pode lê-los na aplicação chamada usando a variável embutida *script_arguments*.

Exemplos:

Vamos supor que temos uma aplicação separada que mede a temperatura do ambiente. Este exemplo chama esta aplicação repetidamente, uma vez a cada minuto.

```
cycle_time = 60.0
time_measurement = time.perf_counter()
run_script("measurements/measure_temperature")
while True:
    if time.perf_counter() - time_measurement > cycle_time:
        run_script("measurements/measure_temperature")
        time_measurement += cycle_time
    else:
        wait(0.1)
```

start_parallel_script(*script_name*)

Executando uma aplicação (não bloqueante) em paralelo à aplicação principal. Note que a aplicação principal continua imediatamente e todas as aplicações paralelas são interrompidas quando a aplicação principal termina.

Parâmetros:

script_name (*str*) – O nome da aplicação paralela a ser iniciada. Se a aplicação estiver em uma pasta, use o caminho completo da aplicação (ex.: *folder/subfolder/app*).

Exemplos:

Este exemplo inicia uma aplicação paralela com o objetivo de monitorar um sensor de temperatura para garantir que o robô opere apenas dentro de uma faixa de temperatura específica. A aplicação principal é executada continuamente e é abortada se o critério de temperatura operacional não for mais cumprido.

```
start_parallel_script("measurements/measure_temperature_continuously")
set_global_variable("temperature_value") = 20
while -10 < get_global_variable("temperature_value") < 40:
    move_to_pose("pose_1")
    move_to_pose("pose_2")
    move_to_pose("pose_3")
    move_to_pose("pose_4")
print("Aborting the program, because the measured temperature is outside of
the operational temperature range.)
```

stop_application()

Parando as aplicações atualmente em execução. Isso inclui a aplicação principal e quaisquer aplicações iniciadas pela aplicação principal. Observe que uma aplicação em *background* em execução continua rodando.

Levanta:

ScriptInterrupt – Gerado na aplicação principal se esta função for executada a partir de uma aplicação iniciada usando a função *run_script*.

Exemplos:

Este exemplo de código pode ser usado em uma aplicação *background* para pausar temporariamente e retomar manualmente as aplicações principais em execução.

```
pause_application()
answer = dialog_choice("Do you want to continue?", ["Continue", "Abort"],
title="Application Interrupt", dialog_value="Abort")
if answer == "Continue":
    resume_application()
else:
    stop_application()
```

pause_application()

Pausa as aplicações atualmente em execução. Isso inclui a aplicação principal e quaisquer aplicações iniciadas pela aplicação principal. Observe que uma aplicação *background* em execução continua ativa.

Exemplos:

Veja o exemplo combinado de *stop_application*, que usa esta função.

resume_application()

Retomando os aplicativos atualmente em pausa. Como esta função só pode ser executada no estado *paused*, só pode ser usada a partir de uma aplicação *background*.

Exemplos:

Veja o exemplo combinado de *stop_application*, que usa esta função.

stop_and_release()

Para as aplicações atualmente em execução e libera todas as juntas (ativando o modo de compensação da gravidade). O usuário precisa segurar manualmente todas as juntas (desativando o modo de compensação da gravidade) para continuar executando outras aplicações posteriormente.

is_status(*status*)

Verifica se o status do programa corresponde ao status fornecido.

Parâmetros:

status (*str or list of str*) –

O status do programa a verificar pode ser um ou vários dos seguintes:

- "ready" (o robô está ocioso e aguardando comandos)
- "processing" (o robô está processando dados, o que pode demorar)
- "hand_guidance_control" (o robô está em modo de controlo por orientação manual)
- "running" (o robô está a executar uma aplicação ou um movimento)
- "paused" (o robô está a pausar uma aplicação ou um movimento)
- "emergency_stop" (uma parada de emergência foi acionada)
- "normal_stop" (uma parada normal foi acionada)
- "protective_stop" (uma parada de proteção foi acionada)
- "error" (ocorreu um erro ou uma violação de segurança)
- "position_limit" (ocorreu uma violação do limite de posição segura)
- "safeguard" (uma interrupção de segurança foi acionada)
- "collision" (uma interrupção de colisão foi acionada)
- "proceed" (o robô está a continuar após uma interrupção)
- "stopping" (o robô está a parar o seu programa)

Retorno:

Se o status do programa corresponde ao status fornecido (ou a um dos status fornecidos)

Tipo de retorno:

bool

dialog_choice(*text*, *options*, *title=''*, *dialog_type=0*, *dialog_value=None*, *mandatory=False*)

Criação de um diálogo com botões. Este diálogo bloqueia o fluxo do programa até que o utilizador responda com uma das opções apresentadas.

Parâmetros:

- **text** (*str*) – A mensagem (ou pergunta) apresentada ao utilizador.
- **options** (*list of str*) – As opções que podem ser escolhidas, que são os rótulos de cada botão.
- **title** (*str*) – O título da janela de diálogo
- **dialog_type** (*int*) –
  O tipo de diálogo a apresentar ao usuário. Pode ser um dos seguintes:
  - 0: informação / azul
  - 1: sucesso / verde
  - 2: aviso / laranja
  - 3: erro / vermelho
- **dialog_value** (*str*) – O valor padrão a retornar se o diálogo não for respondido.
- **mandatory** (*bool*) – Se o diálogo deve ser respondido para que o fluxo do programa continue. Se definido como *True*, o diálogo não pode ser ignorado, por exemplo, fechando a janela pop-up, e bloqueará até que uma resposta seja recebida.

Retorno:

A opção selecionada, que é o rótulo do botão pressionado pelo usuário.

Tipo de retorno:

str

Exemplos:

Este exemplo mostra como criar uma janela pop-up informativa simples contendo um botão «Continuar». Ao pressionar o botão, o programa continuará executando seu código.

```
dialog_choice("This is just an informative message.", ["Continue"],
title="Information", dialog_type=0)
```

dialog_dropdown(*text*, *options*, *title*='', *dialog_type*=*0*, *dialog_value*=*None*, *mandatory*=*False*)

Criando um diálogo com opções apresentadas em uma lista suspensa. O diálogo bloqueia o fluxo do programa até que o usuário selecione e confirme uma das opções.

Parâmetros:

- **text** (*str*) – A mensagem (ou pergunta) apresentada ao utilizador.
- **options** (*list of str*) – As opções que podem ser escolhidas, que são os rótulos das opções da lista suspensa.
- **title** (*str*) – O título da janela de diálogo
- **dialog_type** (*int*) –
  O tipo de diálogo a apresentar ao usuário. Pode ser um dos seguintes:
  - 0: informação / azul
  - 1: sucesso / verde
  - 2: aviso / laranja
  - 3: erro / vermelho
- **dialog_value** (*str*) – O valor padrão a retornar se o diálogo não for respondido.
- **mandatory** (*bool*) – Se o diálogo deve ser respondido para que o fluxo do programa continue. Se definido como *True*, o diálogo não pode ser ignorado, por exemplo, fechando a janela pop-up, e bloqueará até que uma resposta seja recebida.

Retorno:

A opção selecionada, que é o rótulo da opção selecionada na lista suspensa.

Tipo de retorno:

str

Exemplos:

Este exemplo mostra como criar uma janela pop-up de aviso e permite que o usuário escolha entre várias opções para influenciar o fluxo do programa.

```
answer = dialog_dropdown("High temperature detected. You may want to reduce
the robot's speed.",
                         options=["Reduce by 20%", "Reduce by 10%", "Do not
reduce"],
                         title="High Temperature Warning", dialog_type=2,
dialog_value="Do not reduce")
if answer == "Reduce by 20%":
    velocity *= 0.8
elif answer == "Reduce by 10%":
    velocity *= 0.9
```

dialog_text(*text*, *title*='', *dialog_type=0*, *dialog_value=None*, *mandatory=False*)

Criando um diálogo com um campo de texto. O diálogo bloqueia o fluxo do programa até que o usuário insira um texto e confirme.

Parâmetros:

- **text** (*str*) – A mensagem (ou pergunta) apresentada ao utilizador.
- **title** (*str*) – O título da janela de diálogo
- **dialog_type** (*int*) –
  O tipo de diálogo a apresentar ao usuário. Pode ser um dos seguintes:
  - 0: informação / azul
  - 1: sucesso / verde
  - 2: aviso / laranja
  - 3: erro / vermelho
- **dialog_value** (*str*) – O valor padrão a retornar se o diálogo não for respondido.
- **mandatory** (*bool*) – Se o diálogo deve ser respondido para que o fluxo do programa continue. Se definido como *True*, o diálogo não pode ser ignorado, por exemplo, fechando a janela pop-up, e bloqueará até que uma resposta seja recebida.

Retorno:

O texto inserido no campo de texto.

Tipo de retorno:

str

Exemplos:

Este exemplo mostra como criar uma janela pop-up simples com um campo de texto para inserir seu nome. Após o usuário inserir seu nome, ele pode ser usado no programa, por exemplo, para registro.

```
name = dialog_text("Please enter your name:", title="Welcome!")
log_message("Registered user: {name}")
```

dialog_yes_no(*text*, *title*='', *dialog_type*=*0*, *dialog_value*=*None*, *mandatory*=*False*)

Criando um diálogo com botões *Sim* e *Não*. O diálogo bloqueia o fluxo do programa até que o usuário pressione um desses botões.

Parâmetros:

- **text** (*str*) – A mensagem (ou pergunta) apresentada ao utilizador.
- **title** (*str*) – O título da janela de diálogo
- **dialog_type** (*int*) –
  O tipo de diálogo a apresentar ao usuário. Pode ser um dos seguintes:
  - 0: informação / azul
  - 1: sucesso / verde
  - 2: aviso / laranja
  - 3: erro / vermelho
- **dialog_value** (*str*) – O valor padrão a retornar se o diálogo não for respondido.
- **mandatory** (*bool*) – Se o diálogo deve ser respondido para que o fluxo do programa continue. Se definido como *True*, o diálogo não pode ser ignorado, por exemplo, fechando a janela pop-up, e bloqueará até que uma resposta seja recebida.

Retorno:

Verdadeiro se o usuário selecionar «sim», Falso se o usuário selecionar «não».

Tipo de retorno:

bool

Exemplos:

Este exemplo usa um diálogo de confirmação simples para permitir que o usuário escolha se deseja continuar ou abortar a aplicação.

```
if not dialog_yes_no("An error occurred. Do you want to continue anyway?",
title="Error", dialog_type=3):
    stop_application()
```

print(**args*)

Imprimindo uma mensagem na saída da aplicação. Observe que esta função é semelhante à função *print* embutida do Python. Suporta múltiplos argumentos e vários tipos de argumentos. Os argumentos são automaticamente convertidos em strings e concatenados com um separador de espaço.

Parâmetros:

args (*tuple of various*) – Os argumentos a serem impressos.

Exemplos:

Imprimindo o valor de uma variável em um estilo bem formatado. Observe que a primeira chamada usa conversão e concatenação automáticas, enquanto a segunda usa f-strings com o mesmo resultado.

```
print("Variable value:", var)
print(f"Variable value: {var}")
```

show_notification(*message_text*, *message_title*='*Notification*')

Exibir uma mensagem de notificação personalizada na interface. Isso não afeta o fluxo do programa. O usuário pode especificar no menu de configurações se as notificações desaparecem ou permanecen.

Parâmetros:

- **message_text** (*str*) – O texto de notificação a ser exibido na interface.
- **message_title** (*str*) – O título do pop-up de notificação.

raise_error(*message*)

Gera uma mensagem de erro personalizada na aplicação em execução. O erro é do tipo `ApplicationError` e pode ser capturado na aplicação. Caso não seja capturado, a aplicação em execução é interrompida. O erro também é registado no ficheiro `error.log`, que faz parte do sistema de registo. Os registos podem ser descarregados através da interface premindo `CTRL + SHIFT + L`.

Parâmetros:

message (*str*) – A mensagem de erro a gerar na aplicação.

Levanta:

ApplicationError – Contém a mensagem personalizada.

Exemplos:

Uma aplicação em segundo plano monitoriza entradas digitais e verifica se uma entrada de erro específica é acionada. Se uma entrada de erro for acionada, este erro é gerado na aplicação.

```
# keeping track of the previous error flag value
error_status = False
# do forever
while True:
    # read the value of the error flag (True or False)
    error_flag = read_digital_main_input("error_flag")
    # if the error flag switches from False to True, raise an application
error
    if error_flag:
        if not error_status:
            raise_error("The error input was raised!")
        error_status = True
    else:
        error_status = False
    wait(0.1)
```

Este erro da aplicação pode então ser capturado e tratado pela aplicação principal que controla o robô.

```
try:
    my_main_program()
except ApplicationError as error:
    ...
```

raise_warning(*message*)

Apresenta uma caixa de mensagem de aviso personalizada na interface. Isto não tem efeito no fluxo do programa. Se os avisos desaparecem ou permanecem na interface pode ser definido pelo utilizador no menu de definições.

Parâmetros:

message (*str*) – A mensagem de aviso a apresentar na interface.

Exemplos:

Suponhamos que existe um sensor de temperatura ligado a uma entrada analógica. Este exemplo apresenta um aviso na interface se o valor ultrapassar um determinado limiar e o remove quando o valor volta a descer abaixo desse limiar.

```
# flag to store whether a warning was raised
high_temperature = False
# do forever
while True:
    # check temperature value
    temperature_value = read_analog_main_input("temperature_sensor")
    # if value is too high, raise a warning
    if temperature_value > 5.2:
        if not high_temperature:
            raise_warning("High temperature detected! Slow down the robot to
prevent overheating.")
        high_temperature = True
    # if value recovers, reset the flag, such that a warning can be raised
again later on
    elif temperature_value < 5.0:
        high_temperature = False
    wait(5)
```

log_message(*message*, *logger='custom'*, *include_timestamp=True*)

Regista uma mensagem num ficheiro de registo personalizado.

Parâmetros:

- **message** (*str*) – A mensagem personalizada a registar.
- **logger** (*str*) – O nome do ficheiro de registo onde gravar. O ficheiro de registo encontra-se em `custom/<logger>.log` na pasta de registos. O nome de ficheiro predefinido é `custom/custom.log`. Os registos podem ser descarregados através da interface premindo `CTRL + SHIFT + L`.
- **include_timestamp** (*bool*) – Indica se um carimbo temporal é adicionado à mensagem registada.

Exemplos:

Regista o valor das medições de um sensor no ficheiro *custom/measurements.log*.

```
value = read_analog_main_input("my_sensor")
log_message(f"Sensor value: {value}", logger="measurements")
```

clear_log()

Limpa e remove todos os ficheiros de registo personalizados.

set_shared_variable(*name*, *value*)

Define o valor de uma variável partilhada.

Note que as *variáveis partilhadas* só existem enquanto a aplicação principal estiver em execução. Após a terminação da aplicação principal, todas as *variáveis partilhadas* são automaticamente removidas. Se for necessário que as variáveis mantenham os seus valores ao longo do tempo de execução da aplicação ou mesmo após reiniciar o robô, utilize *variáveis globais*.

Parâmetros:

- **name** (*str*) – O nome da variável a definir.
- **value** (*various*) – O novo valor desta variável.

Exemplos:

Este exemplo mostra como uma aplicação paralela pode ser terminada pela aplicação principal utilizando variáveis partilhadas. A aplicação paralela apenas escuta um sinalizador de terminação e termina assim que o sinalizador é definido como *True*. Adicionalmente, a variável partilhada é removida quando deixa de ter efeito. Como as variáveis partilhadas são eliminadas quando a aplicação principal termina, isto não é estritamente necessário, mas é uma boa prática.

```
while not get_shared_variable("terminate_parallel_application"):
    ...
    wait(0.1)
remove_shared_variable("terminate_parallel_application")
```

A aplicação principal assegura que o sinalizador de terminação existe, executa a aplicação paralela e, por fim, define o sinalizador de terminação como *True* para finalizar a aplicação paralela.

```
set_shared_variable("terminate_parallel_application", False)
start_parallel_script("my_parallel_application")
...
set_shared_variable("terminate_parallel_application", True)
...
```

get_shared_variable(*name=None*)

Lê o valor de uma variável partilhada específica. Por omissão, esta função lê os valores de todas as variáveis partilhadas.

Note que as *variáveis partilhadas* só existem enquanto a aplicação principal estiver em execução. Após a terminação da aplicação principal, todas as *variáveis partilhadas* são automaticamente removidas. Se for necessário que as variáveis mantenham os seus valores ao longo do tempo de execução da aplicação ou mesmo após reiniciar o robô, utilize *variáveis globais*.

Parâmetros:

name (*str (or None)*) – Nome da variável a devolver (ou None se todas as variáveis forem devolvidas).

Retorno:

Valor da variável a devolver (ou um dicionário que contém todas as variáveis).

Tipo de retorno:

various (or dict)

Exemplos:

Consulte os exemplos de *set_shared_variable*, que combinam a utilização de todas as funções de *variáveis partilhadas*.

remove_shared_variable(*name*)

Remove uma variável partilhada. Após a sua remoção, o valor deixa de poder ser acedido.

Note que as *variáveis partilhadas* só existem enquanto a aplicação principal estiver em execução. Após a terminação da aplicação principal, todas as *variáveis partilhadas* são automaticamente removidas. Se for necessário que as variáveis mantenham os seus valores ao longo do tempo de execução da aplicação ou mesmo após reiniciar o robô, utilize *variáveis globais*.

Parâmetros:

name (*str*) – nome da variável a remover

Exemplos:

Consulte os exemplos de *set_shared_variable*, que combinam a utilização de todas as funções de *variáveis partilhadas*.

set_global_variable(*name*, *value*)

Definir o valor de uma variável global.

Note que as alterações em *variáveis globais* são permanentes e permanecem mesmo após desligar e reiniciar o robô. Portanto, é altamente recomendado remover *variáveis globais* assim que deixarem de servir a qualquer propósito. Se precisar de variáveis apenas durante a execução de uma única aplicação, use *variáveis partilhadas*.

Parâmetros:

- **name** (*str*) – O nome da variável a definir.
- **value** (*various*) – O novo valor desta variável.

Exemplos:

Este exemplo mostra como uma aplicação repetitiva conta o número de iterações bem-sucedidas. Isso é feito inicializando um contador global na primeira execução e incrementando esse contador no final de cada iteração do ciclo.

```
if "counter" not in get_global_variable():
    set_global_variable("counter", 0)
while True:
    ...
    counter = get_global_variable("counter")
    set_global_variable("counter", counter + 1)
```

get_global_variable(*name*=*None*)

Ler o valor de uma variável global específica. Por padrão, esta função lê os valores de todas as variáveis globais.

Note que as alterações em *variáveis globais* são permanentes e permanecem mesmo após desligar e reiniciar o robô. Portanto, é altamente recomendado remover *variáveis globais* assim que deixarem de servir a qualquer propósito. Se precisar de variáveis apenas durante a execução de uma única aplicação, use *variáveis partilhadas*.

Parâmetros:

name (*str (or None)*) – Nome da variável a devolver (ou None se todas as variáveis forem devolvidas).

Retorno:

Valor da variável a devolver (ou um dicionário que contém todas as variáveis).

Tipo de retorno:

various (or dict)

Exemplos:

Consulte os exemplos de *set_global_variable*, que combinam o uso das funções de *variáveis globais*.

remove_global_variable(*name*)

Remover uma variável global. Após a remoção, o seu valor já não pode ser acedido.

Note que as alterações em *variáveis globais* são permanentes e permanecem mesmo após desligar e reiniciar o robô. Portanto, é altamente recomendado remover *variáveis globais* assim que deixarem de servir a qualquer propósito. Se precisar de variáveis apenas durante a execução de uma única aplicação, use *variáveis partilhadas*.

Parâmetros:

name (*str*) – nome da variável a remover

Exemplos:

Este exemplo remove todas as variáveis globais.

```
for name in get_global_variable().keys():
    remove_global_variable(name)
```

set_state_variable(*name*, *value*)

Definir o valor de uma variável de estado.

Note que as *variáveis de estado* são definidas individualmente para cada aplicação e são usadas apenas em aplicações principais. Podem ser configuradas e personalizadas no separador *variables* do editor da aplicação. Cada *variável de estado* tem um tipo específico e só aceita valores desse tipo. O principal objetivo das *variáveis de estado* é mostrar o seu valor em tempo real na *Panel Interface* para observar e controlar a aplicação em execução.

Parâmetros:

- **name** (*str*) – O nome da variável a definir.
- **value** (*various*) – O novo valor desta variável.

Exemplos:

Suponhamos que para esta aplicação foi definida uma variável de estado apenas de leitura do tipo *percent* com o nome *progress* e uma variável de estado dinamicamente editável do tipo *boolean* com o nome *logging*. O objetivo da variável *progress* é armazenar o progresso atual do programa em percentagem. O objetivo do sinalizador *logging* é definir se a atualização do progresso deve ser registada. O utilizador pode alterar o valor de *logging* a qualquer momento, adaptando dinamicamente o programa.

```
number_of_iterations = 128
for i in range(number_of_iterations):
    progress = i / number_of_iterations * 100
    set_state_variable("progress", progress)
    if get_state_variable("logging"):
        log_message(f"The current progress is {progress}%.")
    ...
set_state_variable("progress", 100)
    if get_state_variable("logging"):
        log_message(f"The current progress is 100%. The program successfully
completed.")
```

get_state_variable(*name*=*None*)

Ler o valor de uma variável de estado específica. Por padrão, esta função lê os valores de todas as variáveis de estado.

Note que as *variáveis de estado* são definidas individualmente para cada aplicação e são usadas apenas em aplicações principais. Podem ser configuradas e personalizadas no separador *variables* do editor da aplicação. Cada *variável de estado* tem um tipo específico e só aceita valores desse tipo. O principal objetivo das *variáveis de estado* é mostrar o seu valor em tempo real na *Panel Interface* para observar e controlar a aplicação em execução.

Parâmetros:

name (*str (or None)*) – Nome da variável a devolver (ou None se todas as variáveis forem devolvidas).

Retorno:

Valor da variável a devolver (ou um dicionário que contém todas as variáveis).

Tipo de retorno:

various (or dict)

Exemplos:

Consulte os exemplos de *set_state_variable*, que combinam o uso das funções de *variáveis de estado*.

read_actuator_position(*actuator_ids*=*None*)

Ler a posição angular de um, vários ou de todos os atuadores.

Parâmetros:

actuator_ids (*None, str, int, list, set*) –

Os IDs dos atuadores a ler. Especifique o valor da seguinte forma:

- o ID de uma única articulação (ex. *6*)
- uma lista de IDs para várias articulações (ex. *[1, 4, 6]*)
- um conjunto de nomes de juntas para várias juntas (ex.: *{1, 4, 6}*)
- a constante *ALL_KIN_JOINTS* ou *None* para ler de todas as juntas

Retorno:

A posição angular dos atuadores solicitados (em [deg]), que pode ser:

- um único valor de posição de junta -> float
- uma lista de valores de posição de juntas -> lista de float
- um mapeamento entre IDs de juntas e os respetivos valores de posição -> dict de float

Tipo de retorno:

float, or list of float, or dict of float

Exemplos:

Imprimir as posições atuais dos atuadores das juntas do punho.

```
print(read_actuator_position([1, 4, 6]))
```

read_actuator_velocity(*actuator_ids=None*)

Lendo a velocidade angular de um, vários ou todos os atuadores.

Parâmetros:

actuator_ids (*None, str, int, list, set*) –

Os IDs dos atuadores a ler. Especifique o valor da seguinte forma:

- o ID de uma única articulação (ex. *6*)
- uma lista de IDs para várias articulações (ex. *[1, 4, 6]*)
- um conjunto de nomes de juntas para várias juntas (ex.: *{1, 4, 6}*)
- a constante *ALL_KIN_JOINTS* ou *None* para ler de todas as juntas

Retorno:

A velocidade angular dos atuadores solicitados (em °/s), que pode ser:

- um único valor de velocidade de junta -> float
- uma lista de valores de velocidade de junta -> lista de floats
- um mapeamento entre IDs de juntas e seus valores de velocidade -> dict de floats

Tipo de retorno:

float, or list of float, or dict of float

Exemplos:

Monitorar a velocidade durante um movimento para determinar o atuador que se move mais rapidamente.

```
move_to_pose("target", block=False)
max_velocities = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
while is_motor_moving():
    velocities = read_actuator_velocities({1, 2, 3, 4, 5, 6})
    for joint in range(1, 7):
        max_velocities[joint] = max([max_velocities[joint],
abs(velocities[joint])])
fastest_joint = max(max_velocities, key=max_velocities.get)
print(f"The fastest joint is {fastest_joint} with a maximum velocity of
{max_velocities[fastest_joint]}°/s")
```

get_linear_velocity()

Obtendo a velocidade linear do TCP (ponto central da ferramenta) do robô.

Retorno:

A velocidade linear da ferramenta em [mm/s].

Tipo de retorno:

list

Exemplos:

Obtém e imprime a velocidade linear atual.

```
print(get_linear_velocity())
```

get_angular_velocity()

Obtendo a velocidade angular do TCP (ponto central da ferramenta) do robô.

Retorno:

A velocidade angular da ferramenta em [deg/s].

Tipo de retorno:

list

Exemplos:

Obtém e imprime a velocidade angular atual.

```
print(get_angular_velocity())
```

read_actuator_current(*actuator_ids=None*)

Lendo a corrente aplicada em um, vários ou todos os atuadores.

Parâmetros:

actuator_ids (*None, str, int, list, set*) –

Os IDs dos atuadores a ler. Especifique o valor da seguinte forma:

- o ID de uma única articulação (ex. *6*)
- uma lista de IDs para várias articulações (ex. *[1, 4, 6]*)
- um conjunto de nomes de juntas para várias juntas (ex.: *{1, 4, 6}*)
- a constante *ALL_KIN_JOINTS* ou *None* para ler de todas as juntas

Retorno:

Os valores de corrente dos atuadores solicitados (em A), que podem ser:

- um único valor de corrente de junta -> float
- uma lista de valores de corrente de junta -> lista de floats
- um mapeamento entre IDs de juntas e seus valores de corrente -> dict de floats

Tipo de retorno:

float, or list of float, or dict of float

Exemplos:

Imprimindo as correntes dos atuadores do robô em uma pose específica duas vezes, uma antes e outra depois de pegar um objeto.

```
move_to_pose("measure_pose")
print(f"Currents without payload: {read_actuator_current()}")
move_to_pose("pick_pose")
tool_pick()
move_to_pose("measure_pose")
print(f"Currents with payload: {read_actuator_current()}")
```

tool_pick()

Ativa a função de pegar da ferramenta para agarrar um objeto. Isso pode resultar em fixação para ferramentas mecânicas ou na aplicação de sucção para ferramentas pneumáticas.

Exemplos:

Ativa a função de pegar da ferramenta.

```
tool_pick()
```

tool_place()

Ativa a função de posicionar da ferramenta para liberar um objeto. Isso pode resultar na abertura de uma pinça para ferramentas mecânicas ou na interrupção da sucção para ferramentas pneumáticas.

Exemplos:

Ativa a função de posicionar da ferramenta.

```
tool_place()
```

set_tool_payload(*payload*, *center_of_mass=None*, *inertia=None*)

Define dinamicamente a carga da ferramenta durante a operação do robô.

Parâmetros:

- **payload** (*float*) – a carga da ferramenta (peso + itens agarrados) em [kg]
- **center_of_mass** (*list of float*) – centro de massa [x, y, z] relativo ao quadro da flange em [m]
- **inertia** (*list of float*) – matriz de inércia dividida em lista com comprimento 9 [kg m^2]

Exemplos:

Alterando a carga útil enquanto se pega e coloca objetos de uma posição para outra.

```
tool_mass = 1.3
object_mass = 0.7

while True:
    move_pose("pick_pose")
    tool_pick()
    set_tool_payload(tool_mass + object_mass)

    move_pose("place_pose")
    tool_place()
    set_tool_payload(tool_mass)
```

read_digital_main_input(*identifiers*)

Lendo o valor de uma, várias ou todas as I/Os da categoria “Digital Main Inputs”.

Os dois estados das I/Os digitais são:

- *HIGH*, representado pelo valor booleano *True*
- *LOW*, representado pelo valor booleano *False*

Parâmetros:

identifiers (*int, str, list (of int or str), set (of int or str)*) –

Especifica quais E/S ler. Os identificadores são os números de E/S ou os nomes personalizados exclusivos que podem ser atribuídos a essas E/S. Isto pode ser um dos seguintes:

- um único número (int) ou nome personalizado (str) -> retorna um bool
- uma lista de números (int) ou nomes personalizados (str), ou misto -> retorna uma lista de bools
- um conjunto de números (int) ou nomes personalizados (str), ou misto -> retorna um dicionário com valores booleanos e identificadores como chaves

Retorno:

Os valores booleanos das I/Os digitais solicitadas (HIGH = True, LOW = False)

Tipo de retorno:

bool, list, dict

Exemplos:

Vamos supor que o nome “trigger” foi atribuído a uma das entradas digitais. Este bloco de código aguarda até que este trigger mude para HIGH antes de continuar a execução.

```
while not read_digital_main_input("trigger"):
    wait(0.1)
```

Lendo as primeiras 8 I/Os digitais e convertendo a lista binária de bools em um número inteiro entre 0 e 255.

```
bool_list = read_digital_main_input(list(range(1, 9)))
int_representation = int(''.join([str(int(bit)) for bit in bool_list]), 2)
```

read_digital_tool_input(*identifiers*)

Lendo o valor de uma, várias ou todas as I/Os da categoria “Digital Tool Inputs”.

Para mais detalhes, consulte “read_digital_main_input”, que tem um uso semelhante.

read_analog_main_input(*identifiers*)

Lendo o valor de uma, várias ou todas as I/Os da categoria “Analog Main Inputs”.

As E/S analógicas podem operar em modo de tensão ou de corrente. Os valores são aplicados e lidos em incrementos de 0.1, o que significa que o valor de E/S 55 representa o valor real de tensão de 5.5 V ou o valor de corrente de 5.5 mA. O intervalo de valores das E/S analógicas é:

- Entre 0.0 e 10.0 V (em modo de tensão)
- Entre 4.0 e 20.0 mA (em modo de corrente)

Parâmetros:

identifiers (*int, str, list (of int or str), set (of int or str)*) –

Especifica quais E/S ler. Os identificadores são os números de E/S ou os nomes personalizados exclusivos que podem ser atribuídos a essas E/S. Isto pode ser um dos seguintes:

- um único número (int) ou nome personalizado (str) -> retorna um bool
- uma lista de números (int) ou nomes personalizados (str), ou misto -> retorna uma lista de bools
- um conjunto de números (int) ou nomes personalizados (str), ou misto -> retorna um dicionário com valores booleanos e identificadores como chaves

Retorno:

Os valores numéricos das E/S analógicas solicitadas

Tipo de retorno:

float, list, dict

Exemplos:

Suponha que o nome “temperature” tenha sido atribuído a uma das entradas analógicas em modo de corrente. Um sensor de temperatura foi conectado a essa entrada, e foi determinado que um valor de retorno de 10.6 mA representa um limite crítico de temperatura que não deve ser excedido. Em seguida, execute o programa abaixo em paralelo com a sua aplicação principal, o qual interrompe o programa se o limite crítico for atingido.

```
threshold = 10.6
while True:
    value = read_analog_main_input("temperature")
    if value >= threshold:
        raise_error("High temperature detected -> aborting the program")
    wait(0.1)
```

read_analog_tool_io(*identifiers*)

Lendo o valor de uma, várias ou todas as I/Os da categoria “Analog Tool I/O”.

Para mais detalhes, consulte “read_analog_main_input”, que tem um uso semelhante.

write_digital_main_output(*identifiers*, *values*)

Escrevendo o valor de uma, várias ou todas as I/Os da categoria "Digital Main Output".

Os dois estados das I/Os digitais são:

- *HIGH*, representado pelo valor booleano *True*
- *LOW*, representado pelo valor booleano *False*

Parâmetros:

- **identifiers** (*int, str, list (of int or str), set (of int or str)*) –
  Especifica quais E/S escrever. Os identificadores são os números de E/S ou os nomes personalizados exclusivos que podem ser atribuídos a essas E/S. Isto pode ser um dos seguintes:
  - um único número (int) ou nome personalizado (str) -> *values* é um bool
  - uma lista de números (int) ou nomes personalizados (str), ou misto -> *values* é uma lista de bools
  - um conjunto de números (int) ou nomes personalizados (str), ou misto -> *values* é um dicionário com valores booleanos e identificadores como chaves
- **values** (*bool, list (of bool), dict (of bool)*) –
  Os valores booleanos das I/Os digitais a escrever (HIGH = True, LOW = False). Dependendo dos *identifiers*, isto pode ser um dos seguintes:
  - um único bool -> se *identifiers* for uma única I/O
  - uma lista de booleanos -> se *identifiers* for uma lista de I/Os
  - um dicionário com valores booleanos e identificadores como chaves -> se *identifiers* for um conjunto de I/Os

Exemplos:

Suponha que o nome "trigger" foi atribuído a uma das saídas digitais. Este código envia um impulso para essa saída digital definindo-a como HIGH por 100 ms e depois retornando a LOW.

```
write_digital_main_output("trigger", True)
wait(0.1)
write_digital_main_output("trigger", False)
```

Escrevendo um número inteiro de 8 bits (entre 0 e 255) nos primeiros 8 I/Os digitais, convertendo primeiro o número em uma lista de booleanos e depois escrevendo todos os 8 valores simultaneamente.

```
number = 123
bool_list = [bool(int(bit)) for bit in f"{number:08b}"]
write_digital_main_output(list(range(1, 9)), bool_list)
```

write_digital_tool_output(*identifiers*, *values*)

Escrevendo o valor de uma, várias ou todas as I/Os da categoria "Digital Tool Output".

Para mais detalhes, consulte "write_digital_main_output", que tem um uso semelhante.

write_analog_main_output(*identifiers*, *values*)

Escrevendo o valor de uma, várias ou todas as I/Os da categoria “Analog Main Outputs”.

As E/S analógicas podem operar em modo de tensão ou de corrente. Os valores são aplicados e lidos em incrementos de 0.1, o que significa que o valor de E/S 55 representa o valor real de tensão de 5.5 V ou o valor de corrente de 5.5 mA. O intervalo de valores das E/S analógicas é:

- Entre 0.0 e 10.0 V (em modo de tensão)
- Entre 4.0 e 20.0 mA (em modo de corrente)

Parâmetros:

- **identifiers** (*int, str, list (of int or str), set (of int or str)*) –
  Especifica quais E/S escrever. Os identificadores são os números de E/S ou os nomes personalizados exclusivos que podem ser atribuídos a essas E/S. Isto pode ser um dos seguintes:
  - um único número (int) ou nome personalizado (str) -> *values* é um float
  - uma lista de números (int) ou nomes personalizados (str), ou misto -> *values* é uma lista de float
  - um conjunto de números (int) ou nomes personalizados (str), ou misto -> *values* é um dicionário com valores float e identificadores como chaves
- **values** (*float, list (of float), dict (of float)*) –
  Os valores numéricos das E/S analógicas a escrever. Dependendo dos *identifiers*, isto pode ser um dos seguintes:
  - um float único -> se *identifiers* for um único I/O
  - uma lista de float -> se *identifiers* for uma lista de I/O
  - um dict com valores float e os identificadores como chaves -> se *identifiers* for um conjunto de I/O

Exemplos:

Vamos assumir que os valores RGB de um LED controlável estão diretamente ligados a três E/S analógicas com os nomes personalizados “red”, “green” e “blue”, de forma que o intervalo máximo de valores [0, 255] é mapeado diretamente para o intervalo máximo de tensão [0.0 V, 10.0 V]. Este exemplo escreve esses valores simultaneamente.

```
if color == "orange":
    color_code = [255, 188, 32]
elif color == "green":
    color_code = [31, 226, 0]
elif color == "purple":
    color_code = [170, 22, 218]
else:
    raise_error("Unknown color")
write_analog_main_output(["red", "green", "blue"], [c/2.55 for c in
color_code])
```

write_analog_tool_io(*identifiers*, *values*)

Escrevendo o valor de uma, várias ou todas as I/Os da categoria "Analog Tool I/O".

Para mais detalhes, consulte "write_analog_main_output", que tem um uso semelhante.

wait_for_digital_main_input(*identifier*, *value*)

Aguardando até que um I/O da categoria “Digital Main Inputs” leia um valor específico.

Os dois estados das I/Os digitais são:

- *HIGH*, representado pelo valor booleano *True*
- *LOW*, representado pelo valor booleano *False*

Parâmetros:

- **identifier** (*int, str*) – Especifica qual I/O esperar. O identificador é o número do I/O (int) ou o nome personalizado único (str) que pode ser atribuído a este I/O.
- **value** (*bool*) – O valor a ser aguardado.

Exemplos:

Vamos supor que o nome “trigger” foi atribuído a uma das entradas digitais. Este bloco de código aguarda até que este trigger mude para HIGH antes de continuar a execução.

```
wait_for_digital_main_input("trigger", True)
```

Em casos raros, você pode querer pausar um programa e ainda assim quer que a função de espera termine se o I/O for acionado. Esta função de espera não pode ser usada para esse caso, mas o exemplo a seguir faz exatamente isso:

Adicione este código em um aplicativo em segundo plano. Ele monitora o valor de um I/O específico e define uma flag caso o valor especificado seja atingido durante a monitoração.

```
# writing the 'trigger' value into a global variable for further processing
set_global_variable("triggered_flag", False)
while True:
    if not get_global_variable("triggered_flag"):
        set_global_variable("triggered_flag",
read_digital_main_input("trigger"))
    wait(0.01)
```

E adicione este código à contraparte do aplicativo principal que aciona o monitor em segundo plano e aguarda até que a flag seja definida pelo monitor.

```
# waiting for the 'trigger' value to be written into a global variable
set_global_variable ("triggered_flag", False)
while not get_global_variable("triggered_flag"):
    wait(0.01)
```

wait_for_digital_tool_input(*identifier*, *value*)

Aguardando até que um I/O da categoria “Digital Tool Inputs” leia um valor específico.

Para mais detalhes, consulte “wait_for_digital_main_input”, que tem um uso semelhante.

wait_for_analog_main_input(*identifier*, *value_range*)

Aguardando até que um I/O da categoria “Analog Main Inputs” leia um valor específico.

As E/S analógicas podem operar em modo de tensão ou de corrente. Os valores são aplicados e lidos em incrementos de 0.1, o que significa que o valor de E/S 55 representa o valor real de tensão de 5.5 V ou o valor de corrente de 5.5 mA. O intervalo de valores das E/S analógicas é:

- Entre 0.0 e 10.0 V (em modo de tensão)
- Entre 4.0 e 20.0 mA (em modo de corrente)

Parâmetros:

- **identifier** (*int, str*) – Especifica qual I/O esperar. O identificador é o número do I/O (int) ou o nome personalizado único (str) que pode ser atribuído a este I/O.
- **value_range** (*list of float*) –
  Aguardando que esta E/S atinja um valor dentro deste intervalo. Este é um dos seguintes:
  - tanto o limite inferior como o superior (por ex., [5.5, 7.0])
  - apenas um limite inferior (por ex., [5.5, None])
  - apenas um limite superior (por ex., [None, 7.0])

Exemplos:

Vamos considerar um I/O analógico em modo de tensão. Este exemplo aguarda que seu valor de tensão ultrapasse 8.0 V, o que é representado pelo valor 80 (em passos de 0.1 V).

```
wait_for_analog_main_input(6, [80, None])
```

wait_for_analog_tool_io(*identifier*, *value_range*)

Aguardando até que um I/O da categoria “Analog Tool I/O” leia um valor específico.

Para mais detalhes, consulte “wait_for_analog_main_input”, que tem um uso semelhante.

get_runtime_position_offsets()

Esta função retorna os deslocamentos de posição do robô.

Retorno:

deslocamentos de posição do robô

Tipo de retorno:

list of float

read_button_state(*channel=None*)

Lendo o estado atual (pressionado ou não) de um ou todos os botões personalizáveis da interface do robô. Observe que os botões com funcionalidades específicas atribuídas não podem ser lidos em tempo de execução, pois estão sendo utilizados.

Estes botões estão localizados na junta mais alta do braço do robô.

Parâmetros:

channel (*int or None*) – O identificador do botão, que é um número entre 1 e o número de botões. Por padrão, os estados de todos os botões personalizáveis são lidos.

Retorno:

O estado atual do botão solicitado como um booleano, ou o estado atual de cada botão, empacotado em um dicionário no formato {1: bool, 2: bool, ...}.

Tipo de retorno:

bool or dict of bool

Exemplos:

Este exemplo imprime sempre que o estado de um botão personalizável muda.

```
button_states = read_button_state()
while True:
    for channel, value in read_button_state().items():
        if not button_states[channel] and value:
            print(f"Button {channel} was pressed.")
        elif button_states[channel] and not value:
            print(f"Button {channel} was released.")
        button_states[channel] = value
    wait(0.1)
```

Execute este exemplo como um aplicativo em segundo plano. Ele permite pausar e retomar o aplicativo principal em execução usando os botões 1 e 2.

```
pausing = False
while True:
    if read_button_state(1) and not pausing:
        pause_application()
        pausing = True
    elif read_button_state(2) and pausing:
        resume_application()
        pausing = False
    wait(0.1)
```

create_tcp_client(*ip*, *port*)

Criando um cliente de socket TCP para transmitir mensagens. O cliente tenta continuamente conectar-se ao servidor no endereço especificado. As mensagens são codificadas com o padrão *UTF-8* e separadas pelo caractere de fim de texto *0x03* (`\x03` em Python).

Parâmetros:

- **ip** (*str*) – O endereço IP do servidor (por exemplo, `"192.168.80.2"`)
- **port** (*int*) – O número da porta através da qual o servidor é acessível (ex.: `60001`). As portas estão limitadas ao intervalo entre 60000 e 61000.

Retorno:

O objeto cliente de socket TCP, que é usado para enviar e receber mensagens.

Tipo de retorno:

Socket

Exemplos:

Este exemplo realiza um handshake entre um cliente de socket TCP na aplicação e um servidor de socket TCP na mesma rede. Primeiro, a conexão é estabelecida. Em seguida, o cliente aguarda o servidor enviar uma mensagem inicial. Ao receber essa mensagem, o cliente responde com sua própria mensagem e, finalmente, fecha a conexão.

```
client = create_tcp_client("192.168.80.2", 60001)
print(client.receive())
client.send("I am your client")
client.close()
```

create_tcp_server(*port*)

Criando um servidor de socket TCP que pode lidar com até 20 clientes. As mensagens são codificadas com o padrão *UTF-8*. Além disso, as mensagens são separadas pelo caractere de fim de texto *0x03* (`\x03` em Python).

Parâmetros:

port (*int*) – O número da porta através do qual este servidor pode ser acessado pelos clientes (ex.: `60002`). As portas estão limitadas ao intervalo entre 60000 e 61000.

Retorno:

O objeto servidor de socket TCP, que é usado para enviar e receber mensagens.

Tipo de retorno:

Socket

Exemplos:

Este exemplo realiza um handshake entre um servidor de socket TCP na aplicação e o primeiro cliente que se conecta a ele. Primeiro, o servidor abre a conexão em uma porta específica e aguarda a conexão de um cliente. Em seguida, o servidor aguarda que o cliente envie uma mensagem inicial. Ao recebê-la, o servidor responde com sua própria mensagem e, finalmente, fecha a conexão com todos os clientes conectados.

```
server = create_tcp_server(60002)
print(server.receive()
server.send("I am your server")
server.close()
```

create_udp_socket(*port*)

Criando um socket UDP para transmissão de mensagens. As mensagens são codificadas com o padrão *UTF-8* e separadas pelo caractere de fim de texto *0x03* (`\x03` em Python).

Parâmetros:

port (*int*) – O número da porta através do qual este socket pode ser acessado (ex.: `60003`). As portas estão limitadas ao intervalo entre 60000 e 61000.

Retorno:

O objeto socket UDP, que é usado para enviar e receber mensagens.

Tipo de retorno:

Socket

Exemplos:

Este exemplo realiza um handshake entre um socket UDP na aplicação e a primeira conexão que se abre para ele. Primeiro, o socket abre a conexão em uma porta específica e aguarda a conexão de algum par. Em seguida, o socket aguarda que o par envie uma mensagem inicial. Ao recebê-la, o socket responde com sua própria mensagem e, finalmente, fecha a conexão.

```
udp_socket = create_udp_socket(60003)
print(udp_socket.receive()
udp_socket.send("I am your socket")
udp_socket.close()
```

send_message(*message*)

Enviando uma mensagem com a chave “script_send” para uma conexão TCP aberta.

Parâmetros:

message (*str*) – A mensagem a ser enviada.

Exemplos:

Enviando uma mensagem via TCP para qualquer dispositivo que possa ouvi-la:

```
send_message("Hello World!")
```

No lado do dispositivo conectado, estes dados são recebidos no seguinte formato:

```
{"script_send": "Hello World!"}
```

receive_message(*timeout*=*None*)

Recebendo uma mensagem de uma conexão TCP aberta chamando a ação “script_receive”.

Parâmetros:

timeout (*None or float*) – O tempo máximo para aguardar uma mensagem. Se não for especificado, aguarde indefinidamente.

Exemplos:

Um dispositivo externo conecta-se ao robô via TCP e envia os seguintes dados:

```
{"bridge": "core", "action": "script_receive", "message": "Hello World!"}
```

No lado do robô, esta função pode ser usada para receber e imprimir a mensagem:

```
message = receive_message(timeout=3.0)
print(message)
```

clear_messages()

Limpando a fila de mensagens TCP de mensagens recebidas anteriormente que podem não ter sido lidas.

Exemplos:

Antes de aguardar uma nova mensagem, às vezes faz sentido garantir que mensagens antigas sejam rejeitadas.

```
clear_message()
message = receive_message()
```

class Coordinates

O objeto Coordinates define uma transformação matemática que pode ser usada, por exemplo, como coordenadas alvo em funções de movimento ou para definir frames de coordenadas.

__init__(*coordinates*)

Crie um objeto Coordinates, seja com valores de coordenadas específicos, ou contendo as coordenadas atuais do robô.

Parâmetros:

coordinates (*list, dict, or Coordinates*) –

Um conjunto específico de coordenadas, que pode ser um dos seguintes:

- Uma lista na forma [x, y, z, roll, pitch, yaw] com valores de posição em [mm] e valores de orientação em [°]
- Um dicionário na forma {"x": x, "y": y, "z": z, "roll": roll, "pitch": pitch, "yaw": yaw} com valores de posição em [mm] e valores de orientação em [°]
- Um objeto Coordinates (copiando outro objeto Coordinates)

Construtor:

Cria um objeto Coordinates a partir de valores de coordenadas.

```
coordinates = Coordinates({"x": 100, "y": 200, "z": 300,
                           "roll": 15, "pitch": -15, "yaw": 30})
coordinates = Coordinates([100, 200, 300, 15, -15, 30])
```

Cria um objeto Coordinates que contém as coordenadas atuais do robô usando a função `get_current_coordinates`.

```
current_coordinates = get_current_coordinates()
```

***property* position**

A parte de posição do objeto Coordinates.

Retorno:

A posição na forma [x, y, z] em [mm].

Tipo de retorno:

list of float

***property* orientation**

A parte de orientação do objeto Coordinates.

Retorno:

A orientação na forma [roll, pitch, yaw] em [°].

Tipo de retorno:

list of float

to_list()

Converte o objeto Coordinates em uma lista de valores de coordenadas na forma [x, y, z, roll, pitch, yaw], onde os valores de posição estão em [mm] e os valores de rotação em [°].

Retorno:

Valores de coordenadas em formato de lista.

Tipo de retorno:

list of float

Exemplos:

Esta função pode ser usada quando as Coordinates são necessárias como lista:

```
coord = Coordinates({"x": 700.0, "y": -125.0, "z": 500.0,
                     "roll": 180.0, "pitch": 0.0, "yaw": -90.0})

# convert to list
coord_list = coord.to_list()
```

to_dict()

Converte o objeto Coordinates em um dicionário de valores de coordenadas na forma {«x»: x, «y»: y, «z»: z, «roll»: roll, «pitch»: pitch, «yaw»: yaw}, onde os valores de posição estão em [mm] e os valores de rotação em [°].

Retorno:

Valores de coordenadas em formato de dicionário.

Tipo de retorno:

dict of float

Exemplos:

Esta função pode ser usada quando as Coordinates são necessárias como dicionário:

```
coord = Coordinates([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])

# convert to dict
coord_dict = coord.to_dict()
```

inverted()

Calcula a transformação inversa deste objeto Coordinates.

Retorno:

inverso do objeto de transformação

Tipo de retorno:

Transform

Exemplos:

```
c = Coordinates([100, 200, 300, 15, -15, 30])

# create the inverse transform
c_inv = c.inverted()

# the inverse transform applied to the original transform
# this results in the identity transform
assert c * c_inv == Coordinates([0, 0, 0, 0, 0, 0])
assert c_inv * c == Coordinates([0, 0, 0, 0, 0, 0])
```

class Pose

O objeto Pose define uma pose do robô, consistindo de suas coordenadas (posição e orientação) e sua configuração (uma vez que o mesmo conjunto de coordenadas pode ser alcançado com múltiplas configurações).

__init__(*pose*)

Cria um objeto Pose, seja carregando uma pose salva por nome ou ID, ou a partir da pose atual do robô.

Parâmetros:

pose (*str, int, or Pose*) –

Referência para a pose a ser carregada, que pode ser um dos seguintes:

- o nome (str) de uma pose salva no banco de dados.
- o ID (int) de uma pose salva no banco de dados
- Um objeto Pose (copiando outro objeto Pose)

Construtor:

Carrega uma pose do banco de dados.

```
pose = Pose("start_pose")
```

Cria um objeto Pose que contém a pose atual do robô usando a função `get_current_pose`.

```
current_pose = get_current_pose()
```

Cria um objeto Pose que contém as coordenadas e configuração especificadas usando a função `create_pose`.

```
pose = create_pose(coordinates, configuration)
```

***property* coordinates**

As coordenadas desta pose, que é um objeto Coordinates contendo os valores de posição e orientação.

Retorno:

As coordenadas desta pose.

Tipo de retorno:

Coordinates

***property* configuration**

Como um conjunto de coordenadas de pose pode ser alcançado com até 8 configurações possíveis das juntas do robô, a configuração é definida como um número (de 0 a 7) ou uma string (de “c1” a “c8”) para definir a pose de forma única.

Retorno:

O índice da configuração da pose (de 0 a 7).

Tipo de retorno:

int

class LinearSegment

O objeto LinearSegment descreve um segmento de trajeto linear do TCP (ponto central da ferramenta) até as coordenadas desejadas. Além disso, inclui informações sobre a velocidade linear desejada, aceleração linear e o raio de blend no início do segmento. Esse raio de blend permite uma pequena variação do caminho linear para garantir uma transição suave entre os segmentos.

__init__(*linear_segment*)

Criar um objeto LinearSegment.

Parâmetros:

- **coordinates** (*list, dict, or Coordinates*) – Coordenadas de destino.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usado o valor global de velocidade linear.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usado o valor global de aceleração linear.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, será usado 0.0.

Construtor:

Criando um objeto LinearSegment definindo todos os parâmetros:

```
linear_seg = LinearSegment([100, 200, 300, 15, -15, 30],
                           linear_velocity=500, linear_acceleration=1000,
                           blend_radius=100)
```

ou usando os valores padrão:

```
linear_seg = LinearSegment([100, 200, 300, 15, -15, 30])
```

***property* coordinates**

As coordenadas de destino deste segmento, que é um objeto Coordinates contendo os valores de posição e orientação.

Retorno:

As coordenadas de destino deste segmento.

Tipo de retorno:

Coordinates

***property* blend_radius**

O raio de blend no início deste segmento.

Retorno:

O raio de blend inicial.

Tipo de retorno:

float

***property* linear_acceleration**

A aceleração linear desejada deste segmento. Se for None, será usado o valor global de aceleração linear.

Retorno:

A aceleração linear desejada.

Tipo de retorno:

float or None

***property* linear_velocity**

A velocidade linear desejada deste segmento. Se for None, será usado o valor global de velocidade linear.

Retorno:

A velocidade linear desejada.

Tipo de retorno:

float or None

class CircularSegment

O objeto CircularSegment descreve um movimento circular do TCP (ponto central da ferramenta) através de uma posição intermediária até as coordenadas desejadas. Note que a orientação dessa pose intermediária é definida pelo algoritmo. Além disso, inclui informações sobre a velocidade linear desejada, aceleração linear e o raio de blend no início do segmento. Esse raio de blend permite uma pequena variação do caminho circular para garantir uma transição suave entre os segmentos.

__init__(*circular_segment*)

Criar um objeto CircularSegment.

Parâmetros:

- **intermediate_position** (*list*) – Uma posição intermediária [x, y, z] no caminho até as coordenadas de destino. A posição intermediária determina a curvatura do movimento circular.
- **coordinates** (*list, dict or Coordinates*) – Coordenadas de destino.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto CircularSegment definindo todos os parâmetros:

```
circular_seg = CircularSegment([0, 300, 300], [100, 200, 300, 15, -15, 30],
                               linear_velocity=500, linear_acceleration=1000,
                               blend_radius=100)
```

ou usando os valores padrão:

```
circular_seg = CircularSegment([0, 300, 300], [100, 200, 300, 15, -15, 30])
```

***property* intermediate_position**

A posição intermediária deste segmento. Inclui as coordenadas x, y, z: [x, y, z]

Retorno:

A posição intermediária deste segmento.

Tipo de retorno:

list

***property* coordinates**

As coordenadas de destino deste segmento, que é um objeto Coordinates contendo os valores de posição e orientação.

Retorno:

As coordenadas de destino deste segmento.

Tipo de retorno:

Coordinates

***property* blend_radius**

O raio de blend no início deste segmento.

Retorno:

O raio de blend inicial.

Tipo de retorno:

float

***property* linear_acceleration**

A aceleração linear desejada deste segmento. Se for None, será usado o valor global de aceleração linear.

Retorno:

A aceleração linear desejada.

Tipo de retorno:

float or None

***property* linear_velocity**

A velocidade linear desejada deste segmento. Se for None, será usado o valor global de velocidade linear.

Retorno:

A velocidade linear desejada.

Tipo de retorno:

float or None

class OrientationSegment

O objeto OrientationSegment descreve um segmento de trajetória de orientação pura, onde o TCP (ponto central da ferramenta) gira de acordo com a orientação desejada da ferramenta, mantendo sua posição constante. Além disso, inclui informações sobre a velocidade angular desejada, aceleração angular e raio de blend no início do segmento. Se o raio de blend for maior que 0, a rotação do TCP começa antes de alcançar a posição alvo.

__init__(*orientation_segment*)

Criar um objeto OrientationSegment.

Parâmetros:

- **orientation** (*list*) – Orientação alvo. Observe que a posição permanece constante.
- **angular_velocity** (*float or None*) – Velocidade angular desejada [°/s]. Se nenhuma velocidade angular for definida, será usada a velocidade angular global.
- **angular_acceleration** (*float or None*) – Aceleração angular desejada [°/s^2]. Se nenhuma aceleração angular for definida, será usada a aceleração angular global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Um raio de blend maior que 0 permite que o TCP (ponto central da ferramenta) gire antes que a posição deste segmento seja alcançada. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto OrientationSegment definindo todos os parâmetros:

```
orientation_seg = OrientationSegment([15, -15, 30],
                                     angular_velocity=250,
angular_acceleration=500,
                                     blend_radius=100)
```

ou usando os valores padrão:

```
orientation_seg = OrientationSegment([15, -15, 30])
```

***property* orientation**

A orientação alvo deste segmento. Fornecida como ângulos náuticos [roll, pitch, yaw] em graus

Retorno:

A orientação alvo deste segmento.

Tipo de retorno:

list

***property* angular_velocity**

A velocidade angular desejada deste segmento. Se for None, será usada a velocidade angular global.

Retorno:

A velocidade angular desejada.

Tipo de retorno:

float or None

***property* angular_acceleration**

A aceleração angular desejada deste segmento. Se for None, será usada a aceleração angular global.

Retorno:

A aceleração angular desejada.

Tipo de retorno:

float or None

***property* blend_radius**

O raio de blend no início deste segmento.

Retorno:

O raio de blend inicial.

Tipo de retorno:

float

class CoordinatesSegment

O objeto CoordinatesSegment descreve um movimento linear no espaço das juntas até as coordenadas alvo. O TCP (ponto central da ferramenta), por outro lado, geralmente não se move linearmente. Além disso, inclui informações sobre a velocidade linear desejada, aceleração linear e raio de blend no início do segmento. Esse raio de blend permite uma pequena variação no caminho para garantir uma transição suave entre segmentos.

__init__(*coordinates_segment*)

Criar um objeto CoordinatesSegment.

Parâmetros:

- **coordinates** (*list, dict or Coordinates*) – Coordenadas de destino.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto CoordinatesSegment definindo todos os parâmetros:

```
coordinates_seg = CoordinatesSegment([100, 200, 300, 15, -15, 30],
                                     linear_velocity=250,
linear_acceleration=500,
                                     blend_radius=100)
```

ou usando os valores padrão:

```
coordinates_seg = CoordinatesSegment([100, 200, 300, 15, -15, 30])
```

***property* coordinates**

As coordenadas de destino deste segmento, que é um objeto Coordinates contendo os valores de posição e orientação.

Retorno:

As coordenadas de destino deste segmento.

Tipo de retorno:

Coordinates

***property* blend_radius**

O raio de blend no início deste segmento.

Retorno:

O raio de blend inicial.

Tipo de retorno:

float

***property* linear_acceleration**

A aceleração linear desejada deste segmento. Se for None, será usado o valor global de aceleração linear.

Retorno:

A aceleração linear desejada.

Tipo de retorno:

float or None

***property* linear_velocity**

A velocidade linear desejada deste segmento. Se for None, será usado o valor global de velocidade linear.

Retorno:

A velocidade linear desejada.

Tipo de retorno:

float or None

class PoseSegment

O objeto PoseSegment descreve um movimento linear no espaço das juntas até a pose alvo. O TCP (ponto central da ferramenta), por outro lado, geralmente não se move linearmente. Além disso, inclui informações sobre a velocidade linear desejada, aceleração linear e raio de blend no início do segmento. Esse raio de blend permite uma pequena variação no caminho para garantir uma transição suave entre segmentos.

__init__(*pose_segment*)

Criar um objeto PoseSegment.

Parâmetros:

- **pose** (*Pose*) – Pose alvo.
- **linear_velocity** (*float or None*) – Velocidade linear desejada [mm/s]. Se nenhuma velocidade linear for definida, será usada a velocidade linear global.
- **linear_acceleration** (*float or None*) – Aceleração linear desejada [mm/s^2]. Se nenhuma aceleração linear for definida, será usada a aceleração linear global.
- **blend_radius** (*float or None*) – Raio de blend no início do segmento [mm]. Se nenhum raio de blend for definido, 0.0 será usado.

Construtor:

Criando um objeto PoseSegment definindo todos os parâmetros:

```
pose_seg = PoseSegment(create_pose([100, 200, 300, 15, -15, 30], "c1"),
                       linear_velocity=250, linear_acceleration=500,
                       blend_radius=100)
```

ou usando os valores padrão e a pose do banco de dados:

```
pose_seg = PoseSegment("target_pose")
```

***property* pose**

A pose alvo deste segmento, que é um objeto Pose contendo os valores de posição, orientação e configuração.

Retorno:

A pose alvo deste segmento.

Tipo de retorno:

Pose

***property* blend_radius**

O raio de blend no início deste segmento.

Retorno:

O raio de blend inicial.

Tipo de retorno:

float

***property* linear_acceleration**

A aceleração linear desejada deste segmento. Se for None, será usado o valor global de aceleração linear.

Retorno:

A aceleração linear desejada.

Tipo de retorno:

float or None

***property* linear_velocity**

A velocidade linear desejada deste segmento. Se for None, será usado o valor global de velocidade linear.

Retorno:

A velocidade linear desejada.

Tipo de retorno:

float or None

class Path

O objeto Path define como o robô se move entre duas ou mais poses. Um caminho contínuo é definido apenas no espaço da ferramenta por múltiplos segmentos, como linear ou circular. Um caminho é chamado de contínuo sem interrupções se as soluções discretas do caminho não exigirem que o robô se reposicione (mudança de configuração). Um caminho é chamado de contínuo se a aceleração for contínua ao longo do caminho. Um caminho começa em uma pose especificada (em oposição às coordenadas). Os segmentos do caminho geralmente definem coordenadas, enquanto a configuração do caminho é definida pela pose inicial.

__init__(*path*)

Crie um objeto Path carregando um caminho salvo pelo nome ou ID.

Parâmetros:

path (*str, int, or Path*) –

Referência ao caminho a ser carregado, que pode ser um dos seguintes:

- o nome (str) de um caminho salvo no banco de dados
- o ID (int) de um caminho salvo no banco de dados
- um objeto Path (copiando outro objeto Path)

Construtor:

Carregar um caminho do banco de dados.

```
path = Path("pick_place_path")
```

Crie um objeto Path que contenha a pose inicial especificada e os segmentos de caminho definidos usando a função `create_path`. Note que os segmentos também podem ser adicionados posteriormente.

```
path = create_path(initial_pose, segments)
```

***property* initial_pose**

A pose inicial do robô no caminho.

Retorno:

Pose inicial do robô.

Tipo de retorno:

Pose

***property* segments**

Os segmentos incluídos no caminho. Observe que os segmentos não precisam ser todos do mesmo tipo.

Retorno:

Segmentos incluídos.

Tipo de retorno:

list of Segments (LinearSegment, CircularSegment, OrientationSegment, CoordinatesSegment, PoseSegment)

add_segment(*segment*)

Adiciona um segmento ao final do caminho.

Parâmetros:

segment (*LinearSegment*, *CircularSegment*, *OrientationSegment*, *CoordinatesSegment*, *PoseSegment*) – Segmento a ser adicionado.

Exemplos:

Um segmento pode ser adicionado da seguinte forma:

```
# creating and adding a segment
lin_seg = LinearSegment([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])
path.add_segment(lin_seg)
```

add_linear(*coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Adiciona um segmento linear ao final do caminho.

Parâmetros:

- **coordinates** (*list, dict or Coordinates*) – Coordenadas alvo do caminho.
- **linear_velocity** (*float*) – Velocidade linear máxima permitida da ferramenta [mm/s].
- **linear_acceleration** (*float*) – Aceleração linear máxima permitida da ferramenta [mm/s^2].
- **blend_radius** (*float*) – Raio de blend permitido [mm] no início do segmento.

Exemplos:

Um segmento linear pode ser adicionado da seguinte forma:

```
# add a linear segment with different argument types for the "coordinates"
parameter
path.add_linear([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])
path.add_linear({"x": 700.0, "y": -125.0, "z": 500.0,
                 "roll": 180.0, "pitch": 0.0, "yaw": -90.0})
path.add_linear(Coordinates([700.0, -125.0, 500.0, 180.0, 0.0, -90.0]))
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
# first linear segment is added
coord = Coordinates([700.0, -250.0, 500.0, 180.0, 0.0, -90.0])
path.add_linear(coord, linear_velocity=500.0, linear_acceleration=1000.0)

# second linear segment is added
path.add_linear([700.0, 250.0, 500.0, 180.0, 0.0, -90.0],
                linear_velocity=250.0, linear_acceleration=500.0,
blend_radius=100.0)
```

O primeiro segmento define um movimento linear em direção às coordenadas alvo especificadas com uma velocidade máxima de 500 mm/s e uma aceleração máxima de 1000 mm/s^2. Observe que essas coordenadas não são atingidas exatamente, pois o segmento seguinte define um raio de blend de 100 mm, permitindo uma variação dessas coordenadas para garantir uma transição mais suave e eficiente. O segundo segmento adiciona então um movimento linear para as próximas coordenadas com uma velocidade máxima de 250 mm/s e uma aceleração máxima de 500 mm/s^2.

add_circular(*intermediate_position*, *coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Adiciona um segmento circular ao final do caminho.

Parâmetros:

- **intermediate_position** (*list*) – Uma posição intermediária ([x, y, z]) no caminho em direção às coordenadas alvo. A posição intermediária determina a curvatura do movimento circular.
- **coordinates** (*list, dict or Coordinates*) – Coordenadas alvo do caminho.
- **linear_velocity** (*float*) – Velocidade linear máxima permitida da ferramenta [mm/s].
- **linear_acceleration** (*float*) – Aceleração linear máxima permitida da ferramenta [mm/s^2].
- **blend_radius** (*float*) – Raio de blend permitido [mm] no início do segmento.

Exemplos:

Um segmento circular pode ser adicionado da seguinte forma:

```
# add a circular segment with different argument types for the "coordinates"
parameter
path.add_circular([0.0, -500.0, 500.0], [-500.0, -125.0, 500.0, 180.0, 0.0,
-90.0])
path.add_circular([0.0, -500.0, 500.0], {"x": -500.0, "y": -125.0, "z": 500.0,
                  "roll": 180.0, "pitch": 0.0, "yaw": -90.0})
path.add_circular([0.0, -500.0, 500.0],
                  Coordinates([-500.0, -125.0, 500.0, 180.0, 0.0, -90.0]))
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
# first circular segment is added
coord = Coordinates([-500.0, 250.0, 500.0, 180.0, 0.0, -90.0])
path.add_circular([-700.0, 0.0, 500.0], coord,
                  linear_velocity=500.0, linear_acceleration=1000.0)

# second circular segment is added
path.add_circular([-700.0, 0.0, 500.0], [-500.0, -250.0, 500.0, 180.0, 0.0,
-90.0],
                  linear_velocity=250.0, linear_acceleration=500.0,
blend_radius=0.0)
```

O primeiro segmento define um movimento circular através da posição intermediária especificada em direção às coordenadas alvo com uma velocidade máxima de 500 mm/s e uma aceleração máxima de 1000 mm/s^2. Note que essas coordenadas não são alcançadas exatamente, pois o segmento consecutivo define um raio de mistura de 100 mm, permitindo

uma pequena variação para garantir uma transição mais suave e eficiente. O segundo segmento adiciona então um movimento circular através da próxima posição intermediária em direção às próximas coordenadas alvo com uma velocidade máxima de 250 mm/s e uma aceleração máxima de 500 mm/s^2.

add_orientation(*coordinates*, *angular_velocity=None*, *angular_acceleration=None*, *blend_radius=None*)

Adiciona um segmento de orientação ao final do caminho.

Parâmetros:

- **coordinates** (*list, dict or Coordinates*) – Coordenadas alvo do caminho.
- **angular_velocity** (*float*) – Velocidade angular máxima permitida da ferramenta [°/s].
- **angular_acceleration** (*float*) – Aceleração angular máxima permitida da ferramenta [°/s²].
- **blend_radius** (*float*) – Raio de mistura permitido [mm] no início do segmento. Um valor maior que 0 significa que ele pode começar a girar antes de atingir a posição desejada deste segmento.

Exemplos:

Um segmento de orientação pode ser adicionado da seguinte forma:

```
# add an orientation segment
path.add_orientation([150.0, 0.0, -90.0])
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
# first orientation segment is added
path.add_orientation([180.0, 0.0, -90.0],
                     angular_velocity=250.0, angular_acceleration=500.0)

# second orientation segment is added
path.add_orientation([150.0, 0.0, -90.0], angular_velocity=500.0,
                     angular_acceleration=1000.0, blend_radius=100.0)
```

O primeiro segmento define uma rotação para a orientação alvo especificada com uma velocidade máxima de 250°/s e uma aceleração máxima de 500°/s². Note que, ao executar este caminho, essa primeira rotação será ignorada, pois o raio de mistura do segmento consecutivo é maior que 0 (veja `OrientationSegment.__init__`), e ele executará diretamente o segundo segmento. Este segmento define uma rotação para a orientação alvo de [150.0, 0.0, -90.0] com uma velocidade máxima de 500°/s e uma aceleração máxima de 1000°/s². Em contraste, se o raio de mistura do segundo segmento fosse definido como 0.0, ambas as rotações seriam executadas em sequência.

add_coordinates(*coordinates*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Adiciona um segmento de coordenadas ao final do caminho.

Parâmetros:

- **coordinates** (*list, dict or Coordinates*) – Coordenadas alvo do caminho.
- **linear_velocity** (*float*) – Velocidade linear máxima permitida da ferramenta [mm/s].
- **linear_acceleration** (*float*) – Aceleração linear máxima permitida da ferramenta [mm/s^2].
- **blend_radius** (*float*) – Raio de blend permitido [mm] no início do segmento.

Exemplos:

Um segmento de coordenadas pode ser adicionado da seguinte forma:

```
# different argument types for the "coordinates" parameter
path.add_coordinates([700.0, -125.0, 500.0, 180.0, 0.0, -90.0])
path.add_coordinates({"x": 700.0, "y": -125.0, "z": 500.0,
                      "roll": 180.0, "pitch": 0.0, "yaw": -90.0})
path.add_coordinates(Coordinates([700.0, -125.0, 500.0, 180.0, 0.0, -90.0]))
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
# first coordinates segment is added
coord = Coordinates([700.0, -250.0, 500.0, 180.0, 0.0, -90.0])
path.add_coordinates(coord, linear_velocity=500.0, linear_acceleration=1000.0)

# second coordinates segment is added
path.add_coordinates([700.0, 250.0, 500.0, 180.0, 0.0, -90.0],
linear_velocity=250.0,
                     linear_acceleration=500.0, blend_radius=100.0)
```

O primeiro segmento define um movimento em direção às coordenadas alvo especificadas com uma velocidade máxima de 500 mm/s e uma aceleração máxima de 1000 mm/s^2. Note que essas coordenadas não são alcançadas exatamente, pois o segmento consecutivo define um raio de mistura de 100 mm, permitindo uma pequena variação para garantir uma transição mais suave e eficiente. O segundo segmento adiciona então um movimento para as próximas coordenadas com uma velocidade máxima de 250 mm/s e uma aceleração máxima de 500 mm/s^2.

add_pose(*pose*, *linear_velocity=None*, *linear_acceleration=None*, *blend_radius=None*)

Adiciona um segmento de pose ao final do caminho.

Parâmetros:

- **pose** (*Pose*) – Coordenadas alvo do caminho.
- **linear_velocity** (*float*) – Velocidade linear máxima permitida da ferramenta [mm/s].
- **linear_acceleration** (*float*) – Aceleração linear máxima permitida da ferramenta [mm/s^2].
- **blend_radius** (*float*) – Raio de blend permitido [mm] no início do segmento.

Exemplos:

Um segmento de pose pode ser adicionado da seguinte forma:

```
# using a pose from the database
pose_db = get_pose("pick_pose")
path.add_pose(pose_db)

# using a newly created pose
new_pose = create_pose([700.0, -125.0, 500.0, 180.0, 0.0, -90.0], "c8")
path.add_pose(new_pose)
```

Nestes exemplos, nenhum parâmetro opcional foi especificado. Portanto, os parâmetros globais são usados. Os parâmetros também podem ser definidos explicitamente na chamada da função.

```
# first pose segment is added
new_pose = create_pose([700.0, -125.0, 500.0, 180.0, 0.0, -90.0], "c8")
path.add_pose(new_pose, linear_velocity=500.0, linear_acceleration=1000.0)

# second pose segment is added
path.add_pose(get_pose("pick_pose"), linear_velocity=250.0,
              linear_acceleration=500.0, blend_radius=100.0)
```

O primeiro segmento define um movimento em direção à pose alvo especificada com uma velocidade máxima de 500 mm/s e uma aceleração máxima de 1000 mm/s^2. Note que esta pose não é alcançada exatamente, pois o segmento consecutivo define um raio de mistura de 100 mm, permitindo uma pequena variação das coordenadas correspondentes para garantir uma transição mais suave e eficiente. O segundo segmento adiciona então um movimento para a próxima pose com uma velocidade máxima de 250 mm/s e uma aceleração máxima de 500 mm/s^2.

class Socket

O Socket é um objeto retornado pelas funções `create_tcp_client`, `create_tcp_server` e `create_udp_socket`. Use este objeto para enviar e receber dados através do socket.

receive(*timeout*=*None*)

Recebendo uma mensagem através deste socket.

Parâmetros:

timeout (*None or float*) – O tempo máximo para aguardar uma mensagem. Se não for especificado, aguarde indefinidamente.

Retorno:

A mensagem recebida, ou uma string vazia se ocorreu um tempo limite.

Tipo de retorno:

str

Exemplos:

Aguardando uma resposta por 3 segundos e imprimindo-a quando for recebida.

```
print(socket.receive(timeout=3.0))
```

send(*message*)

Enviando uma mensagem através deste socket.

Parâmetros:

message (*str*) – A mensagem a ser enviada.

Exemplos:

Enviar uma mensagem para todos os pares conectados.

```
socket.send("Hello world!")
```

close()

Fechando a conexão do socket.

Exemplos:

Fechar o socket.

```
socket.close()
```

Funções do add-on

Funções do Add-on Cognex

class Cognex

O módulo Cognex suporta o uso de câmeras Cognex. Observe que configurar a câmera Cognex requer a instalação do Cognex In-Sight Explorer.

Para mais informações sobre como configurar e conectar corretamente sua câmera Cognex ao robô, consulte o manual do usuário.

Todas as funções deste add-on são chamadas usando o prefixo `cognex.`

connect_to_camera(*ip*, *user*='*admin*', *password*='')

Configurando um cliente TCP que conecta à câmera Cognex conectada através da porta ISE para comandos nativos.

Parâmetros:

- **ip** (*str*) – O endereço IP da câmera, por exemplo `'10.0.0.210'`.
- **user** (*str*) – O nome de usuário para fazer login na câmera (conforme definido anteriormente no Cognex In-Sight Explorer). O nome de usuário padrão é `'admin'`.
- **password** (*str*) – A senha para fazer login na câmera (conforme definido anteriormente no Cognex In-Sight Explorer). A senha padrão é uma string vazia.

Levanta:

MinorError – se as credenciais estiverem incorretas, a conexão for abortada durante a conexão ou a conexão demorou demais.

Exemplos:

Conectar-se a uma câmera recém-entregue com o IP e credenciais padrão.

```
cognex.connect_to_camera("10.0.0.210")
```

Conectar-se a uma câmera com endereço IP e credenciais previamente modificados.

```
cognex.connect_to_camera("192.168.80.235", user="Developer",
password="my_password_1234")
```

trigger_camera(*ip*)

Configurando o evento de disparo da câmera («SW8»), que captura uma nova imagem e retorna os dados processados pelo trabalho ativo do Cognex.

Parâmetros:

ip (*str*) – O endereço IP da câmera, por exemplo `'10.0.0.210'`.

Retorno:

A mensagem de resposta da câmera, contendo as entidades detectadas.

Tipo de retorno:

str

change_job(*ip*, *job_name*)

Alterando o trabalho do Cognex via o comando nativo da câmera «LF», que carrega um arquivo de trabalho já armazenado na câmera. Observe que este comando pode levar vários segundos para ser executado.

Parâmetros:

- **ip** (*str*) – O endereço IP da câmera, por exemplo `'10.0.0.210'`.
- **job_name** (*str*) – O nome do trabalho para alternar (conforme definido anteriormente no Cognex In-Sight Explorer).

Levanta:

MinorError – se a câmera não existir ou a alteração do trabalho falhar.

Exemplos:

Alterar para um trabalho previamente armazenado como “CircleDetection.job” e, em seguida, tirar uma foto para detectar círculos na visualização atual da câmera.

```
ip_address = "10.0.0.210"
cognex.change_job(ip, "CircleDetection")
detected_circle_message = cognex.trigger_camera(ip)
```

get_camera_message(*ip*, *event_id*, *keyword*, *timeout=None*)

Acionar um evento na câmera e solicitar uma mensagem de soquete de palavra-chave específica no lado do Cognex.

Parâmetros:

- **ip** (*str*) – O endereço IP da câmera, por exemplo `'10.0.0.210'`.
- **event_id** (*int*) – O ID do evento Cognex a ser acionado.
- **keyword** (*str*) – A palavra-chave para aguardar.
- **timeout** (*float or None*) – O tempo limite da solicitação de espera (em [s]). Use `None` para esperar indefinidamente.

Levanta:

TimeOutError – se a câmera não respondeu com uma string válida antes do tempo limite expirar.

Exemplos:

Solicita a mensagem contendo a string “my_message”, vinculada ao evento 1 (SW1) do trabalho Cognex In-Sight.

```
message = cognex.get_camera_message("10.0.0.210", 1, "my_message")
```

disconnect_camera(*ip*)

Desconectando uma câmera específica.

Parâmetros:

ip (*str*) – O endereço IP da câmera, por exemplo `'10.0.0.210'`.

Funções do Add-on Modbus

class Modbus

O módulo Modbus contém funções para enviar solicitações a um servidor REST externo.

Todas as funções deste add-on são chamadas usando o prefixo `modbus.`

write_user_register(*address*, *values*)

Escrevendo nos registradores de saída Modbus.

Parâmetros:

- **address** (*int*) – O endereço do registrador para começar a escrever. Valores possíveis vão de 0x10 (16) a 0xFF (255).
- **values** (*int or list of int*) – O(s) valor(es) a serem escritos. Cada registrador pode armazenar até 2 bytes de dados. Portanto, o valor máximo é 65535 (=2^{16}-1). Se vários valores forem fornecidos, eles serão escritos em registradores consecutivos.

Exemplos:

Escreve um único valor em um endereço específico dos registradores de saída Modbus.

```
# write 11111 to the address 0x20
modbus.write_user_register(0x20, 11111)
```

Escreve múltiplos valores em registradores de saída consecutivos usando um único comando.

```
# write 555 to the address 0x30, and 777 to the address 0x31
modbus.write_user_register(0x30, [555, 777])
```

read_user_register(*address*, *count=1*)

Lendo os registradores de entrada Modbus.

Parâmetros:

- **address** (*int*) – O endereço do registrador para começar a ler. Valores possíveis vão de 0x10 (16) a 0xFF (255).
- **count** (*int*) – O número de registradores consecutivos a serem lidos. Por padrão, lê apenas um registrador.

Retorno:

Os valores lidos dos registradores. O tamanho desta lista é igual ao argumento *count*.

Tipo de retorno:

list of int

Exemplos:

Lê um único valor de um endereço específico dos registradores de entrada Modbus.

```
# read value from the address 0x20
value = modbus.read_user_register(0x20)[0]
```

Lê múltiplos valores de registradores de entrada consecutivos usando um único comando.

```
# read values from the addresses 0x30 and 0x31
values = modbus.read_user_register(0x30, 2)
```

wait_for_event(*index*)

Aguardando até que o evento com o índice especificado seja definido. Os eventos são definidos externamente pelo cliente Modbus.

Parâmetros:

index (*int*) – O índice do evento a ser aguardado. Índices possíveis vão de 0 a 15.

Exemplos:

Aguarda até que um evento seja definido.

```
modbus.wait_for_event(2)
```

Funções do Add-on REST

class Rest

O módulo REST contém funções para enviar solicitações a um servidor REST externo.

Todas as funções deste add-on são chamadas usando o prefixo `rest.`

setup_client_connection(*target_url*)

Configurando uma conexão de cliente para enviar solicitações a um servidor REST.

Parâmetros:

target_url (*str*) – A URL (endpoint geral da API) do servidor REST ao qual se conectar.

Retorno:

O objeto de conexão do cliente REST.

Tipo de retorno:

RestClientConnection

Exemplos:

Estabelecer uma conexão com um servidor REST.

```
connection = rest.setup_client_connection("https://some_url:5000/endpoint")
# ... this is similar to ...
connection = rest.setup_client_connection("https://some_url:5000/endpoint/")
```

class RestClientConnection

O RestClientConnection é um objeto retornado por `rest.setup_client_connection`. Ele é usado para enviar solicitações ao servidor REST conectado.

get_default_headers(*method*)

Lendo os cabeçalhos padrão que são enviados com cada solicitação.

Parâmetros:

method (*str*) – O método («get» ou «post») para o qual obter os cabeçalhos.

Retorno:

Os cabeçalhos padrão atualmente definidos para este método de solicitação.

Tipo de retorno:

dict

Exemplos:

Lendo os cabeçalhos padrão dos métodos GET e POST.

```
connection = rest.setup_client_connection("https://some_url:5000/endpoint")
# reading the default headers of the GET method, which by default are set to
#     {'Accept': 'application/json'}
headers = connection.get_default_headers("get")
# reading the default headers of the POST method, which by default are set to
#     {'Accept': 'application/json', 'Content-Type': 'application/json'}
headers = connection.get_default_headers("post")
```

set_default_headers(*method*, *new_headers*)

Alterar os cabeçalhos padrão que são enviados com cada solicitação.

Parâmetros:

- **method** (*str*) – O método («get» ou «post») para o qual definir os cabeçalhos.
- **new_headers** (*dict*) – Os novos cabeçalhos padrão para sobrescrever.

Exemplos:

Estabelecer uma conexão com um servidor REST.

```
connection = rest.setup_client_connection("https://some_url:5000/endpoint")
# resetting the headers of the GET method
connection.set_default_headers("get", {"Accept": "application/json"})
# resetting the headers of the POST method
connection.set_default_headers("post", {"Accept": "application/json",
"Content-Type": "application/json"})
```

get(*resource*, *headers_override=None*, *query_params=None*, *secure_connection=True*)

Enviando uma solicitação GET ao servidor REST.

Parâmetros:

- **resource** – O nome do recurso a ser acessado. O nome do recurso é adicionado à URL (<url>/<resource>) e pode opcionalmente conter um caractere "/" no início.
- **headers_override** (*dict*) – Defina os cabeçalhos para esta solicitação individual. Se não for especificado, ele usa os cabeçalhos padrão.
- **query_params** (*dict*) – Especificando uma query (se houver).
- **secure_connection** (*bool*) – Se deve usar uma conexão segura (verificar certificados) ou não.

Retorno:

O código de resposta (código HTTP, ex.: `200` para solicitação bem-sucedida) e o payload de resposta (dados de resposta) (se houver).

Tipo de retorno:

(int, dict)

Exemplos:

Enviando uma solicitação GET para um servidor REST e verificando seu código de resposta.

```
connection = rest.setup_client_connection("https://some_url:5000/endpoint")
code, payload = connection.get("my_resource")  # or
connection.get("/my_resource")
# print received data if request was successful:
if code == 200:
    print(payload)
```

post(*resource*, *data*, *headers_override=None*, *query_params=None*, *secure_connection=True*)

Enviando uma solicitação POST para o servidor REST.

Parâmetros:

- **resource** (*str*) – O nome do recurso a ser acessado. O nome do recurso é adicionado à URL (<url>/<resource>) e pode opcionalmente conter um caractere "/" no início.
- **data** (*str or dict*) – O corpo/dados da solicitação POST.
- **headers_override** (*dict*) – Defina os cabeçalhos para esta solicitação individual. Se não for especificado, ele usa os cabeçalhos padrão.
- **query_params** (*dict*) – Especificando uma query (se houver).
- **secure_connection** (*bool*) – Se deve usar uma conexão segura (verificar certificados) ou não.

Retorno:

O código de resposta (código HTTP, ex.: `200` para solicitação bem-sucedida) e o payload de resposta (dados de resposta) (se houver).

Tipo de retorno:

(int, dict)

Exemplos:

Enviando uma solicitação POST para um servidor REST e verificando seu código de resposta.

```
connection = rest.setup_client_connection("https://some_url:5000/endpoint")
code, payload = connection.post("my_resource")  # or
connection.post("/my_resource")
# print received data if request was successful:
if code == 200:
    print(payload)
```

Funções do Add-on ROS

class ROSHandler

O módulo ROS adiciona funcionalidades ROS para outros add-ons e para acesso direto pelos usuários.

Todas as funções deste add-on são chamadas usando o prefixo `ros_handler.`

ros_node()

O módulo ROS cria seu próprio nó ROS, que pode ser acessado diretamente através desta função. Isso pode ser usado para criar objetos como publishers, subscribers, etc., em scripts. Além disso, `rclpy` pode ser importado diretamente em aplicações para este propósito.

Aviso

Não execute `rclpy.spin()` neste nó, pois ele já está sendo executado em segundo plano!

Exemplos:

Imprimindo os nomes de todos os tópicos nos quais o robô está publicando.

```
for publisher in ros_handler.ros_node().publishers:
    print(publisher.topic_name)
```

Retorno:

O nó ROS do módulo ROS.

Tipo de retorno:

rclpy.node.Node