TRABALHO DE GRADUAÇÃO

# CONSTRUÇÃO DE UMA PLATAFORMA PARA DESENVOLVIMENTO DE PESQUISAS BASEADA NO ROBÔ MANIPULADOR UR3

**Rafael Ramos de Matos**

**Brasília, Novembro de 2021**

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

# CONSTRUÇÃO DE UMA PLATAFORMA PARA DESENVOLVIMENTO DE PESQUISAS BASEADA NO ROBÔ MANIPULADOR UR3

**Rafael Ramos de Matos**

*Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação*

Banca Examinadora

Prof. João Yoshiyuki Ishihara, ENE/UnB _____
*Orientador*

Prof. Henrique Cezar Ferreira, ENE/UnB _____
*Co-orientador*

Prof. Geovany Araujo Borges, ENE/UnB _____
*Examinador interno*

Prof. Mariana Bernardes, FGA/UnB _____
*Examinador interno*

**Brasília, Novembro de 2021**

**FICHA CATALOGRÁFICA**

R. MATOS, RAFAEL
Construção de uma plataforma para desenvolvimento de pesquisas baseada no robô manipulador UR3,

[Distrito Federal] 2021.

x, 114p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Robôs Colaborativos 2. ROS 3. Plataforma de desenvolvimento
4. Manipuladores Robóticos 5. UR3 6. Parâmetros de Denavit-Hartenberg

I. Mecatrônica/FT/UnB II. Título (Série)

**REFERÊNCIA BIBLIOGRÁFICA**

R. MATOS, RAFAEL (2021). Construção de uma plataforma para desenvolvimento de pesquisas baseada no robô manipulador UR3. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$04, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 114p.

**CESSÃO DE DIREITOS**

AUTOR: Rafael Ramos de Matos

Construção de uma plataforma para desenvolvimento de pesquisas baseada no robô manipulador UR3.

GRAU: Engenheiro ANO: 2021



---

Rafael Ramos de Matos

email: ramosunb@gmail.com

## Agradecimentos

*Primeiramente agradeço a Deus por ter escrito essa aventura, que foi árdua, porém também gratificante, chamada graduação em Eng. Mecatrônica na minha vida e por ter me ensinado, durante esse período, que mudanças requerem esforço e sacrifícios.*
*Para chegar até aqui devo a tantos que citando teria mais páginas que este trabalho. Tentarei resumir. Primeiramente agradeço a meus pais, Antônio Elton e Osvaldina pois devo tudo a eles, desde minha vida a cada aspecto construído nela. A meus irmãos Rakissia e Railson, pela imensa parceria em qualquer situação. A minha namorada Andressa, que me ajudou ao longo de tantas vezes que esqueci completamente de mim enquanto envolvido com alguma tarefa do curso. A minha família, tios, primos e avós por trazerem sempre a alegria em tudo. Aos três grandes amigos que fiz no curso, Antônio, Gabriel e Gustavo, que sempre estiveram nos momentos felizes e triste do curso ao meu lado.*
*Agradeço a cada um dos professores que me ajudaram em cada parte do caminho. Eugênio Fortaleza, Genaina Rodrigues, Marcos Lamar, Renato Borges, Rafael Shayani, Adolfo Bauchspiess, Henrique Ferreira, Francisco Assis, Alex da Rosa, Guilherme Ramos, Marcus Vinicius Lamar, Roberto de Souza Baptista, Bruno Luiggi e Marcos Caetano. Jones Yudi da Silva e Flávio Vidal, preceptores ao longo do curso que me ajudaram incontáveis vezes. Ao professor Geovany Araujo Borges, responsável pelo começo dos meus estudos na área de robótica. E aos professores João Ishihara e Henrique Ferreira pela admirável paciência e ajuda na orientação deste trabalho com diversas dicas extremamente precisas em vários momentos em que estava completamente perdido.*
*A cada, inclusive aos muitos não citados aqui, um sincero Muito Obrigado!*

*Rafael Ramos de Matos*

## RESUMO

Este trabalho tem o propósito de implementar uma plataforma para desenvolvimento de aplicações diversas para o UR3, um manipulador robótico antropomórfico composto por seis juntas e um efetuador terminal do tipo garra. Tendo em vista que o robô UR3 foi desenvolvido para uso comercial, sua arquitetura e software não são totalmente abertos e impossibilitam acesso e atuação em sinais de baixo nível. A falta de acesso a alguns sinais dificulta a aplicação de vários algoritmos de identificação, de controle e de estimação. Afim de reduzir esta dificuldade, acelerar a curva de aprendizado de novos usuários do robô manipulador UR3 e possibilitar a implementação de arquitetura cooperativa, neste trabalho é proposta uma plataforma de desenvolvimento que viabiliza uma mais rápida implementação de aplicações, como controladores e softwares de intercomunicação entre robôs. Além disso, foram estimados experimentalmente os parâmetros de *Denavit–Hartenberg* para o UR3 com o objetivo de estabelecer as orientações de cada junta do manipulador e garantir que as especificações do fabricante estavam corretas. Por fim, foi feita uma série de experimentos com tutoriais para complementar a documentação a respeito da usabilidade da plataforma de desenvolvimento com o intuito de estabelecer um primeiro contato de novos pesquisadores ao UR3.



Palavras Chave: Robôs Colaborativos, ROS, Plataforma de desenvolvimento, Denavit-Hartenberg, UR3, Manipuladores Robóticos



## ABSTRACT

This work aims to implement a platform for the development of diverse applications for the UR3, an anthropomorphic robotic manipulator composed of six joints and a claw-type end-effector. Given that the UR3 robot was developed for commercial use, its architecture and software are not fully open and make it impossible to access and act on low-level signals. The lack of access to some signals makes it difficult to apply various identification, control, and estimation algorithms. To reduce this difficulty, accelerate the learning curve of new users of the UR3 manipulator robot and enable the implementation of a cooperative architecture, this work proposes a development platform that enables faster implementation of applications, such as controllers and intercommunication software between robots. Furthermore, an estimation of the *Denavit–Hartenberg* parameters for the UR3 was carried out, in an experimental way, in order to establish the orientations of each manipulator's joint and ensure that the manufacturer's specifications were correct. Finally, a se-

ries of documentation with tutorials were made to complement the documentation regarding the usability of the development platform to establish the first contact for new researchers at UR3.

Keywords: Corobots, ROS, Development Platform, Denavit-Hartenberg, UR3

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE SÍMBOLOS

### Siglas

| | |
|---|---|
| API | Interface de Programação de Aplicações - *Application Program Interface* |
| ROS | Sistema Operacional de Robôs - *Robotic Operating System* |
| DH | *Denavit-Hartenberg* |
| LARA | Laboratório de Automação e Robótica da UnB |

# Capítulo 1

# Introdução

## 1.1 Contextualização

Ao longo de toda a história da humanidade, máquinas foram desenvolvidas para reduzir os esforços necessários para execução de tarefas repetitivas [6]. Durante a primeira revolução industrial (século XVII) as máquinas foram desenvolvidas para auxiliar o homem a produzir mais bens de consumo, em uma escala maior e com um alto grau de repetibilidade, visando garantia de qualidade dos produtos manufaturados [7]. Esse grande momento da nossa história moderna instigou diversas pessoas a pensar em como melhorar a produção dentro das indústrias [8], como o caso de Henry Ford, um empreendedor e engenheiro mecânico estadunidense, fundador da Ford Motor Company, que teve como principal legado para a evolução da indústria o fordismo [9].

Com o crescente desenvolvimento do campo industrial ao longo dos anos, máquinas mais especializas começaram a surgir para suprir a grande demanda por produtos na sociedade por bens de consumo. Dentre os vários tipos de máquinas existentes, o robô manipulador vem ganhando cada vez mais espaço na área de produção manufatureira e integrando, pouco a pouco, outros espaços como escolas, ateliês e hospitais. Os robôs manipuladores são definidos oficialmente pela norma ISO/TS 15066:2016 [1] como "manipuladores multipropósito controlados automaticamente, reprogramáveis, programáveis em três ou mais eixos". Desta forma, são máquinas extremamente flexíveis que permitem um processo de manufatura com diversidade de tarefas e cenários, garantido repetibilidade com consistência e precisão de forma automática.

No entanto, as mesmas características que conferem a robustez apreciada no ambiente industrial, tornam robôs perigosos. A força, peso e velocidade de operação compõem elementos de muitos riscos durante a execução de atividades na presença de pessoas. Para resolver este problema, novos tipos de robôs industriais têm sido desenvolvidos, incorporando características de segurança em cada aspecto construtivo, como a diminuição do peso, a capacidade de operação em velocidades reduzidas, os protocolos para interrupção em caso de acidentes e a complacência.

Uma nova arquitetura de robô completamente pensada em segurança, traz também uma maior

[1] `https://www.iso.org/obp/ui/#iso:std:iso:ts:15066:ed-1:v1:en`

dificuldade no controle. A ciência em torno do controle de sistemas rígidos está bem consolidada, porém, ainda são necessários estudos para garantir que robôs sejam precisos e complacentes ao mesmo tempo, isto é, que possuam certa maleabilidade em caso de colisão, permitindo assim a execução de tarefas sem danos.

Embora ainda exista no imaginário popular a visão da robótica como vilã na oferta de empregos [10], tem-se ampliado esforços para melhor cooperação entre máquinas e pessoas, a fim de facilitar cada vez mais a vida de todos [11]. Desta forma, a introdução de elementos para garantir a segurança humana traz os benefícios da robótica para ambientes fora da indústria sem a necessidade do isolamento das máquinas. Tal característica confere uma combinação entre a alta capacidade de repetibilidade e precisão dos robôs com a habilidade humana de rápida aprendizagem na hora de lidar com problemas complexos. Assim, com o decorrer do tempo e o emprego de novas tecnologias na robótica, novas oportunidades sugiram, transformando os trabalhadores manufatureiros e as novas gerações, não mais em agentes transformadores da matéria, mas em agentes transformadores do processo.

## 1.2 Definição do problema

No Laboratório de Automação e Robótica (LARA – UnB) encontra-se disponível o braço robótico articulado UR3 produzido pelo fabricante *Universal Robots* [2] que, daqui em diante, será chamado simplesmente de UR3.

O UR3 (Figura 1.1), segundo [12], é um robô colaborativo de mesa que possui carga útil de 3 kg. Seu tamanho reduzido o torna adequado para situações onde o espaço de trabalho é limitado. Com seu giro infinito na junta final, diversas atividades podem ser realizadas com a garra fixada no conector da ferramenta do robô, o que o torna muito hábil. No entanto, como o UR3 foi idealizado para o uso industrial e não para pesquisas acadêmicas, valores de alguns parâmetros não são fornecidos. Além disso, a arquitetura e o software do robô não são totalmente abertos para implementação de controladores de baixo nível, ou seja, seu sistema não permite a atuação em sinais como as correntes e tensões dos atuadores das juntas do robô. A falta de dados e de acesso a alguns sinais dificulta a aplicação de vários algoritmos de identificação, de controle e de estimação.

Como tentativa de obter os valores dos parâmetros dinâmicos (massas, centros de massa e inércias dos links) do UR3 não fornecidos pelo fabricante mas necessários para posterior uso em projetos de controladores, trabalhos de graduação anteriores tiveram a proposta de aplicar algoritmos de identificação. Enquanto [13] utilizou mecânica Lagrangeana, [14] utilizou mecânica Newtoniana na modelagem do manipulador. Também foram empregados diferentes métodos de otimização para obtenção dos valores numéricos dos parâmetros. Em comum, esses trabalhos tiveram parâmetros do modelo com valores fisicamente incoerentes, tais como massas e momentos de inércia negativos. Três possíveis fontes que podem explicar tais resultados:

[2] https://www.universal-robots.com

1. a convenção utilizada para fixar os sistemas de coordenadas dos juntas do manipulador não foi adequadamente aplicada;

2. a natureza dos sinais de entrada e de saída utilizados na identificação não corresponde àquela suposta pelos trabalhos acima. Por exemplo, o torque medido pode ser na realidade valor do torque após alguma compensação interna.

3. a unidade dos sinais não foi a correta. Por exemplo, a unidade da corrente não foi informada pelo fabricante se é em $mA$ ou $\mu A$; desconhecem-se as referências para as medidas de ângulo.

Considerando as dificuldades levantadas acima, este trabalho tem o seguinte objetivo.

### 1.2.1 Objetivo

A proposta é desenvolver uma plataforma de interface homem-robô que possibilite:

- rápida compreensão da operação do manipulador por parte do usuário;
- fácil reconhecimento de quais são os sinais de entrada e de saída envolvidos em um dado experimento e suas unidades;
- fácil implementação de algoritmos de identificação ou de controle do manipulador.

Note-se que não é objetivo desse trabalho fazer uma nova identificação dos parâmetros dinâmicos do UR3 mas sim fornecer um suporte para diminuir as dificuldades apontadas acima.

Para atingir o objetivo geral do trabalho foram propostos os seguintes objetivo específicos.

1. Construir uma documentação da Interface de Comunicação desenvolvida.

2. Descrição e desenvolvimento de um série de experimentos demostrativos sobre o funcionamento da plataforma de desenvolvimento, de modo a estimular e facilitar a produção de mais trabalhos usando o braço manipulador UR3.

3. Como exemplo de experimento, encontrar os parâmetros de *Denavit–Hartenberg* para o UR3.

Figura 1.1: Braço manipulador UR3

## 1.3 Apresentação do manuscrito

Este trabalho de graduação está organizado da seguinte forma. A descrição do *software* e do *hardware* do UR3 é feita no capítulo 2. Também é apresentada a plataforma de desenvolvimento, que integra o *hardware* e o *software*. No capítulo 3 a plataforma de desenvolvimento é utilizada para determinação dos parâmetros de Denavit-Hatenberg. Já no capítulo 4, a plataforma de desenvolvimento é utilizada para implementação de um sistema de controle de posição para uma das juntas do UR3. As conclusões e perspectivas futuras do trabalho são apresentas no capítulo 5. Finalmente, nos apêndices A até D são apresentados experimentos que permitem a um usuário da plataforma de desenvolvimento se familiarizar com ela para implementar, por exemplo, algoritmos de identificação, de controle ou de estimação, quando precisar.

# Capítulo 2

# Fundamentos Técnicos

Este capítulo tem o objetivo de fornecer informações que visam detalhar o *software* e o *hardware* que constituem a plataforma de desenvolvimento proposta nesse trabalho de graduação. Para a construção dessa plataforma e atingir os objetivos propostos na seção 1.2.1, foi realizado as seguintes tarefas:

- Estudo para entender como é feita comunicação do UR3 com um computador presente no LARA baseando-se na documentação do fabricante.
- Estudo de como integrar uma interface de comunicação, baseada no *framework* ROS, com a comunicação adotada pelo fabricante.
- Usar ferramentas de versionamento para fazer o gerenciamento da produção do *software* desenvolvido nesse trabalho.
- Desenvolver aplicações para a plataforma de forma para gerar resultados que embasem a proposta do trabalho de graduação.
- Escrever uma série de experimentos com intuito de acelerar a aprendizagem de novos usuários na plataforma.

## 2.1 Hardware

Essa seção terá o papel de esclarecer alguns pontos a respeito dos hardware que compõe a estrutura do robô manipulador UR3. Além disso, a sessão em questão, fará uma breve descrição da configuração do braço robótico.

### 2.1.1 Braço manipulador UR3

Para o desenvolvimento deste trabalho foi utilizado o manipulador robótico UR3 (Figura 2.4) da *Universal Robot* que é composto por 6 juntas rotacionais com atuadores elétricos. O braço

robótico foi desenvolvido para linhas de produção em tarefas em interação com pessoas de maneira segura em razão do baixo peso e por ter uma alta sensibilidade a colisão.

Cada uma das juntas do UR3 utiliza um atuador composto por **motor sem moldura** (Figura2.1), fabricados pela *Kollmorgen Automation Suite™*[15], em conjunto com uma redução baseada em engrenamento por onda de deformação (Figura 2.2), comercialmente denominado *Harmonic Drive*[16][17][18]. Além disso, cada junta possui acoplado a seus motores um **encoder** rotativo magnético da série AksIM fabricado pela empresa *SLR*[19]. Para fazer o controle de baixo nível individualmente, é usado uma placa desconhecida desenvolvida pelo fabricante que, por sua vez, é comandada por um computador chamado pelo fabricante de **Controller** (Figura 2.5) por meio de comunicação serial. Como forma de entender melhor o comportamento do robô, será explicado, com mais detalhes nas seções adiante, o funcionamento do sistema como um todo, desde o braço manipulador, Controller e acionamento do via Interface de Comunicação ROS usando um computador externo.


The KBM motors offer a great deal of freedom to configure the servo axes in a space-optimized way due to their variable, modular design.

Figura 2.1: Motor sem moldura da Kollmorgen Automation Suite

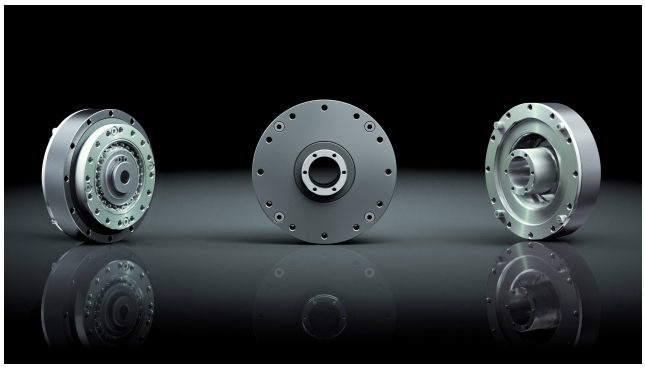

Figura 2.2: Harmonic Drive HFUS-2SH


Figura 2.3: Encoder rotativo magnético da série AksIM da RLS

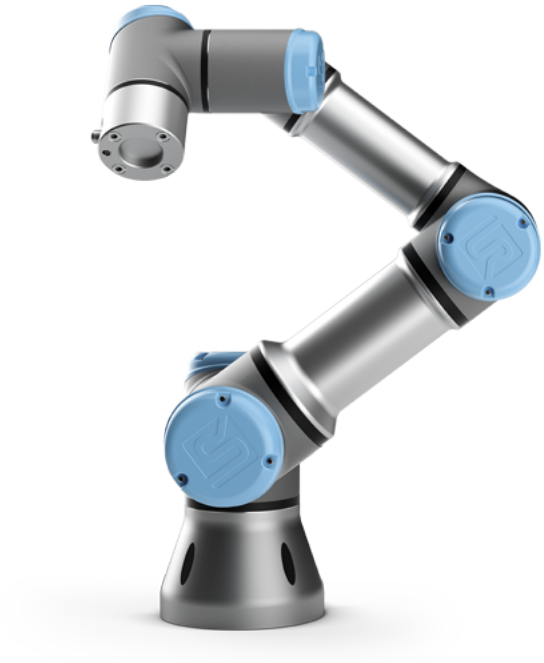

Figura 2.4: Braço manipulador UR3

### 2.1.2 Controller

A unidade de comunicação do UR3 (ver figura 2.5) é chamada de **Controller**. Esse componente contém as vias de entrada e saída digitais e analógicas que podem ser usadas para fazer comunicação com outros componentes, como pode ser visto na figura 2.7, ou dos próprios componentes do sistema [12]. Além disso, essa unidade de comunicação pode ser usada para programar e configurado o robô usando a linguagem de programação do fabricante sem que o usuária tem muita experiência a respeito do UR3 usando a interface de visualização **Polyscope**.


Figura 2.5: Controller

#### 2.1.2.1 Espaço de Trabalho e Configuração do braço manipulador UR3

O espaço de trabalho de um manipulador é o volume de espaço que o efetuador terminal do manipulador pode alcançar. O tamanho e a forma da área de trabalho dependem da geometria coordenada do braço do robô e também do número de graus de liberdade. O UR3 possui 6 graus de liberdade e, por ser um robô serial com juntas de revolução, tem uma configuração esférica.

A Figura 2.6, extraída de [12] página 35, mostra o espaço de trabalho do robô UR3, onde o raio da esfera mede 450 mm e o diâmetro do cilindro mede 200 mm. Assim, o espaço de trabalho total do robô manipulador UR3 é o volume da esfera menos o volume do cilindro.

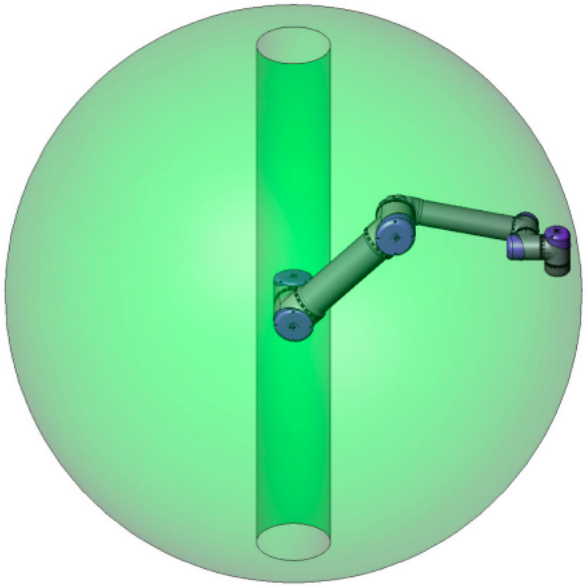

Figura 2.6: Espaço de trabalho do robô manipulador UR3

### 2.1.3 Diagrama de comunicação

O diagrama de comunicação na Figura 2.7 e 2.8, mostra o fluxo de dados que transita entre um computador com sistema operacional Ubuntu (Linux PC) e o robô manipulador UR3. Nesse diagrama, vemos a presença de um intermediário que faz a conexão entre Linux PC e o UR3 via cabo **Ethernet**, chamado **Controller**, que nada mais é do que a parte que compete ao *software* do robô UR3 implementada pela fabricante.

A Interface de Comunicação implementada no projeto de pesquisa [4], está presente no Linux PC e faz todo gerenciamento de dados durante o funcionamento do sistema como um todo. Além disso, a interface mostra, em tempo real, as variáveis de junta do robô UR3 em gráficos dinâmicos.

Para mais detalhe da descrição do sistema contido na Figura 2.7, acesse `https://cooprobo.readthedocs.io/en/latest/manipulators/ur3.html`.

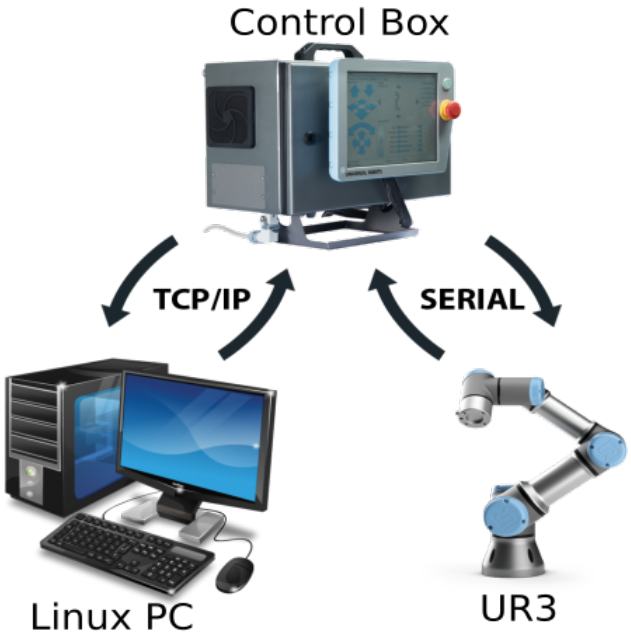


Figura 2.7: Diagrama de comunicação do robô

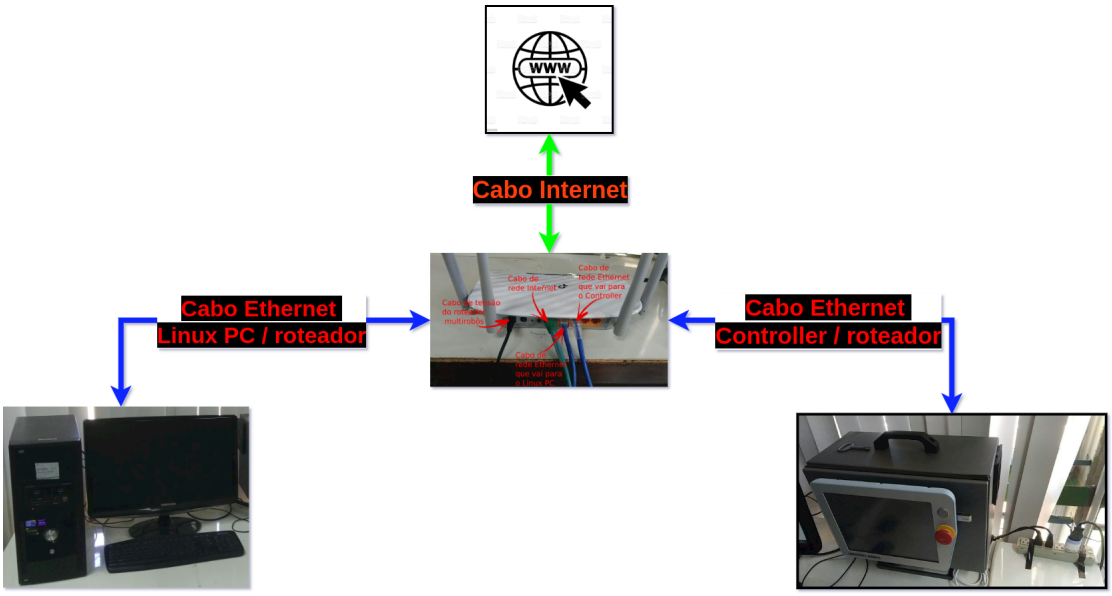


Figura 2.8: Diagrama de rede

### 2.1.4 Linux PC

O Linux PC, que é mostrado na Figura 2.7, é computador de mesa que tem um Linux como sistema operacional com a distribuição Ubuntu 18.04. Este computador, possui a API ROS instalado no seus sistema operacional, que possibilita a criação de aplicações ROS em uma camada de mais alto nível.

### 2.1.5 Controle de força do efetuador terminal

O controle de força do efetuador terminal, segundo a documentação do fabricante [1], é feito usando uma função da linguagem **urscript** [20] chamada **force_mode**. Essa função permite com que os robôs da *Universal Robots* sejam controlada tendo como realimentação a força exercida no seu efetuador terminal (Figura 2.9). Para mais detalhes do funcionamento do controle de força do fabricante consulte a documentação da linguagem **urscript** [20] na pagina 20 do documento.

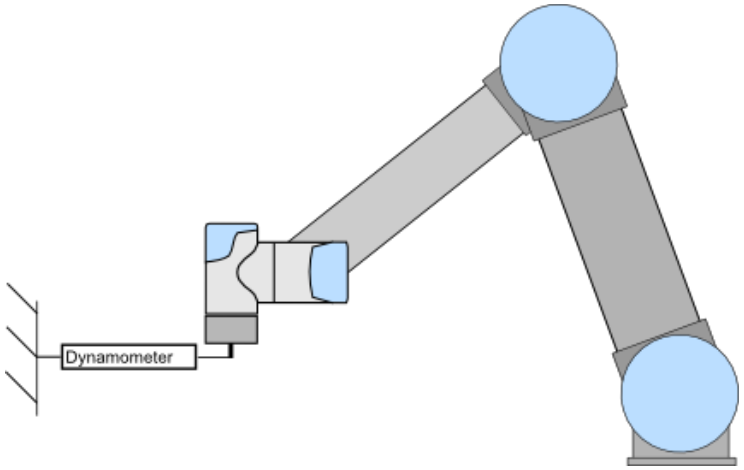


Figura 2.9: UR3 controlado via função **force_mode** (Figura retirada de [1])

## 2.2 Software

Essa seção terá o papel de esclarecer alguns pontos a respeito dos *softwares* que são usados para controlar o manipulador UR3. Além disso, a seção em questão, explicará como é a organização de arquivos para o *framework* ROS [21] usada pela Interface de Comunicação [4].

### 2.2.1 URScript API

Os robôs manipuladores da *Universal Robots* podem ser controlados de duas formas, como é demostrado pelo fabricante em seu manual [12]. A primeira forma de controlar um robô como o UR3 é usando a interface chamada de **PolyScope** (Figura 2.10) que foi desenvolvida como um produto final para uma rápida integração com linhas de produção da indústria. Essa interface possui características de aplicação de alto nível, ou seja, a interface já possui uma série de comandos que podem ser acessados de forma relativamente fácil pelo usuário final usando as setas de movimentação presentes na interface **PolyScope**.

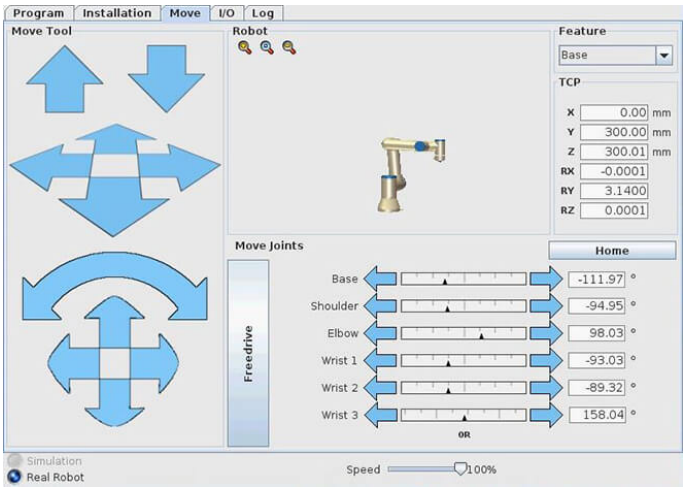


Figura 2.10: Interface PolyScope

A interface **PolyScope** também permite que possamos escrever um *script* de forma simples, sendo necessário apresentar como entrada apenas o ponto inicial e final para a interface de comando **PolyScope** utilizando um de seus métodos. Além disso, o **PolyScope** permite que possamos executar um certo tipo de ação quando o robô atinge sua trajetória final com efetuador terminal, pode ser essa ação pegar ou soltar algum objeto com sua garra.

Para mais detalhes de como usar a interface **PolyScope** para controlar a trajetória do robô UR3, é sugerido a consulta do manual da *Universal Robots* [12].

Outra forma de controlar a movimentação dos robôs da *Universal Robots*, é fazendo o uso de uma linguagem de programação desenvolvida pelo fabricante. Esta linguagem se assemelha com a linguagem de programação Python [22], o que a torna de fácil entendimento e de fácil programação. A linguagem usada para programar o robô UR3 é chamada de **URScript®** e pode ser consultada na documentação [20].

A técnica de controlar o robô via **URScript®**, permite que possamos controlar os manipuladores da *Universal Robots* por meio de um computador externo. Essa técnica tem o propósito de fazer uma exploração das propriedades dos robôs da *Universal Robots* para desenvolvimento de novas técnicas de controle e integração de múltiplos robôs manipuladores.

Na seção 2.2.3 será tratado com mais detalhe como é feito o envio e o recebimento de dados usando uma interface baseada no *framework* ROS e que faz uso da linguagem **URScript®**.

### 2.2.2 ROS

*Robotics Operating System* (ROS) é um *framework* de *software* voltado para a robótica que reúne as melhores práticas na área em conjunto com um sistema de comunicação distribuída que permite o uso de diferentes linguagens de programação no mesmo projeto. Começou em 2007 reunindo conceitos de diversos projetos de software aberto anteriores e com o passar dos anos se tornou um padrão dentro da comunidade, contando com implementação para diversos robôs comerciais e inclusive uma versão completa voltada exclusivamente para a indústria.

#### 2.2.2.1 Sistematização de arquivos no ROS

ROS é mais do que uma estrutura de desenvolvimento. Podemos nos referir a ele como um sistema operacional mais simplificado, ou seja, o ROS é dependente de outros softwares para que ocorra seu funcionamento por completo, como o próprio sistema operacional Linux. Sendo assim, ele oferece não apenas ferramentas e bibliotecas, mas também funções semelhantes a um sistema operacional, como abstração de hardware, gerenciamento de pacotes e uma cadeia de ferramentas de desenvolvedor. Como um sistema operacional real, os arquivos ROS são organizados no disco rígido de uma maneira particular, conforme ilustrado na Figura 2.11.

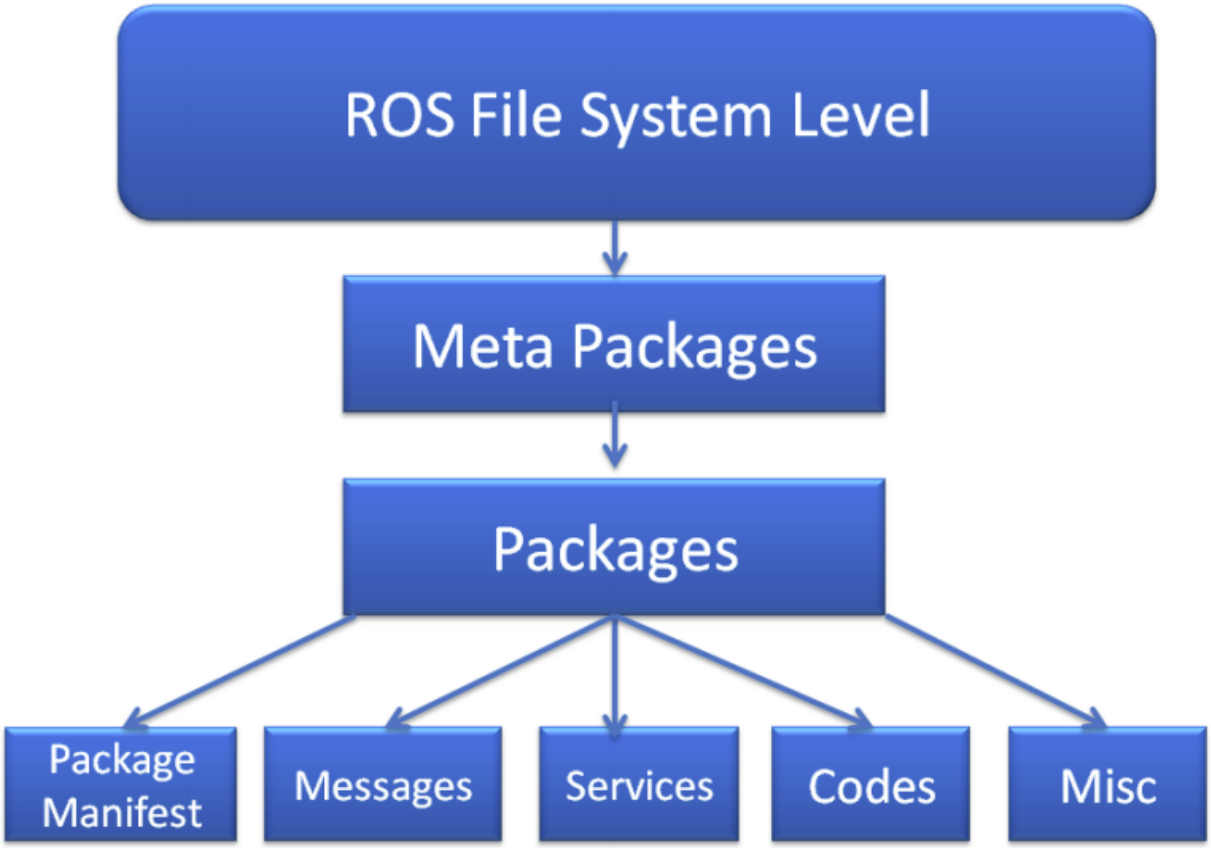


Figura 2.11: Sistematização de arquivos no ROS (extraída de *Mastering ROS for Robotics Programming* [2]).

Os itens a seguir explicam cada bloco da sistematização de arquivos no ROS mostrado na Figura 2.11 [2]:

- **Packages**: Os pacotes são as unidades mais básicas do software ROS. Eles contêm um ou mais programas ROS (nós), bibliotecas, arquivos de configuração e assim por diante, que são organizados como uma única unidade. Os pacotes são os itens de compilação "atômico" e o item de lançamento no software ROS.
- **Package manifest**: O arquivo *package manifest* está dentro de um pacote que contém informações sobre o pacote, autor, licença, dependências, sinalizadores de compilação e assim por diante. O arquivo *package.xml* dentro do pacote ROS é o arquivo *package manifest*.
- **Metapackages**: O termo *Metapackages* se refere a um ou mais pacotes relacionados que podem ser agrupados livremente. Em princípio, eles são pacotes virtuais que não contêm nenhum código-fonte ou arquivos típicos normalmente encontrados em pacotes.
- **Metapackages manifest**: O *Metapackages manifest* é semelhante ao **Package manifest**, tendo como diferença a possibilidade de incluir pacotes dentro dele como dependências de tempo de execução e declarar uma marcação de exportação.

- **Messages**(.msg): As mensagens ROS são um tipo de informação enviada de um processo ROS para o outro processo ROS. Podemos definir uma mensagem personalizada dentro da pasta *msg* dentro de um pacote *(my_package/msg/MyMessageType.msg)*. A extensão do arquivo de mensagem é *.msg*.

- **Services**(.srv): O serviço ROS é um tipo de interação de solicitação/resposta entre processos. Os tipos de dados de resposta e solicitação podem ser definidos dentro da pasta srv dentro do pacote *(my_package/srv/MyServiceType.srv)*.

#### 2.2.2.2 Sistematização de pastas

Na subseção anterior (subseção 2.2.2.1) foi feito uma breve descrição dos principais arquivos para o funcionamento de um pacote ROS. Agora, trataremos de explicar como é a organização das pastas de um pacote ROS para um bom gerenciamento de projeto em robótica usando o *framework* ROS. A Figura 2.11 mostra, o pacote **ur3** para o robô UR3 presente no LARA, a disposição das pastas dentro de um pacote ROS e como elas devem ser nomeadas segundo [2].

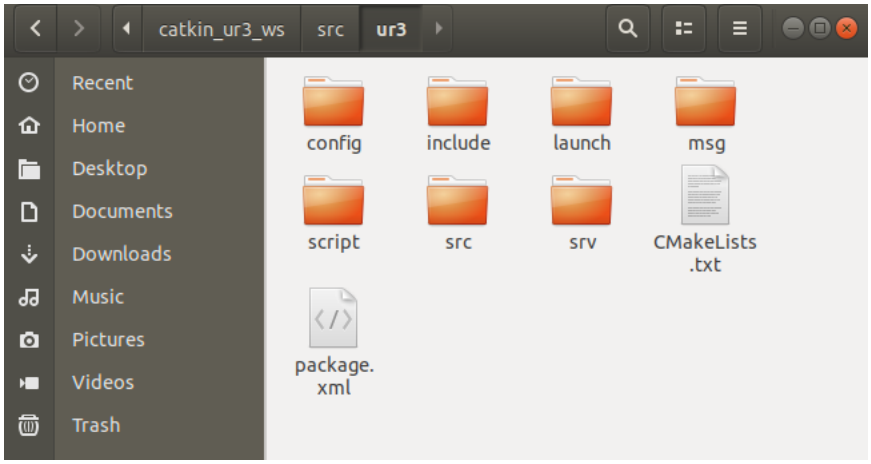


Figura 2.12: Sistema de pastas ROS no Ubuntu

- **config**: Na pasta config devem estar todos os arquivos de configuração para o *setup*. Geralmente, esses arquivos estão no formato **yaml**, que são arquivos descritivos de um dicionário em que pode-se, por exemplo, informar qual controlador para o robô deve ser usado, limitação de angulo das juntos, velocidade máxima das juntas, etc.

- **include**: A pasta include tem a finalidade de armazenar todas as bibliotecas criadas pelo desenvolvedor do pacote. Como o ROS, atualmente, só pode ser programado em *python* ou *C++*, as bibliotecas criadas terão o formato **.hpp** em que podem ser implementadas classes e métodos [23].

- **launch**: A pasta launch tem a função de armazenar os arquivos de lançamento (exemplo.launch) dos pacotes criados pelo desenvolvedor ou pacotes de terceiros. Além disso, os arquivos de lançamento tem a função, no momento do lançamento dos pacotes ROS, de passar os arquivos de configuração (exemplo.yaml) como parâmetro para o pacote ROS.

- **msg**: A pasta **msg** tem a função de armazenar todos os arquivos de mensagens com formato **.msg** criados pelo desenvolvedor. A existência de arquivo **.msg** se dá pelo fato de, em algumas ocasiões, as mensagens nativas do ROS não fornecerem suporte para formato adequado dependendo do projeto que se quer desenvolver.

- **srv**: A pasta **srv**, a pasta **srv** tem a finalidade de armazenar aquivos de mensagens, porém, essas mensagens possuem a característica de gerar um retorno quando são acionadas. Esses retornos podem ser um mensagem de sucesso ou falha, pequenos relatórios da execução do serviço, etc.

- **script**: Como o ROS, atualmente, só possui a linguagem Python como linguagem do padrão *script*, a pasta **script** tem a função de armazenar todos os aquivos de programação implementado nesta linguagem.

- **src**: A pasta **src**, a pasta **src** tem a função de armazenar arquivos de programação, porém ela só armazena aquivos implementados na linguagem *C++*.

#### 2.2.2.3 Rosbag

Rosbag [24] é um utilitário do ROS utilizado para registrar os eventos em cada tópico. Sua principal característica é que a informação pode ser lida e depois reproduzida como tópico sem a necessidade de manter o hardware conectado, permitindo a análise de diferentes experimentos diretamente pelas ferramentas do ROS.

### 2.2.3 Interface de Comunicação

Uma vez que a interface **PolyScope** e a linguagem **URScript®** fornecidos pelo fabricante não permitem acesso/manipulação de alguns sinais necessários para se poder implementar até mesmo controladores de robô usuais, fez-se necessário o desenvolvimento de uma interface de comunicação própria. Desta forma, para o controle do robô UR3, é utilizado um computador externo (vide Figura 2.15) com o sistema operacional **Linux** com a versão Ubuntu 18.04. Neste computador roda a Interface de Comunicação proposta que tem o fluxo de dados como ilustrado na Figura 2.13. A Interface de Comunicação proposta é baseada em [25] e pode ser consultada no trabalho de iniciação científica [4].

A Interface de Comunicação para o robô é acionada via linha de comando no **Terminator**[1] do Ubuntu 18.04 (vide Figura 2.14). Assim, para fazer o carregamento da Interface de Comunicação e fazer seu uso, é necessário inserir comandos típicos do ROS no **Terminator** do Linux (Figura 2.14). O Apêndice A contém informações de como fazer a inicialização da Interface de Comunicação para realizar um experimento completo e, também, as instruções de como iniciar todos os módulos necessários para que a comunicação entre o Linux PC e o robô UR3 ocorra da forma correta.

[1] https://terminator-gtk3.readthedocs.io/en/latest/

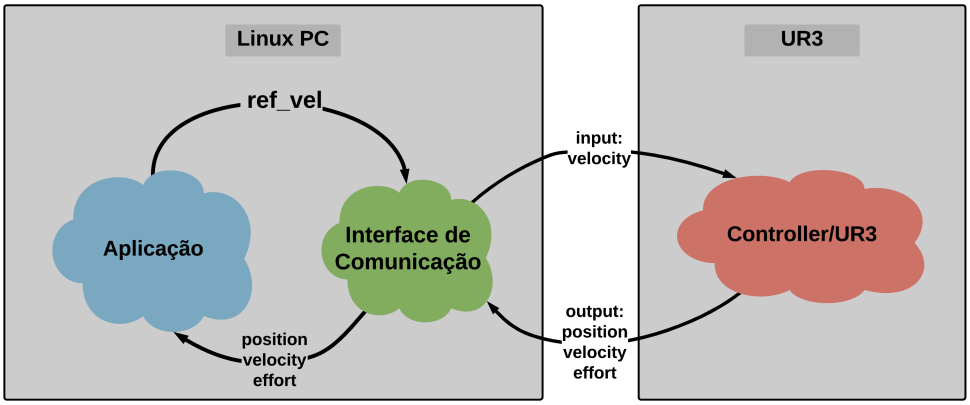


Figura 2.13: Ilustração da relação entre Linux PC e o UR3 no diagrama de comunicação

```
ur3@ur3 ~/catkin_ur3_ws main ± roslaunch ur3 real_ur3.launch
... logging to /home/ur3/.ros/log/1bbf59f2-dfe8-11eb-9369-bcaec59c8518/roslaunch-ur3-9883.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ur3:40909/

SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.11
 * /ur3/max_ref_vel/joint_0: 1.5
 * /ur3/max_ref_vel/joint_1: 1.5
 * /ur3/max_ref_vel/joint_2: 1.5
 * /ur3/max_ref_vel/joint_3: 1.5
 * /ur3/max_ref_vel/joint_4: 1.5
 * /ur3/max_ref_vel/joint_5: 1.5
 * /ur3/max_torque/joint_0: 2.0
 * /ur3/max_torque/joint_1: 2.0
 * /ur3/max_torque/joint_2: 2.0
 * /ur3/max_torque/joint_3: 2.0
 * /ur3/max_torque/joint_4: 2.0
 * /ur3/max_torque/joint_5: 2.0
 * /ur3/robot_ip: 192.168.1.56
 * /ur3/robot_port: 30003

NODES
  /
    ur3 (ur3/interface)

auto-starting new master
process[master]: started with pid [9893]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 1bbf59f2-dfe8-11eb-9369-bcaec59c8518
process[rosout-1]: started with pid [9904]
started core service [/rosout]
process[ur3-2]: started with pid [9908]
[ WARN] [1625747356.650268452]: Init Interface ur3
[ WARN] [1625747356.660381457]: Wainting for ur3 response ...
[ WARN] [1625747356.662069329]: Send script to ur3 ...
[ WARN] [1625747356.686464777]: ur3 is setted in ip 192.168.1.56
[ WARN] [1625747356.686540369]: ur3 is setted in port 30003
[ UR3 ] [1625747362.890395000]: ur3 node is running ;)
```

Figura 2.14: Carregamento da interface usando o Terminal do Linux

#### 2.2.3.1 Aplicação

A Aplicação, mostrada na Figura 2.13 com a cor azul, consiste em um nó em ROS que pode ser apenas um algoritmo que faz a interconexão com outros robôs/computadores que usam o ROS ou pode até ser uma aplicação que tem um propósito voltado para algo mais específico, por exemplo, um controlador de posição para cada junta do UR3.

#### 2.2.3.2 Descrição das entradas e saídas do sistema

A Interface de Comunicação nada mais é do que um nó ROS, escrito em *C++*, que visa o desacoplamento entre a aplicação e o robô em si (Figura 2.13) e tem a finalidade de funcionar como um intermediário entre uma aplicação ROS, que pode ser escrita em *Python* ou *C++*, e o UR3. A interface possui um tópico de subscrição chamado de **/ur3/ref_vel** por onde recebe as referências de velocidade que serão passadas para cada junto do UR3.

Sendo a Interface de Comunicação uma aplicação que faz o uso do *framewrk* ROS, ela possui algumas entradas e saídas de dados que serão abordadas nessa subseção. Nessa descrição, focaremos nos tipos de mensagens usados na Interface de Comunicação com o objetivo de destacar e de onde as mensagens vem e para onde as mensagens vão tentando não deixar margem para interpretação do leitor acerca de como é o funcionamento correto.

Primeiro vamos descrever o tópico que recebe as mensagens de referência de velocidade para as juntas do UR3 que têm origem a partir de uma determinada aplicação que faz uso do ROS.

O nome do tópico que recebe as mensagens de referência de velocidade na Interface de Comunicação é **/ur3/ref_vel**. Esse tópico é do tipo *subscriber* e, segundo a arquitetura do ROS, só é permitido escrever nele. Além disso, esse tópico possui um tipo específico de mensagem ROS, **std_msgs/Float64MultiArray Message**, que é um vetor de tamanho 6, ou seja, nesse vetor exite 6 campos e cada um armazena, em rad/s seguindo o padrão do fabricante [20], o dado de referência de velocidade para uma determinada junta, sendo que, o primeiro campo corresponde ao dado para junta da base do robô e o ultimo campo corresponde ao dado para sexta junta do robô.

Tendo em mente o que foi exposto no parágrafo acima, se o leitor deseja implementar um aplicação ROS em *C++, Python ou Matlab* que forneça referencias de velocidade para o robô UR3, deve seguir o padrão de mensagem ROS explicado.

O segundo tema que iremos tratar será a respeito das saídas de dados que representam o estado das juntas do UR3 que até o momento, para cada junta, corresponde a posição (em radianos), velocidade (em rad/s) e torque (em N.m). Essa unidades escolhidas para os estados das juntas do UR3 na Interface de Comunicação tem como base a documentação do fabricante [20].

Assim que a Interface de Comunicação fica de posse das mensagens dos estados das juntas do robô, é feito o encaminhamento para camada de aplicação ROS usando a mensagem ROS do tipo **sensor_msgs/JointState Message** e o tópico portador dessa mensagem se chama **/ur3/arm**. Com este tópico em perfeito funcionamento, os dados das juntas do robô UR3 fica público para

qualquer outra aplicação ROS fazer uso, tais como:

- Gravar os dados em rosbag.bag
- Enviar esses dados para outros robôs conectados a mesma rede.
- Usar os dados em controlador em uma malha mais externa.
- etc

## 2.3 A plataforma de desenvolvimento

Nesta seção, fazemos uma apresentação do ponto de vista do usuário da plataforma de desenvolvimento. A Figura 2.15, mostra, com ajudas de legendas, cada componente de hardware da plataforma para desenvolvimento que foram descritos no capitulo 2. Os códigos usado na plataforma de desenvolvimento podem ser encontrados no endereço do **github** `https://github.com/lara-unb/catkin_ur3_ws`.

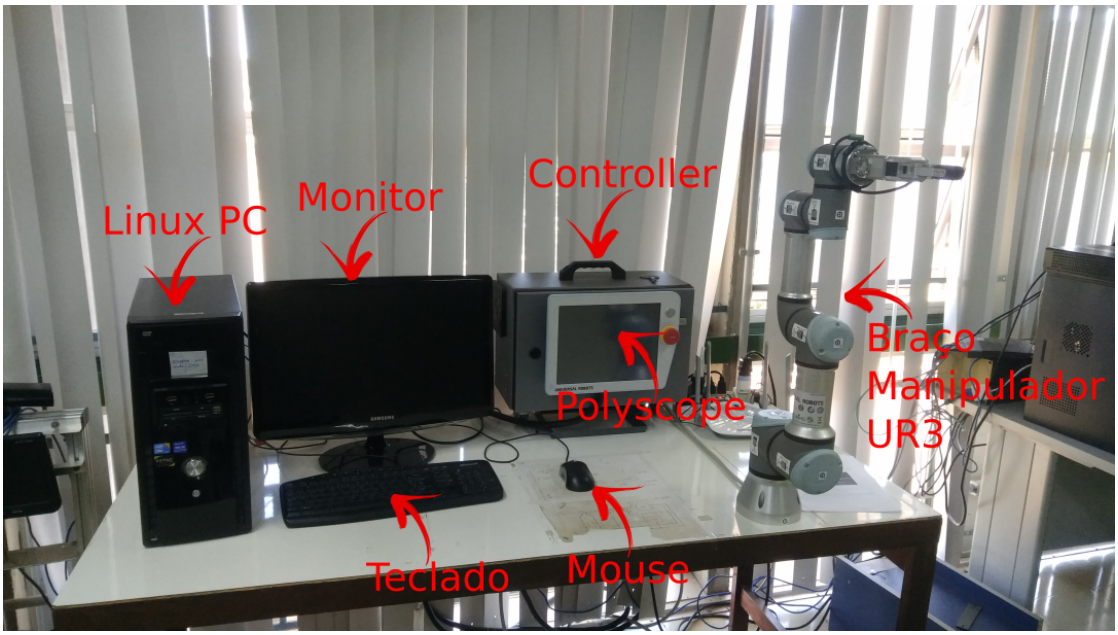


Figura 2.15: Plataforma para desenvolvimento de pesquisas baseada no robô manipulador UR3

Como um dos objetivos da plataforma de interface homem-máquina é possibilitar uma rápida introdução ao uso/controle do UR3 para novos usuários/alunos de graduação, a descrição do sistema foi organizada na forma de experimentos didáticos como segue:

- Apêndice A Experimento 1: Ligar e desligar a plataforma de desenvolvimento
- Apêndice B Experimento 2: Usando controlador do Capítulo 4.
- Apêndice C Experimento 3: Visualização dos dados dos estados das juntas, sinais de entrada e gravação dos dados.

- Apêndice D Experimento 4: Como construir um controlador Avanço Atraso usando python.

A seguir apresentamos um breve resumo dos experimentos. Os experimentos completos se encontram nos apêndices de A até D.

### 2.3.1 Experimento 1: Ligar e desligar a plataforma de desenvolvimento

Uma grande dificuldade para novos usuários que querem trabalhar com pesquisa no UR3 é a simples tarefa de fazer a inicialização do sistema e deixar pronto para uso com a Interface de Comunicação 2.2.3. Para resolver esse problema, foi escrita uma documentação (Apêndice A) que guia o novo usuário com o passo a passo de como preparar a plataforma de desenvolvimento construída nesse trabalho de graduação.

A documentação do Apêndice A tem como objetivo fornecer um primeiro contato do usuário com o sistema robótico do manipulador UR3. Ele permitirá que o usuário seja capaz de:

- realizar o processo de inicialização do robô manipulador UR3;
- realizar o processo de inicialização do Linux PC;
- rodar um experimento simples;
- realizar o processo de desligamento do sistema.

### 2.3.2 Experimento 2: Usando controlador do Capítulo 4

#### 2.3.2.1 Gerar trajetórias usando Matlab

Afim de tornar a plataforma de desenvolvimento mais completa com relação as formas de entrada de referência de posição/velocidade (Capítulo 2 seção 2.2.3.2), foi escrito um algoritmo usando a linguagem matlab para gerar ondas de referência para UR3. Esse código é mais detalhando no Apêndice B, onde é feito um experimento com o passo a passo de como gerar as ondas de referência, como mover os arquivos com as ondas para a pasta que a aplicação (Capítulo 2 seção 2.2.3.1) possa usar e configurações em geral.

#### 2.3.2.2 Usando controlador

Neste experimento o usuário tem a opção de enviar ondas de referência de posição de forma offline, como é feito no experimento 1 mostrado no Apêndice A, ou mandar o comando especificando a posição da junta via terminal.

### 2.3.3 Experimento 3: Visualização dos dados dos estados das juntas, sinais de entrada e gravação dos dados

Pensando na possibilidade de trabalhar com os dados experimentais gerados durante o funcionamento do UR3, foi desenvolvido uma forma mais automatizada de fazer coletadas de dados durante os experimentos na plataforma de desenvolvimento proposta nesse nesse trabalho.

Uma das forma de fazer a coleta ou visualização de dados em tempo real é usando as ferramentas do ROS (como demonstrado no Apêndice C) para se conectar a Interface de Comunicação 2.2.3 e depois usar o matlab para extrair os dados compactados no formato rosbag. Outra forma de fazer coleta de dados é usando a interface **ur3_contrls** clicando na opção **Record_Bag** destacado com o retângulo vermelho na Figura 2.16. A interface **ur3_contrls** foi construída usando o pacote ROS **Dynamic Reconfigure** [26] e a forma de execução da mesma pode ser encontrado no Apêndice D.

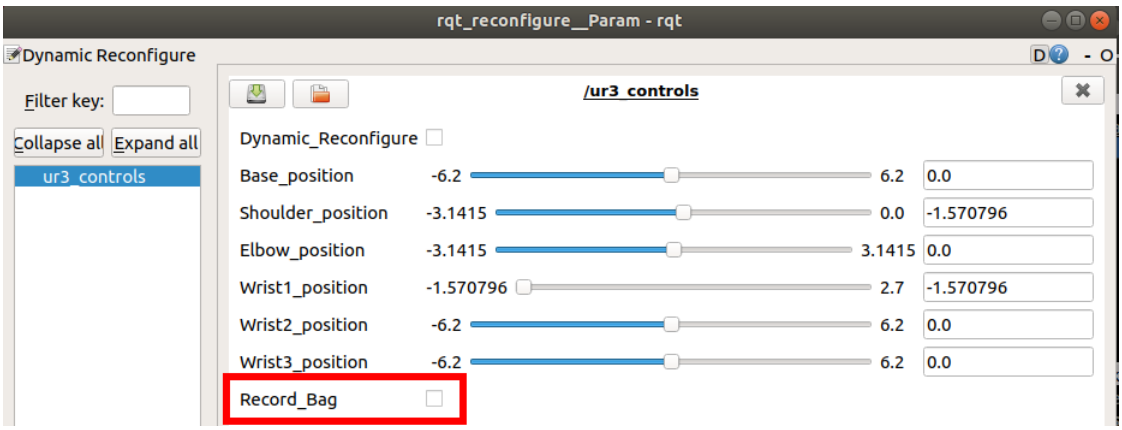


Figura 2.16: Interface ur3_contros

### 2.3.4 Experimento 3: Adição de controladores na plataforma de desenvolvimento

Tendo em vista que o LARA possui muitos pesquisadores com diferentes níveis de conhecimento técnico, foi construído um pacote ROS chamado **ur3_controls** para ajudar os iniciantes na plataforma proposta nesse projeto de graduação a codificar seu próprio controlador personalizado usando a linguagem de programação Python e poder inseri-lo da forma mais simples possível.

A figura 2.17 mostra como o pacote **ur3_controls** assume a posição no diagrama de comunicação, que na Figura 2.13 era chamado de Aplicação. O bloco **ur3_controls** foi pensado como um pacote que possibilita o desacoplamento do *framework* ROS dos controladores que serão implementados, por isso a representação do bloco **meu controlador**, em rosa na Figura 2.17, está fora do fundo roxo.

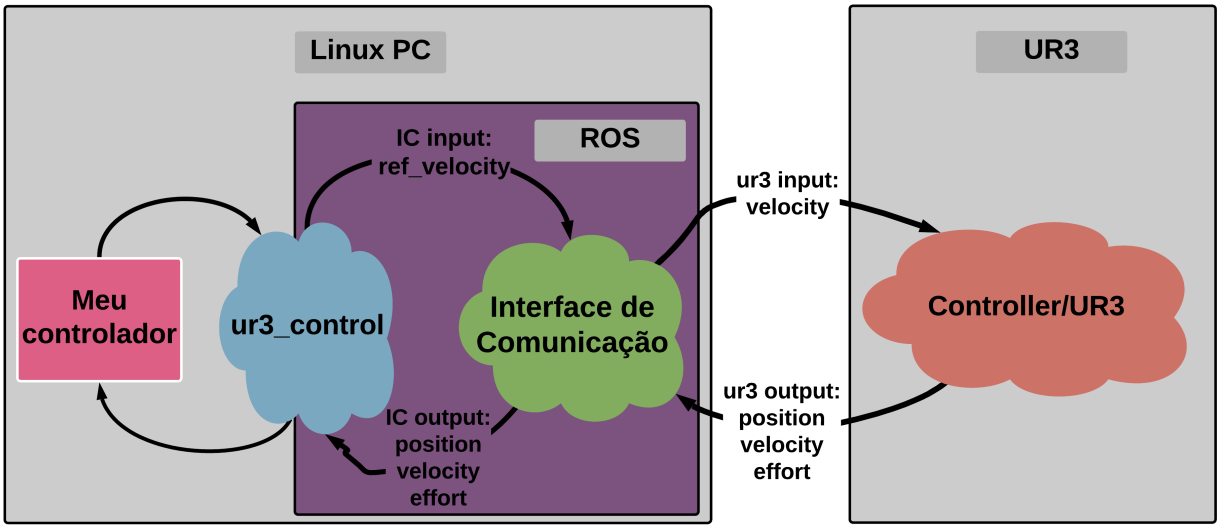


Figura 2.17: Adição de controlador usando o pacote **ur3_controls**

A descrição dos nós, tópicos e serviços da camada ROS (em roxo na Figura 2.17) que envolve a Interface de Comunicação e o **ur3_control** é apresentada a seguir:

- Interface de Comunicação
  - nós
    - ur3
  - tópicos
    - ur3/arm
      - função: publicar as posições, velocidade e torque das juntas do UR3
      - tipo: publisher
      - mensagem ROS: sensor_msgs/JointState
        - posição em radiano
        - velocidade em rad/s
        - Torque em N.m
      - frequência: 125 hz
    - ur3/ref_vel
      - função: subscrever os sinais de velocidade de uma aplicação externa e enviar para o UR3
      - tipo: subscriber
      - mensagem ROS: std_msgs/Float64MultiArray
        - velocidade em rad/s
      - frequência: 125 hz
  - serviços

        * ur3/reset
            · função: reiniciar a Interface de Comunicação e colocar o UR3 na posição de descanso
            · mensagem ROS: std_srvs/Trigger

- ur3_control
    - nós
        * ur3_control
    - tópicos
        * ur3/ref_vel
            · função: publicar os sinais de velocidade de referência para a Interface de Comunicação
            · tipo: publisher
            · mensagem ROS: std_msgs/Float64MultiArray
                · velocidade em rad/s
            · frequência: 125 hz
        * ur3/arm
            · função: subscrever as posições, velocidade e torque das juntas enviadas pela Interface de Comunicação
            · tipo: subscriber
            · mensagem ROS: sensor_msgs/JointState
                · posição em radiano
                · velocidade em rad/s
                · Torque em N.m
            · frequência: 125 hz
    - serviços
        * ur3_control/start_control
            · função: habilitar ou desabilitar o controlador (**True** habilita e **False** desabilita)
            · mensagem ROS: std_srvs/SetBool

Caso queria se aprofundar no pacote **ur3_control**, consulte o Apêndice D.

## 2.4 Aplicações da plataforma

Os experimentos listados nos apêndices objetivam servir como primeiro contato a funcionalidades do sistema. Para demonstrar a utilidade da plataforma desenvolvida, foram realizados experimentos mais complexos como obtenção de parâmetros de Denavit-Hartenberg e projeto de controle de posição de uma junta. Estas aplicações estão descritas nos Capítulos 3 e 4.

# Capítulo 3

# Aplicação 1: Parâmetros de Denavit-Hartenberg para o UR3

## 3.1 Parâmetros Denavit-Hartenberg (DH)

Uma das formas de se determinar o comportamento cinemático de robôs manipulador é utilizando a matriz de Denavit-Hartenberg (DH) do robô. Essa matriz é escrita a partir de quatro parâmetros ($\theta_i$, $d_i$, $a_i$ e $a_i$) (vide Figura 3.1) para cada junta que descrevem a relação entre os links de um robô. Dois desses parâmetros descrevem o link em si ($a_i$, $d_i$) e dois explicam como cada junta se conecta ($\theta_i$ e $\alpha_i$ ). Uma característica desse procedimento é que três desses parâmetros são constantes para dado robô, pois dependem de como ele foi construído enquanto o último parâmetro depende apenas do ângulo de rotação que uma junta realizou.

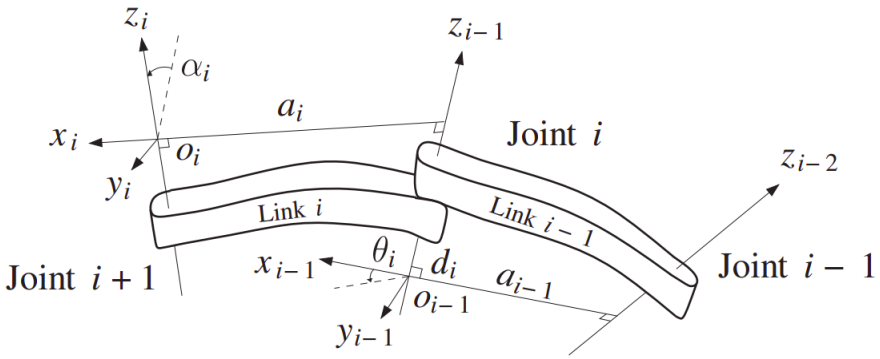


Figura 3.1: Atribuição de estrutura Denavit – Hartenberg extraída de [3].

- $\theta_i$: distância ao longo de $x_i$ da interseção dos eixos $x_i$ e $z_{i-1}$ para $o_i$.
- $d_i$: distância ao longo de $z_{i-1}$ de $o_{i-1}$ até a interseção dos eixos $o_i$ e $z_{i-1}$. Se a articulação i for prismática, $d_i$ é variável.
- $a_i$: o ângulo de $z_{i-1}$ a $z_i$ medido em torno de $x_i$.

- $\alpha_i$: o ângulo de $x_{i-1}$ a $x_i$ medido em torno de $z_{i-1}$. Se a junta i é a rotação, $\alpha_i$ é variável.

A metodologia adotada para encontrar os parâmetros listado acima para o UR3 (parâmetros Denavit-Hartenberg mostrada acima) foi baseada na notação de Denavit-Hartenberg [27]. Essa notação permite que se descreva todos os movimentos de um manipulador robótico, a partir dos movimento gerados pela juntas, por meio de um eixo referencial, posicionado na base do robô (Modelo Cinemático Direto). Nesta seção, será feito um estudo afim de estabelecer os parâmetros de Denavit-Hartenberg (que também será mencionada como DH a partir de agora) segundo a notação aborda no livros texto **Robot Modeling and Control - Mark W Spong**[3] e que serão consolidados em uma tabela para o UR3.

## 3.2 Marcações dos eixos de coordenadas e paramentos DH segundo fabricante

As marcações dos eixos de coordenas mostradas na Figura 3.2, que vão da Base até o efetuador terminal (End Effector), foram extraídas de um documento do próprio fabricante **Universal Robots** [28]. Nesse trabalho de graduação, as marcações dos eixos de coordenada extraídas da documentação do fabricante tem o objetivo de fornecer uma primeira informação a respeito das orientações de cada junta para que seja feito um estudo de validação das informações e um trabalho de gerar parâmetros DH para as convenções estabelecidas em [3].

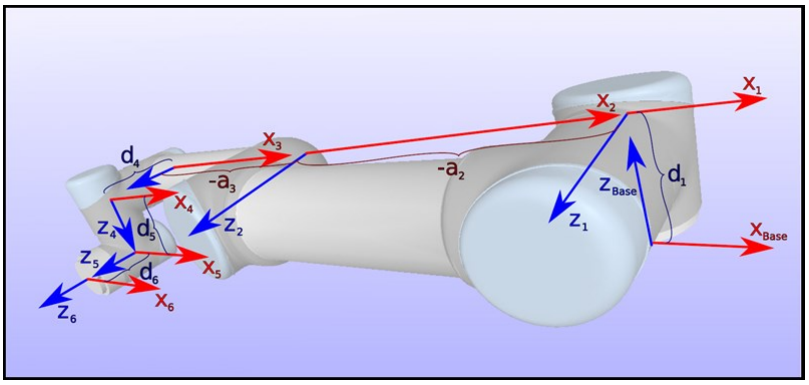


Figura 3.2: Posicionamentos dos eixos de coordenada para junta do UR3.

Os resultados apresentados pelo fabricante como sendo os paramentos DH para o robô manipulador UR3 estão consolidados na Tabela. 3.1.

| Parâmetros Denavit-Hartenberg | | | | |
|---|---|---|---|---|
| **Cinemática** | $\theta[rad]$ | $a$ [m] | $d$ [m] | $\alpha[rad]$ |
| Joint 1 | 0 | 0 | 0.1519 | $\pi/2$ |
| Joint 2 | 0 | -0.24365 | 0 | 0 |
| Joint 3 | 0 | -0.21325 | 0 | 0 |
| Joint 4 | 0 | 0 | 0.11235 | 0 |
| Joint 5 | 0 | 0 | 0.08535 | $-\pi/2$ |
| Joint 6 | 0 | 0 | 0.0819 | $\pi/2$ |

Tabela 3.1: Parâmtros de Denavit-Hartenberg segundo o fabricante

## 3.3 Marcação dos eixos de coordenadas do UR3

1. Usando as setas da Interface do **Polyscope**, o manipulador UR3 foi colocado na mesma posição do esquemático da Figura 3.2 (veja a Figura 3.3).

2. Depois, usando interface a tela do **Polyscope** (veja Figura 2.10), foram feitos movimentos de rotação no sentido positivo indicado pelas setas de movimentação para cada junta. Com isso, o sentido do eixo **z** de cada uma das juntas zfoi encontrado. Pode-se verificar que a localização, direção e sentido de todos eixos **z** correspondem aos apresentados na Figura 3.2. Cada eixo **z** passa pelo centro do eixo do rotor da junta correspondente [1].

3. Em seguida, usando a visualização dos valores dos ângulos no Polyscope, foi feito um procedimento para por cada junta na posição que marcava 0 (zero) radiano (Como exemplifica a Figura 3.4). Com isso, foi escolhido o mesmo sentido do eixo **x** com o vetor que aponta para a marcação de 0 (zero) radiano (vide Figura 3.5).

4. O eixo **y**, para cada junta, surgiu fazendo uso Regra da Mão Direita tendo como base os eixos **x** e **z** já consolidados nos itens acima.

Depois de realizar os procedimentos listados nos itens acima, os dados gerados foram consolidados na Figura 3.6. Com os resultados da Figura 3.6 foi feito um trabalho manual de marcações dos eixos de coordenada para cada junta e marcações para identificação da posição de cada motor nas juntas do UR3 como mostra a Figura 3.7, que pode ser visto com mais detalhes na Figura 3.8.

[1] verificar harmonic drive


Figura 3.3: UR3 real posicionado como na Figura 3.2

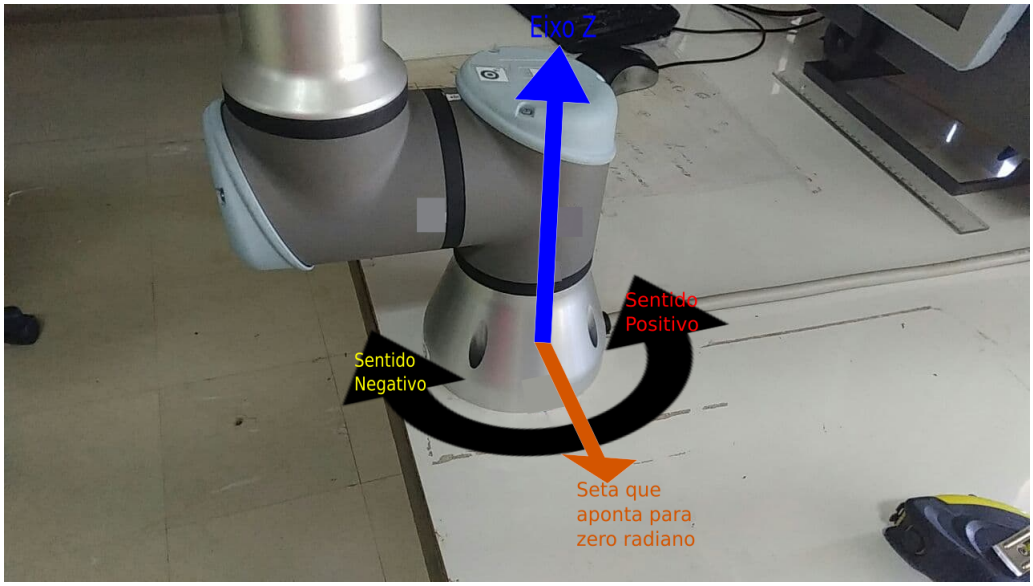


Figura 3.4: Procedimento para escolha do eixo X

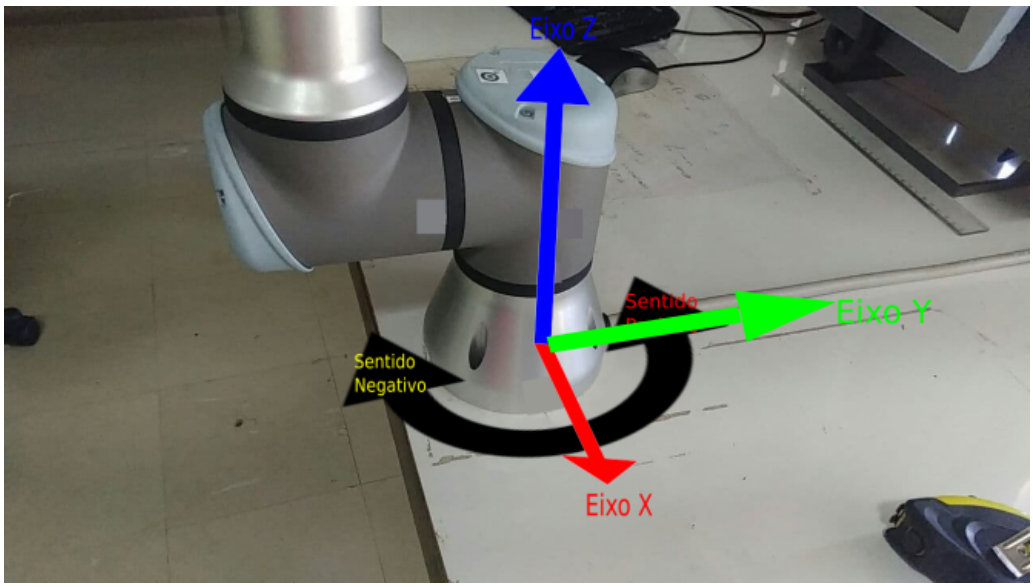


Figura 3.5: Eixos Z, Y e Z

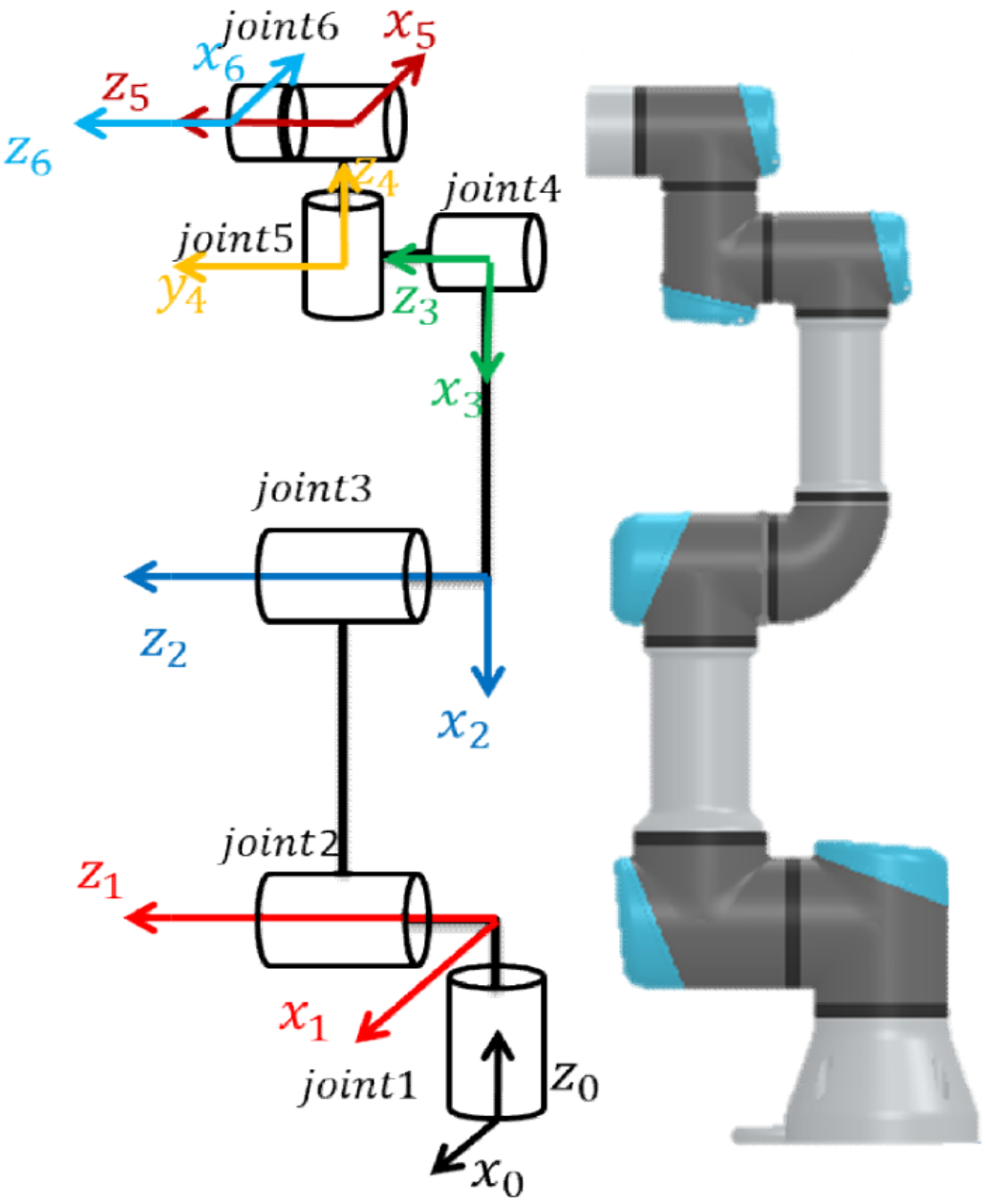


Figura 3.6: Marcação dos eixo das juntas para o UR3 em um esquemático.

**Marcação dos Motores** **Marcação dos Eixos**

Figura 3.7: Marcações dos motores e dos Eixos das justas do UR3

(a) Base

(b) Joint 1

(c) Joint 2

(d) Joint 3

(e) Joint 4

(f) Joint 5

(g) Joint 6

(h) Effector

(i) End Effector from view

Figura 3.8: Marcações da juntas do UR3 com mais detalhes

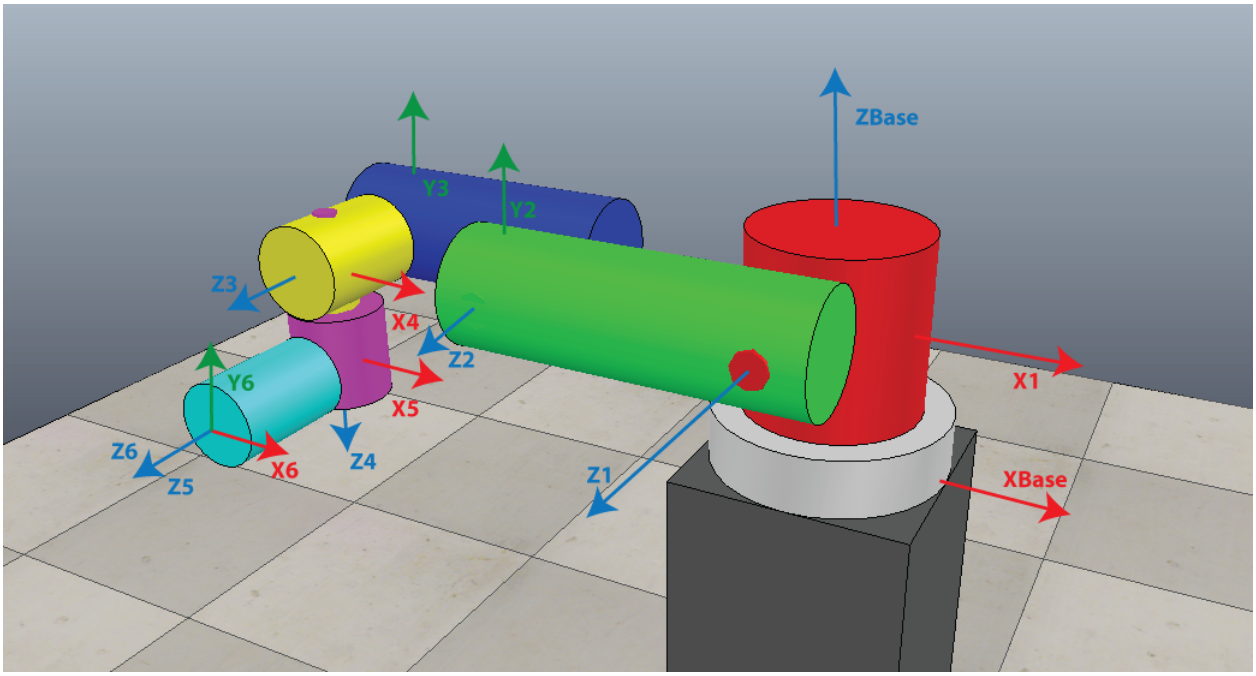


Figura 3.9: Ilustração 3D da orientação de cada articulação do robô UR3

## 3.4 Verificação dos paramentos DH experimentalmente

Para a verificação dos parâmetros DH do manipulador UR3 será realizado um experimento que envolve a movimentação individual de cada junta do robô e, ao mesmo tempo, será gravando a posição do efetuador terminal no espaço tridimensional com relação a Base do UR3. De forma mais sucinta, este experimento consiste na realização dos 4 (quatro) passos a seguir:

- Definir a configuração do UR3 como na Figura 3.10 usando os dados da Tabela 3.2
- Mover cada junta separadamente até que fome um arco maior que 90°
- Estimar a circunferência gerada por cada arco
- Medir a distância entre as circunferências

Para realizar o experimento de coleta de dados, foi feito um experimento baseado nos procedimentos do Apêndice D usando a ferramenta de **Dynamic Reconfigure**. Para mais detalhes da ferramenta **Dynamic Reconfigure** consulte o documentação em [26].

| Ângulos inicial e final | | | | | | |
|---|---|---|---|---|---|---|
| | J1 | J2 | J3 | J4 | J5 | J6 |
| $\theta_{estático}$ | 0.0 | 0.0 | -90.0 | 0.0 | 90.0 | 0.0 |
| $\theta_{inicial}$ | -90.0 | -115.0 | -145.0 | -90.0 | -90.0 | -90.0 |
| $\theta_{final}$ | 90.0 | 0.0 | 0.0 | 90.0 | 90.0 | 90.0 |

Tabela 3.2: Ângulos inicial e final usados para o experimento de identificação dos parâmetros de Denavit-Hartenberg.

Figura 3.10: Posicionamento do UR3 com a configuração gerada com as posições das juntas a partir dos valores $\theta_{inicial}$ da tabela 3.2

Durante a a realização do experimento, foi posicionado uma câmera (retângulo em vermelho vide Figura 3.11) para fazer o rastreamento da posição do efetuador terminal (retângulo em amarelo Figura 3.11) para efeitos de visualização do experimento que estão consolidados no resultado da Figura 3.12.

Figura 3.11: Posicionamento de uma câmera para fazer o rastreio da posição do efetuador terminal para efeitos demostrativos

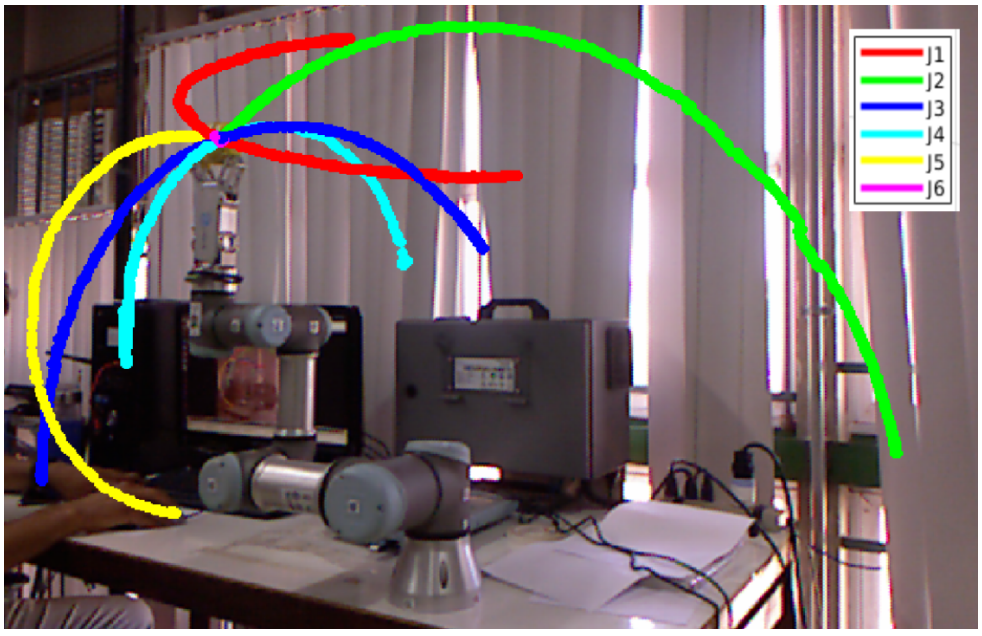


Figura 3.12: Desenho do arcos em tono do eixo z de cada junta feito por um algoritmo de visão computacional.

Para continuar com a identificação dos paramentos DH de forma experimental, foi usado a convenção, abordada no Capítulo 3, seção 3.2-**The Denavit–Hartenberg Convention** presente no livro texto **Robot Modeling and Control - Mark W Spong**[3] (Figura 3.1) para encontrar os valores de $\theta_i$ e $\alpha_i$ e definir as variáveis $d_i$ e $a_i$ nos links do UR3.

O resultado da análise proposta no parágrafo acima está ilustrado na Figura 3.14 e consolidado na tabela 3.3.

| Parâmetros Denavit-Hartenberg | | | | |
|---|---|---|---|---|
| | $\theta_i$ | $a_i$ | $d_i$ | $\alpha_i$ |
| Joint 1 | 0.0000 | 0.0000 | $d_1$ | 90.0000 |
| Joint 2 | 0.0000 | $-a_2$ | 0.0000 | 0.0000 |
| Joint 3 | 0.0000 | $-a_3$ | 0.0000 | 0.0000 |
| Joint 4 | 0.0000 | 0.0000 | $d_4$ | 90.0000 |
| Joint 5 | 0.0000 | 0.0000 | $d_5$ | -90.0000 |
| Joint 6 | 0.0000 | 0.000 | $d_6$ | 0.0000 |

Tabela 3.3: Parâmetros de Denavit-hartenberg usando a convenção de **Robot Modeling and Control - Mark W Spong**[3]

Para encontrar os valores dos parâmetros $a_i$ e $d_i$ , presentes na Tabela 3.3, foi implementado um algoritmo usando o matlab, que, dado um arco (Figura 3.13) de circunferência com N pontos x, y e z no espaço tridimensional, o algoritmo conseguiu estimar a equação da circunferência e com isso fornecer o centro da circunferência.

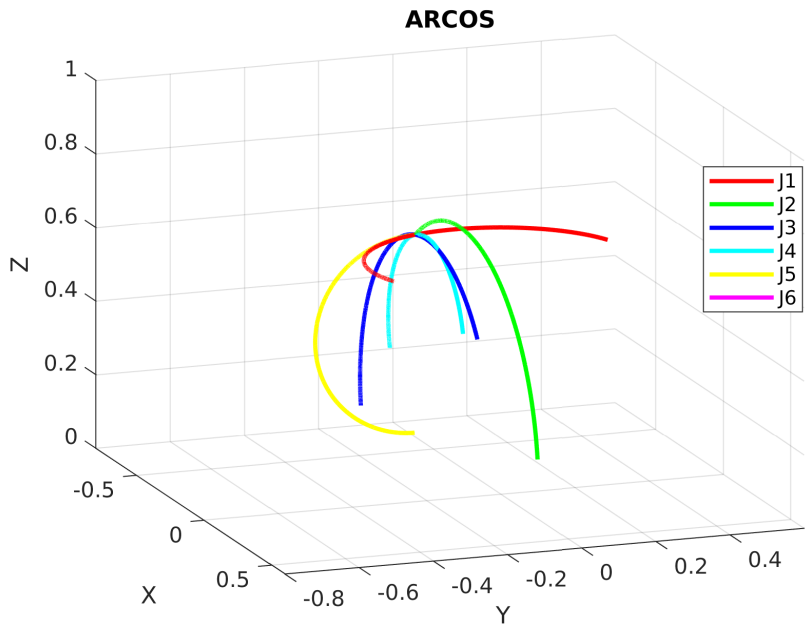


Figura 3.13: Arcos feito pelo efetuador terminal do UR3 para cada movimentação proposta pela Tabela 3.2

Os resultados do algoritmo mencionado no paragrafo acima, gera os círculos que podem ser vistos na Figura 3.15. Com esses resultados, foi feito a medida de distância entre o centro de cada círculos concoctivos, resultando nos valores de $a_2$, $a_3$, $d_1$, $d_4$, $d_5$ e $d_6$ que foram consolidados na Tabela 3.4 e comparados com os resultados do fabricante com podem ser conferidos na Tabela 3.1.

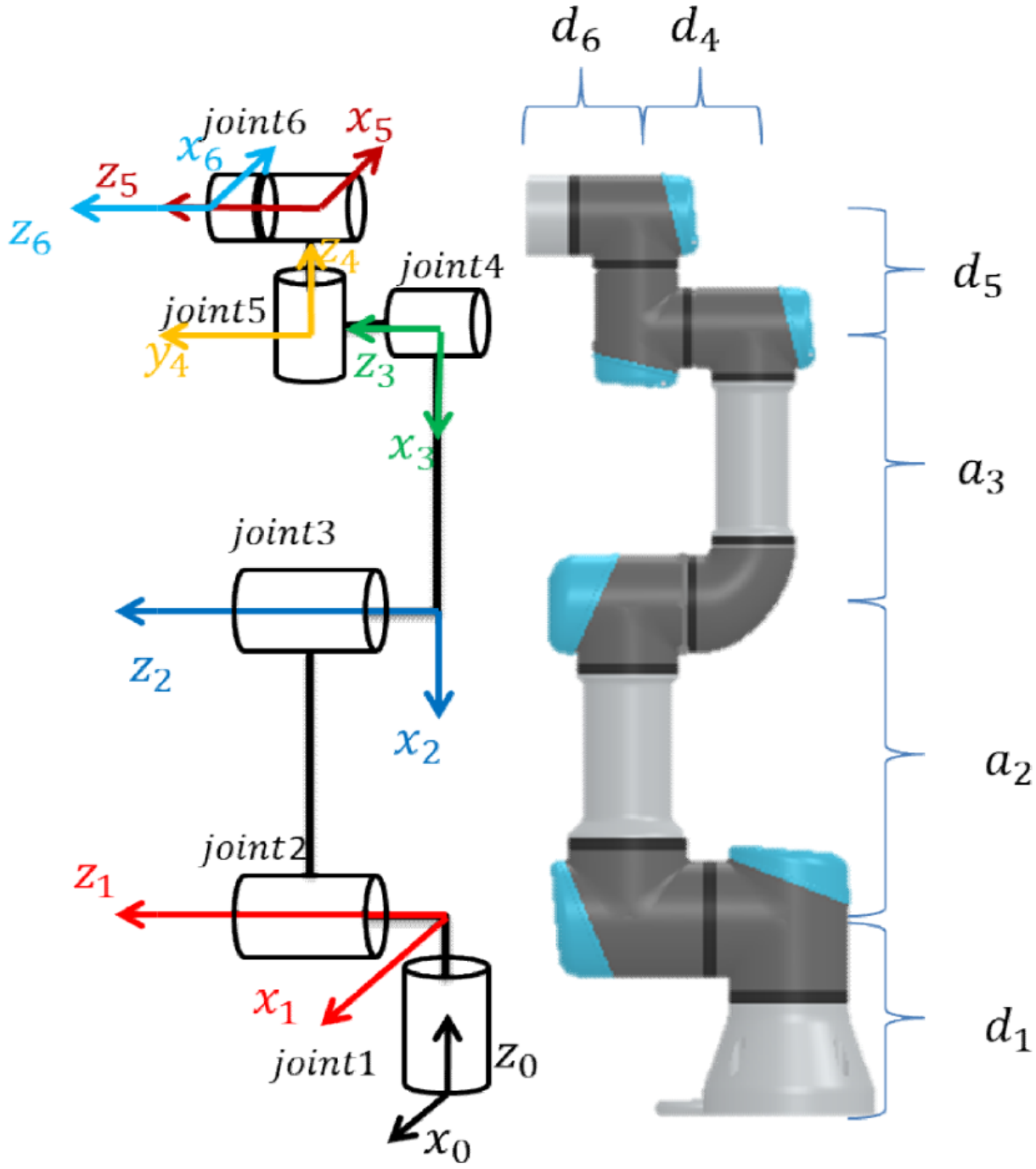


Figura 3.14: Parâmetros de Denavit-hartenberg usando a convenção de **Robot Modeling and Control - Mark W Spong**[3].

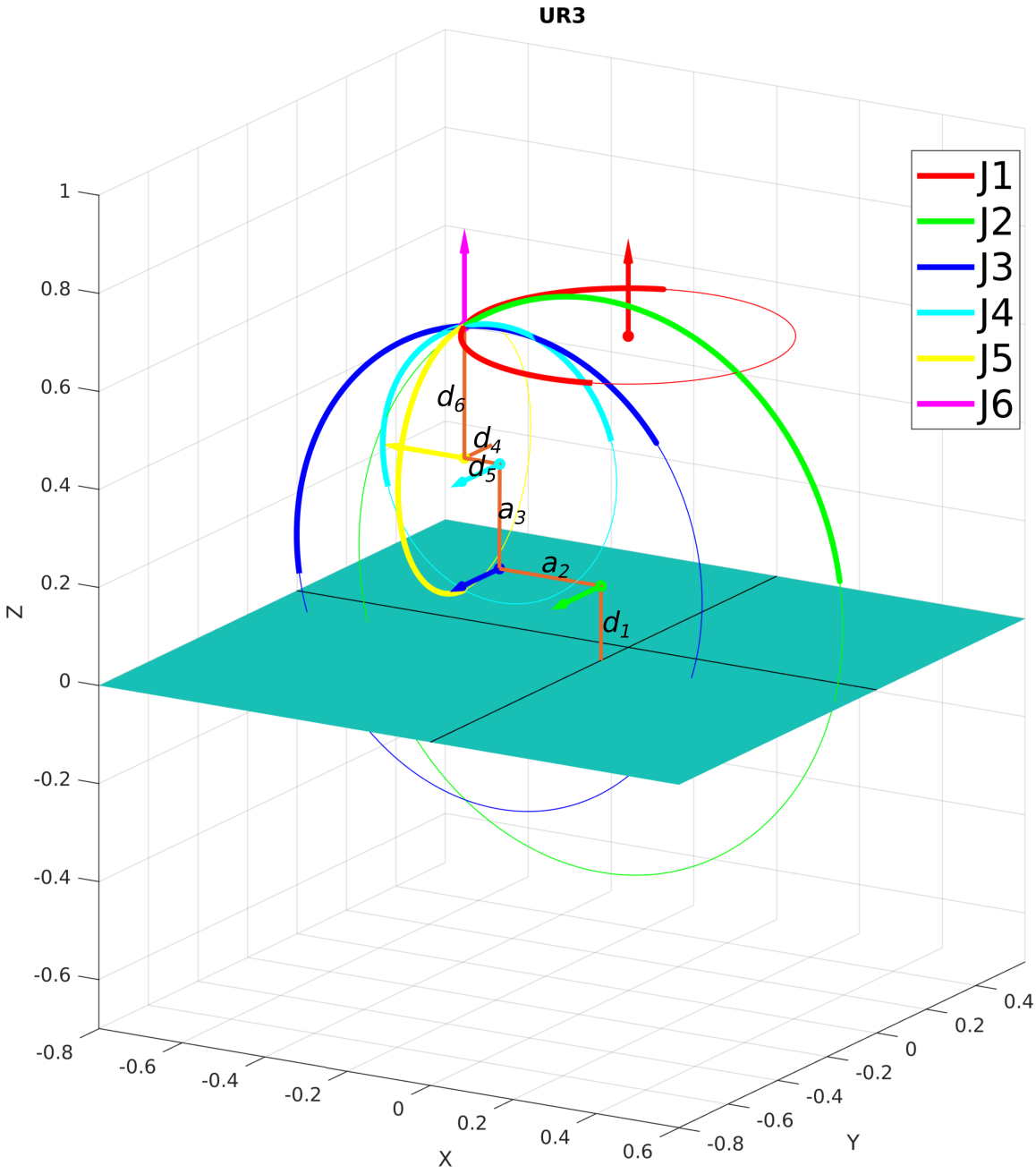


Figura 3.15: Ilustração 3D da orientação de cada articulação do robô UR3.

| Parâmetros Denavit-Hartenberg | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Joint 1 | Joint 2 | Joint 3 | Joint 4 | Joint 5 | Joint 6 | Joint 6* |
| $\theta_{i,calculado}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $\theta_{i,fabricante}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $\theta_{i,erro\%}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $a_{i,calculado}$ | 0.0000 | -243.7951 | -213.2480 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $a_{i,fabricante}$ | 0.0000 | -243.6500 | -213.2500 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $a_{i,,erro\%}$ | 0.0000 | 0.596e-02 | 9.3787e-04 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $d_{i,calculado}$ | 151.980 | 0.0000 | 0.0000 | 111.4367 | 85.3216 | 82.6000 | 269.0576 |
| $d_{i,fabricante}$ | 151.900 | 0.0000 | 0.0000 | 112.350 | 85.350 | 81.9000 | * |
| $d_{i,erro\%}$ | 5.270e-02 | 0.0000 | 0.0000 | 8.129e-01 | 3.327e-02 | 8.444e-01 | * |
| $\alpha_{i,calculado}$ | 90.0000 | 0.0000 | 0.0000 | 90.0000 | -90.0000 | 0.0000 | 0.0000 |
| $\alpha_{i,fabricante}$ | 90.0000 | 0.0000 | 0.0000 | 90.0000 | -90.0000 | 0.0000 | 0.0000 |
| $\alpha_{i,erro\%}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

Tabela 3.4: Parâmetros de DH para o UR3 encontrados experimentalmente ($\theta_i$, $a_i$, $d_i$, $\alpha_i$ ) e fornecidos fabricante ($\theta_{i,fabricante}$, $a_{i,fabricante}$, $d_{i,fabricante}$ e $\alpha_{i,fabricante}$) (veja também Tabela 3.1). Os erros percentuais para cada junta são apresentados ($\theta_{erro\%}$, $a_{erro\%}$, $d_{erro\%}$ e $\alpha_{erro\%}$). Os parâmetros a e d são representados em milímetros, $\theta$ e $\alpha$ em graus.

Como pode ser visto na Tabela 3.4 o maior erro percentual encontrado foi inferior a 1% para o parâmetro $d_6$. Outro ponto a ser mencionado é que na Tabela 3.4 apresenta dois resultados para $d_6$. A explicação para esses dois resultados é que foi encontrado um valor para $d_6$ sem a conexão do efetuador terminal (Figura 3.16) ao UR3 e outro valor para $d_6$ com o efetuador terminal conectado ao UR3.


Figura 3.16: Garra robótica RG2 da OnRobot

Como não temos nenhum valor de referência para verificarmos da veracidade do valor da medida encontrado para $d_6$ com o efetuador terminal, utilizou-se um fita métrica para medir a

distância do centro da junta 5 (Joint 5) até pinça do efetuador terminal (Figura 3.17) e concluímos que o valor do experimento foi consistente com a medição.


Figura 3.17: Verificação da distância $d_6$ usando uma fita métrica

## 3.5 Conclusão

Um experimento para verificar os parâmetros **Denavit-Hartenberg** foi concluído com sucesso fazendo o uso da plataforma de desenvolvimento. Além disso, os parâmetros DH encontrados nesse experimento são bastantes similares aos paramentos DH fornecidos pelo fabricante (pode ser consultado na Tabela 3.1 ou na referência do fabricante 3.2). Por outro lado, se comparamos os valores dos parâmetros DH encontrados nesse experimento, considerando apenas os valores de $a_2, a_3, d_1, d_4, d_5$ e $d_6$, com os valores destes parâmetros obtidos por outros trabalhos com o feito em [29], percebemos que o método usado nesse experimento resultou um desempenho superior, visto que, os erros de estimação fornecidos por [29] são maiores se comparados com os erros estimados nesse experimento.

# Capítulo 4

# Aplicação 2: Controle de posição para uma junta do robô manipulador UR3

O objetivo desse capítulo é demonstrar a funcionalidade da plataforma de desenvolvimento para ser aplicada em projetos de controladores de posição do robô. Afim de não se desviar do enfoque na funcionalidade da plataforma, neste capítulo consideramos um projeto simplificado. Consideram-se a modelagem, a identificação e o controle de posição apenas da junta Elbow joint do UR3 (conforme Figuras 4.1 e 4.2). Com respeito à modelagem e identificação, consideramos por simplicidade que a junta do UR3 tem como entrada a velocidade desejada e como saída a velocidade angular medida. Consideramos também que é possível utilizar um modelo de primeira ordem e que uma entrada senoidal, embora com frequência constante, seja suficiente para fornecer um modelo para projeto de controlador. No projeto do controlador será utilizada a técnica do lugar das raízes no plano $z$. A implementação será feita em computador (Linux PC), no bloco Meu Controlador conforme indicado na figura 2.17.

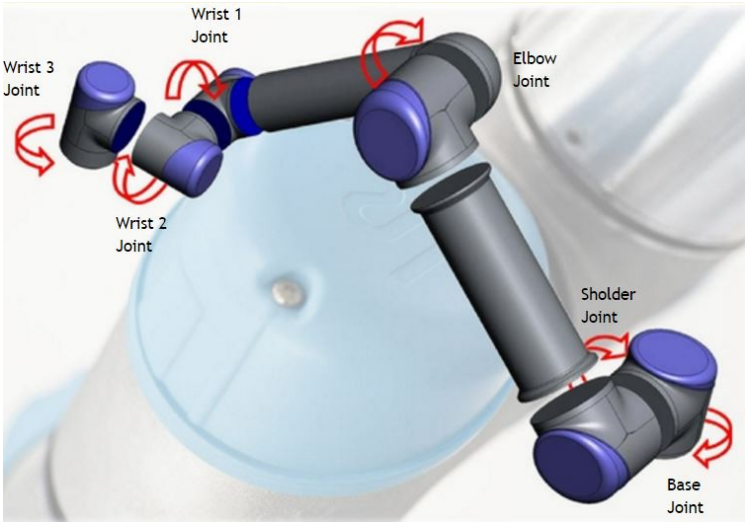


Figura 4.1: Enumeração das juntas do braço UR3


Figura 4.2: Vista real do braço robótico UR3

## 4.1 Modelagem matemática

### 4.1.1 Coleta de dados

O primeiro passo para a obtenção do modelo, consistiu em uma coleta de dados de entrada (velocidade de referência $\Omega_r(s)$ em rad/s) e saída (velocidade medida $\Omega(s)$ em rad/s) da junta em questão. Os tópicos a seguir explicam os procedimentos realizados no experimento de coleta.

#### 4.1.1.1 Preparação da referência de velocidade

Para fazer a identificação da junta **Elbow Joint** do UR3, sinalizada na Figura 4.2, se fez necessário elaborar uma referência de velocidade para junta que está sendo estudada. Utilizando o software Matlab[30], gerou-se uma onda senoidal com características que não forçassem muito as engrenagens do robô usando o tutorial do Apêndice B, evitando o seu desgaste.

Com os resultados do procedimento do Apêndice B, foi gerado uma referência de velocidade (Figura 4.3) para a junta **Elbow Joint** do manipulador com uma amplitude de 1 (um) e uma frequência de 4 rad/s.

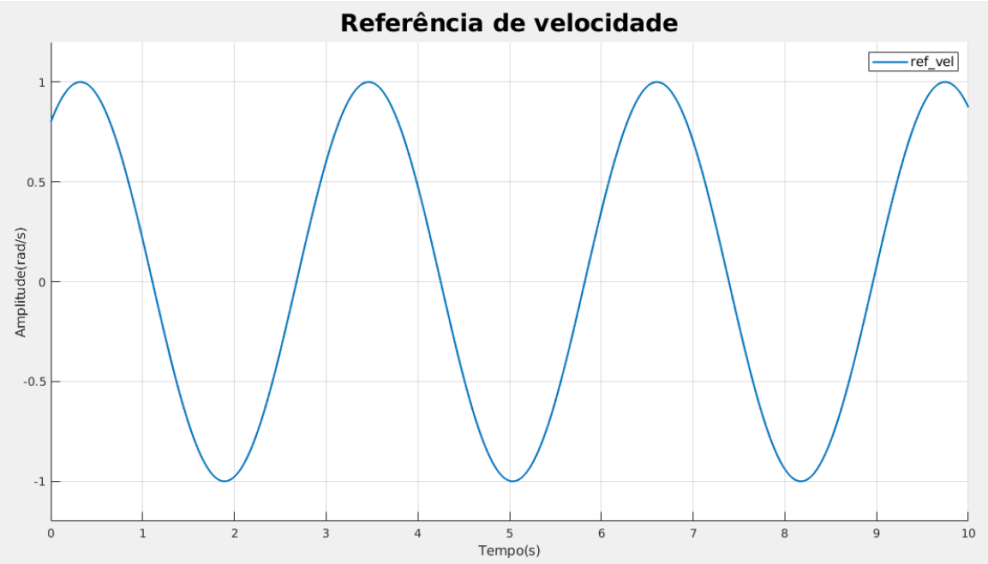


Figura 4.3: Referência de velocidade feita usando o tutorial do Apêndice B

#### 4.1.1.2 Enviando a referência de para UR3

Tendo feito o passo anterior (gerar o aquivo .csv), utilizou-se as ferramentas da Interface de comunicação para enviar as referências de velocidade para o robô.

Dentre os pacotes da Interface ROS[21], foi projetado um pacote que é específico para enviar referências de velocidade para o UR3. Para isso, basta utilizar o arquivo **.csv**, gerado no passo anterior, pondo o arquivo gerado dentro do pacote **demos_ur3** na pasta **cvc** (pasta selecionada na Figura 4.4). Para realizar todo o manuseio do robô e da Interface de Comunicação para o experimento de envio e coleta de dados, mostrado na Figura 4.5, basta seguir o Apêndice B.

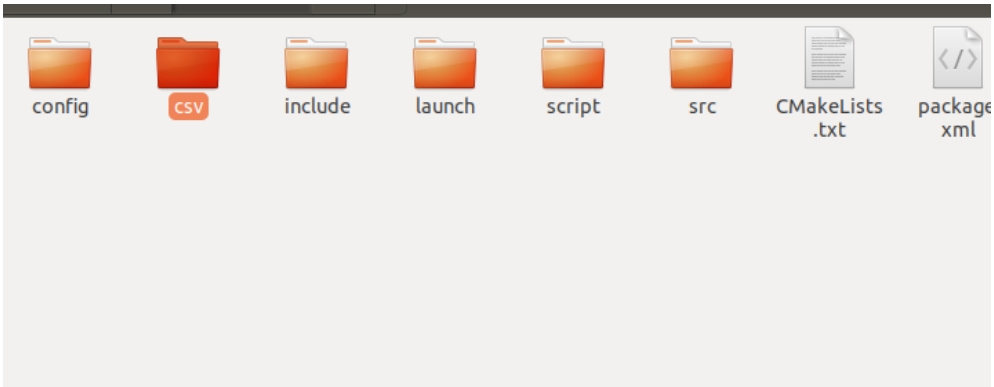


Figura 4.4: Pasta csv selecionada

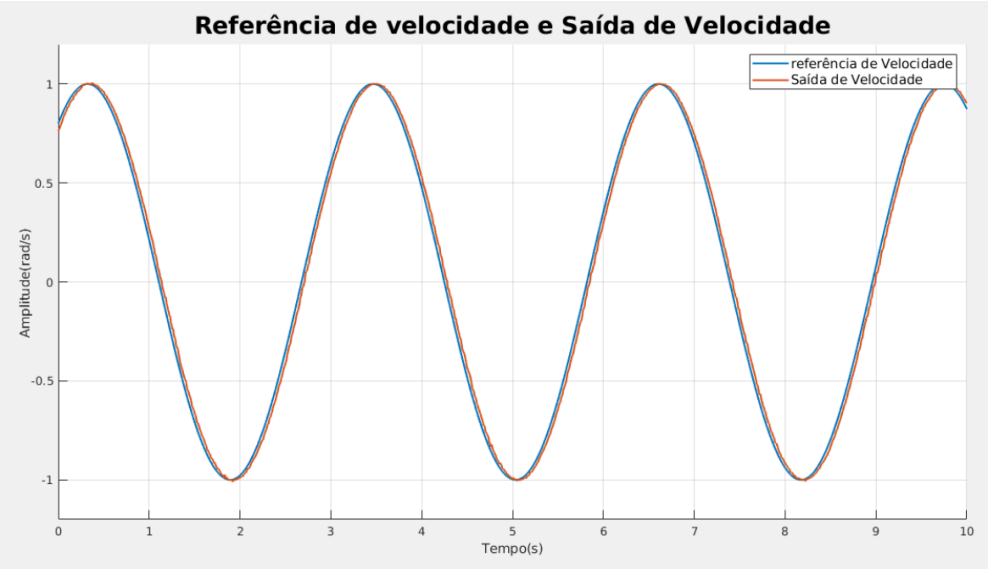


Figura 4.5: Dados coletados usando a Interface de Comunicação [4]

### 4.1.2 Identificação através do Matlab

A partir dos dados coletados, utilizou-se a toolbox "System Identification"do Matlab para fazer a identificação do sistema.

A identificação seguiu um método caixa-cinza, adotando inicialmente um modelo de primeira ordem, $T(s) = \frac{\beta}{\tau s+1}$, como o mais simples para descrever o sistema, pois o foco será dado ao controlador e não a identificação.

Importando os dados coletados e informando qual estimativa de modelo na toolbox do System Identification foi possível achar uma identificação com 97.94 % de aderência, exposta na Figuras 4.6 e 4.7 e com a função de transferência dada pela equação 4.1

$$G(s) = \frac{\Omega(s)}{\Omega_r(s)} = \frac{44.44}{s + 44.39} \tag{4.1}$$

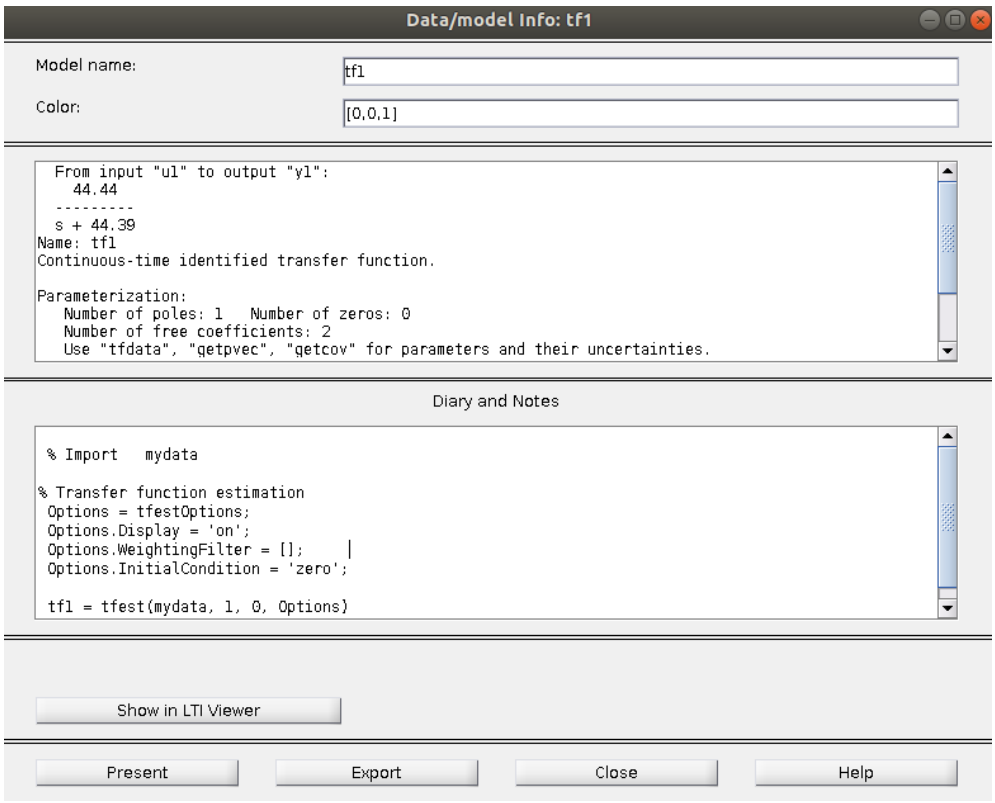


Figura 4.6: Função de transferência obtida através do System Identification

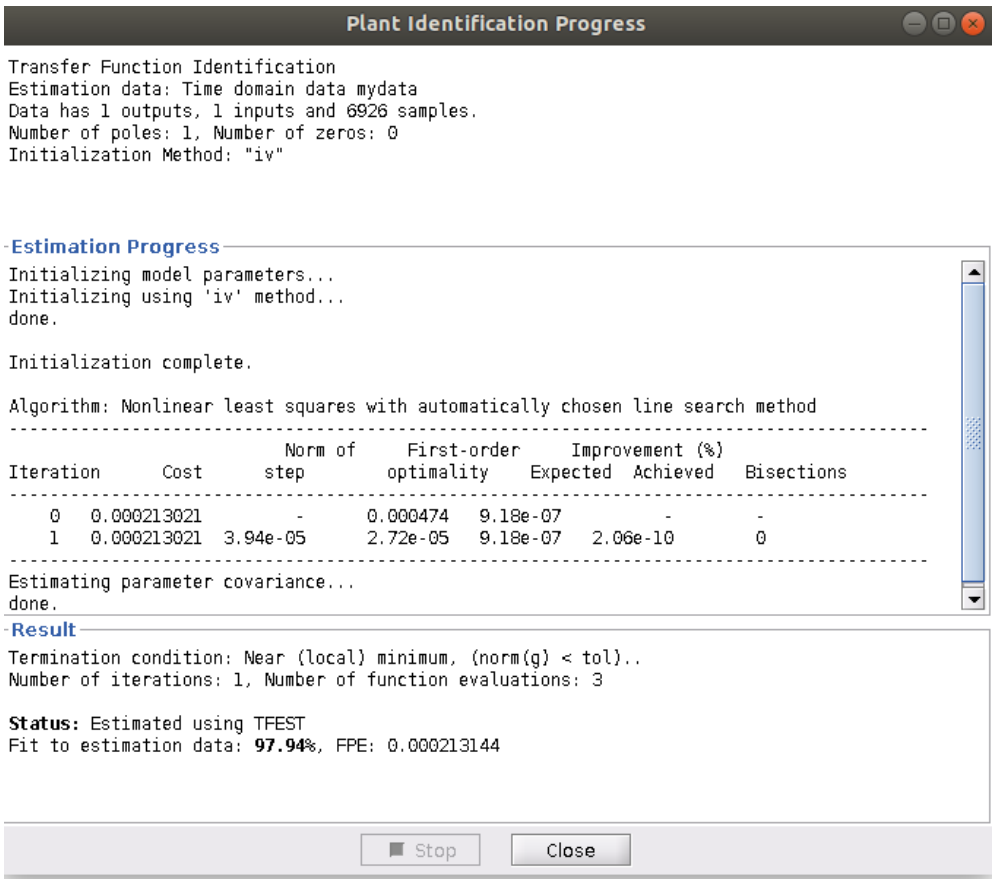


Figura 4.7: Porcentagem de fit para o modelo identificado

## 4.2 Especificações de projeto

Para uma boa aplicação desse projeto, assumimos 2 ideais principais a serem seguidos.

1. Erro estacionário e sobre-sinal baixos para entrada degrau, objetivando um controle de maior precisão.

2. Resposta do sistema mais lenta comparado a velocidade máxima da junta para uma transição segura entre as posições.

Com base nesses ideais adotou-se erro estacionário, *maximum percentage overshoot* e tempo de pico como parâmetros dos projetos.

1. Para o $M_p$ (Maximum Percentage overshoot) o ideal seria ter um valor nulo, pois em alguns casos passar da posição desejada pode ocasionar riscos tanto ao robô quanto ao processo de manipulação que está ocorrendo. Como em nosso projeto não teremos manipulações em procedimentos críticos e nem trabalharemos perto dos limites máximos de deslocamento da junta, definimos $M_p$ dentro de uma margem $M_p < 2\%$

2. Para o erro estacionário podemos admitir $e_{ss} = 0$, pois pode-se manipular o controlador para alocar um polo em $z = 1$ na função transferência em malha aberta o que levaria o erro pra entrada degrau à zero.

3. Para o tempo de pico realizou-se algumas considerações inciais. Primeiramente, foi adotado um intervalo de deslocamento de $\pm 0.3rad$, o qual corresponde a uma fração de $4,77\%$ do limite máximo do deslocamento da junta do robô segundo [31] ($DeslocMax = \pm 2\pi rad$). Após essa definição, planejou-se um controle que tivesse uma resposta transitória no mínimo 10x mais lenta que a máxima para assegurar uma transição segura. Como a velocidade máxima da junta equivale a $180°/s$ arbitrando o tempo de pico para $t_p = 1,5$ teríamos uma velocidade 15,7x mais lenta ($\frac{2\pi/1}{0.3/1.5} = 15,7$), atendendo bem o planejamento inicial.

A lista a seguir sumariza os valores dos parâmetros:

- Maximum percentage overshoot: $M_p = 2\%$
- Peak time: $T_p = 1.5s$
- $e_{ss} = 0$

## 4.3 Implementação do sistema de controle

### 4.3.1 Estrutura do sistema

O diagrama de bloco da Figura 4.8 mostra o projeto do sistema em malha fechada no simulador do Matlab Simulink.

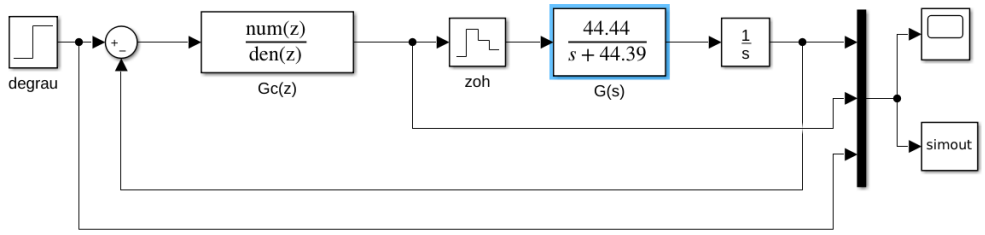


Figura 4.8: Representação do sistema em malha fechada por diagrama de blocos

A entrada degrau representa a referência de posição, com a amplitude do sinal representando as coordenadas em radianos.

O sinal de erro de posição ao passar pelo controlador $G_c(z)$ (a ser implementado), é transformado em um sinal de velocidade, o qual depois passa por um segurador de ordem zero e então é aplicado na junta robótica.

Por fim, integramos o sinal na saída da junta pra transformá-lo novamente em posição e realimentamos o sistema.

### 4.3.2 Localização dos polos

Para implementação do controlar pelo método do lugar geométrico das raízes, inicialmente determinou-se a localização desejada para os polos de malha fechada do sistema com base nas especificações da seção 4.2.

- Calculo do fator de amortecimento:

$$M_p = exp\left(\frac{-\xi\pi}{\sqrt{1-\xi^2}}\right) = 0.02 \tag{4.2}$$

$$\xi \approx 0.7797032 \tag{4.3}$$

- Calculo da frequência natural:

$$t_p = \frac{\pi}{w_n\sqrt{1-\xi^2}} = 1.5 \tag{4.4}$$

$$w_n \approx 3.34488 \tag{4.5}$$

- Calculo dos polos no domínio S:

$$S_{1,2} = -\xi w_n \pm w_n\sqrt{1-\xi^2} = -2.60801 \pm 2.09439j \tag{4.6}$$

- Logo, a localização desejada dos polos em malha fechada no domínio Z é dada por:

$$Z_{1,2} = exp\left([-2.60801 \pm 2.09439j](0.008)\right) = 0.9792146 \pm 0.01640839j \tag{4.7}$$

### 4.3.3 Determinação do ganho, polos e zeros do controlador

Sendo a posição a integral da velocidade

$$\Theta(s) = \frac{\Omega(s)}{s} = \frac{G(s)}{s}\Omega_r(s) = G_1(s)\Omega_r(s) \tag{4.8}$$

com

$$G_1(s) = \frac{44,44}{s(s+44.49)} \tag{4.9}$$

Para traçar o LGR do sistema utilizou-se o Sisotool do Matlab para a função de transferência em malha aberta do sistema $G_1(z) = \frac{1.27e{-}3(z{+}0.8885)}{(z{-}1)(z{-}0.7011)}$, que resultado da discretização de $G_1(s)$ utilizando segurador de ordem zero. A figura 4.9 mostra o resultado obtido desse plot.

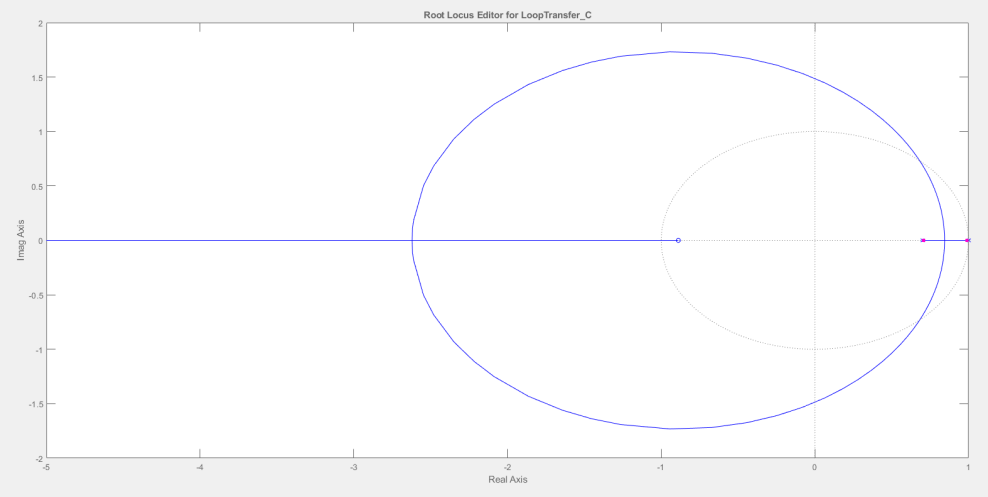


Figura 4.9: Traçado do LGR para $G_1(z)$

Para orientar o projeto do controlador, adicionamos as curvas de especificações de $\xi$ e $w_n$ ao plot do LGR, conforme Figura 4.10 e 4.11. Nas figuras, os traços em negrito delimitam as regiões onde se atendem os requisitos.

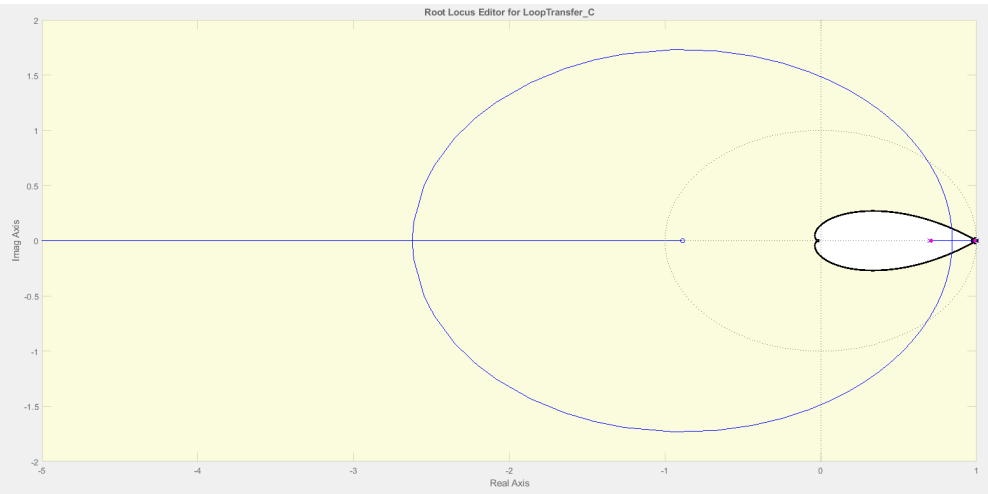


Figura 4.10: LGR com curvas de especificação do sistema

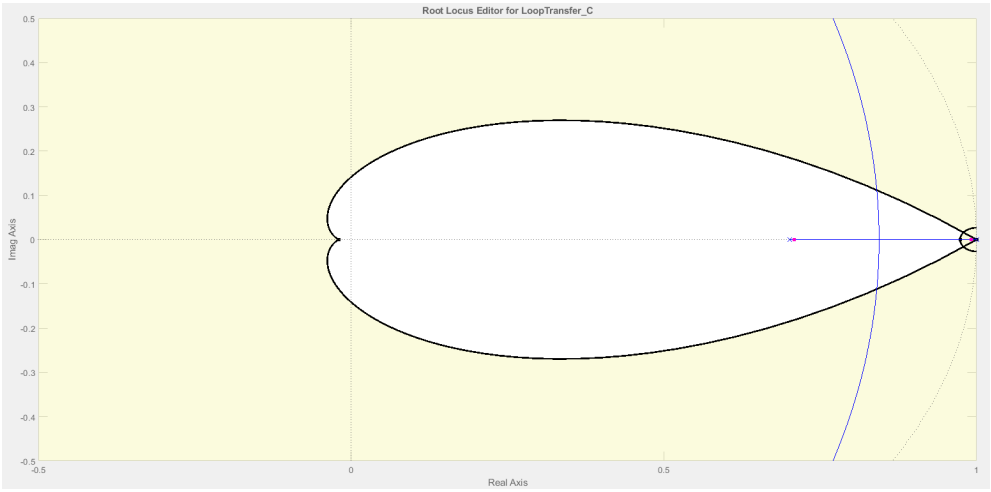


Figura 4.11: Vista amplificada do LGR com curvas de especificação

A figura 4.11 nos mostra que precisamos transladar o ponto de partida dos polos no eixo real para a esquerda para que o traçado do LGR passe pela intersecção das curvas de especificações do projeto.

Adotando inicialmente que o controlador terá o seguinte formato $G_c(z) = \frac{K(z-z_z)}{(z-z_p)}$, pode-se utilizar a condição de ângulo para determinação da localização exata do polo $z_p$ e do zero $z_z$.

$$-\angle(Z_1 - 1) - \angle(Z_1 - 0.7011) - \angle(Z_1 - z_p) + \angle(Z_1 + 0.8885) + \angle(Z_1 - z_z) = 180 \pm 360n \quad (4.10)$$

A equação 4.10 adota $Z_1$ como sendo um dos polos de malha fechada desejados (no caso o que possui parte imaginária positiva) e "n"como um número inteiro qualquer.

Adotando $z_z = 0.7011$ cancelando um dos polos do sistema, podemos resolver a equação 4.10 para determinar $z_p$

$$-141.712 - \angle(Z_1 - z_p) + 0.503346 = -180 \quad (4.11)$$

$$arctan(\frac{0.01640839}{0.9792146 - z_p}) = (38.7913\pi/180) \quad (4.12)$$

$$z_p = 0.9588 \quad (4.13)$$

Com os valores de $z_p$ e $z_z$ podemos adicionar manualmente o polo e zero do controlador no sisotool e determinar o ganho K necessário experimentalmente, ajustando a posição dos polos até obtermos um valor próximo ao de $Z_{1,2}$. A figura 4.12 mostra esse processo, a equação 4.14 mostra a formula final do controlador e a equação 13 mostra a formula da função de transferência do

controlador no formato de equações de diferenças.

$$G_c(z) = \frac{\Omega_r(z)}{E(z)} = \frac{0.30067(z - 0.7011)}{(z - 0.958)} \tag{4.14}$$

$$E(z) = \Theta_r(z) - \Theta(z)$$

$$u[k] = 0.958 * u[k-1] + 0.30067 * e[k] - 0.2108 * e[k-1] \tag{4.15}$$

$$e[k] = \theta_r[k] - \theta[k] \tag{4.16}$$

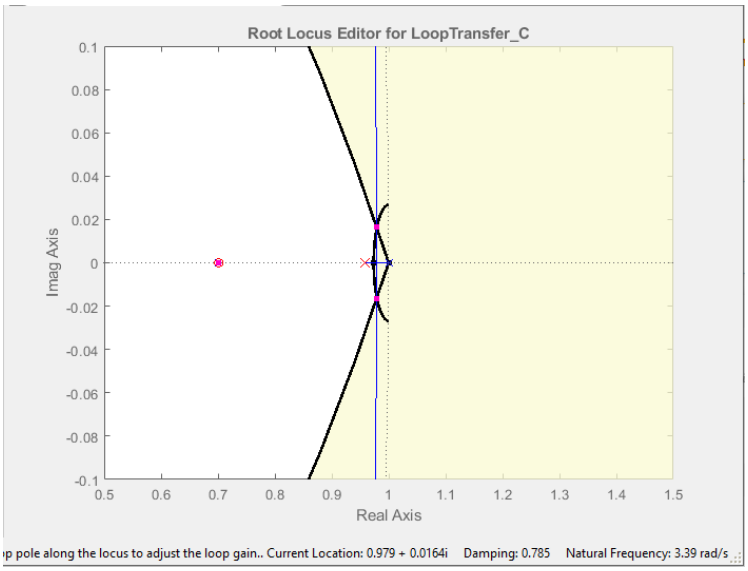


Figura 4.12: Determinação do ganho K

## 4.4 Simulações e Resultados

### 4.4.1 Simulações

A Figura 4.13 mostra a representação do sistema completo através do Simulink.

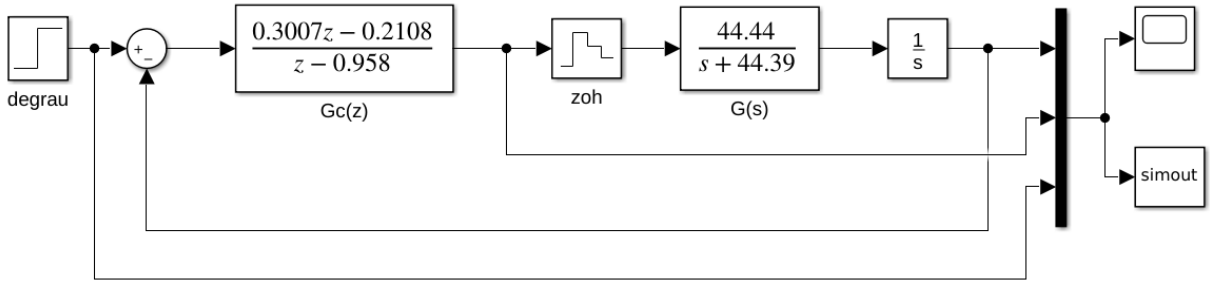


Figura 4.13: Sistema completo em malha fechada representado pelo Simulink

A Figura 4.14 mostra o resultado da simulação para o diagrama da Figura 4.13, mostrando o comportamento do sistema para uma referência de posição do tipo degrau. Na simulação obtemos valores de overshoot e tempo de pico extremamente condizentes com os impostos na seção 4.2.

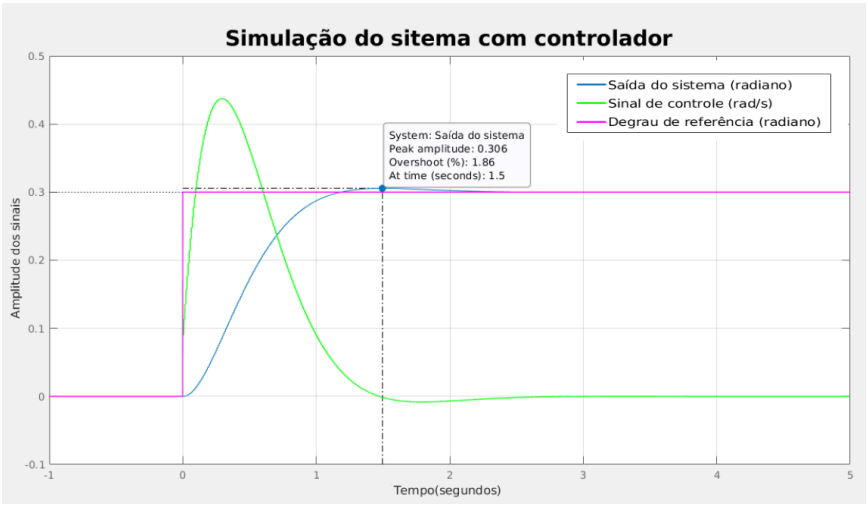


Figura 4.14: Resposta do Sistema com a presença do controlador Gz

### 4.4.2 Resultados

Ao levar o controlador implementado para testes práticos com a junta real do UR3, obteve-se as respostas apresentadas na Figura 4.15 e 4.16.

As figuras 4.15 e 4.16 mostram a resposta do sistema controlado a uma referência de posição do tipo degrau e do tipo onda quadrada respectivamente. Novamente obtemos valores de overshoot e tempo de pico extremamente próximos com os impostos na seção 4.2.

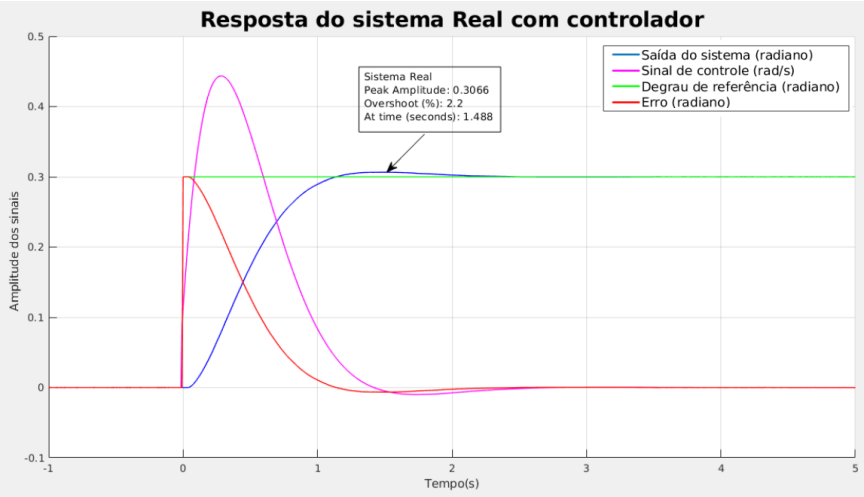


Figura 4.15: Resposta do Sistema real para uma entrada do tipo degrau

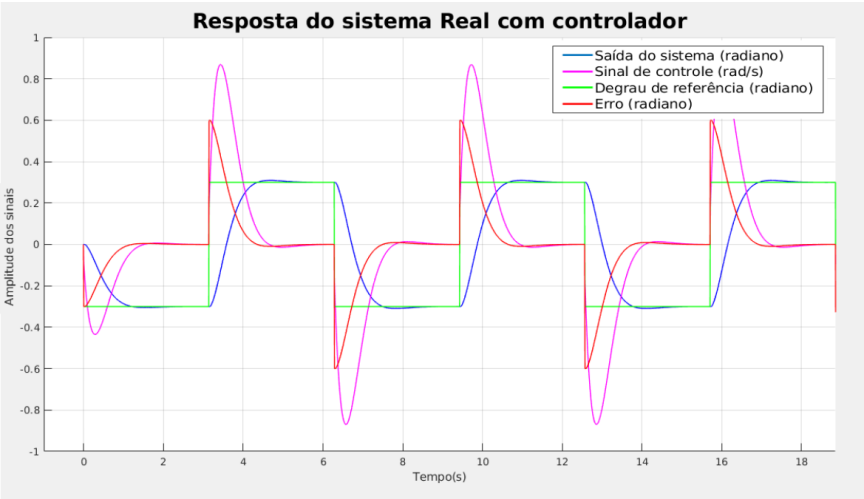


Figura 4.16: Resposta do Sistema real para uma entrada do tipo onda quadrada

Dado que o UR3 deve receber dados a cada 8 milissegundos torna-se necessário verificar se o Linux PC é capaz de executar os cálculos demandados pelo ROS e envia os dados para o UR3 na mesma periodicidade. Para isso, foi colocado dois temporizadores internos no código do controlador para averiguar se os tempos de execução da lei de controle e do ROS são compatíveis com a periodicidade que o UR3 necessita. O primeiro temporizador (em vermelho na Figura 4.17) mede o tempo de execução da lei de controle e pode-se perceber que o tempo de execução máximo é menor que 1 milissegundo. Já a periodicidade da execução da lei de controle (em roxo na Figura 4.17) foi medida para verificar se a lei de controle estava sendo executada como determinado pelo ROS, ou seja, a cada 8 milissegundo a lei de controle é executada e enviada para o UR3.

Quanto ao atraso da transmissão de dados do controlador, localizado no Linux PC, para o UR3 (efeito do *jitter* de comunicação), não foi estudado nesse trabalho pois estava fora do seu escopo.

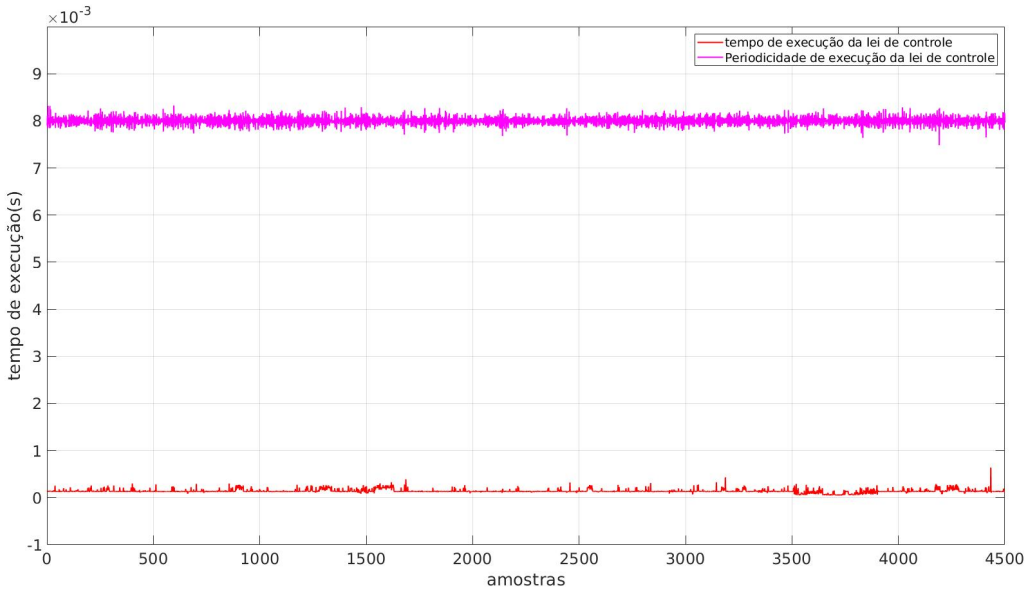


Figura 4.17: Tempo de execução e periodicidade da lei de controle extraído com temporizador Python usando a biblioteca **Time** [5]

Para obter o valor médio da taxa de execução e o menor e maior tempo de execução da lei de controle, foi usada a ferramenta de monitoramento de tópicos do ROS no tópico que publica a informação do sinal de controle, sendo que o tópico foi programado para executar a uma taxa de 125 hz. Foram encontrados os resultados da Figura 4.18 e apresentados a seguir:

- Taxa média de execução: 124,998 hz
- Menor tempo de execução: 0,007482s
- Maior tempo de execução: 0,0083s

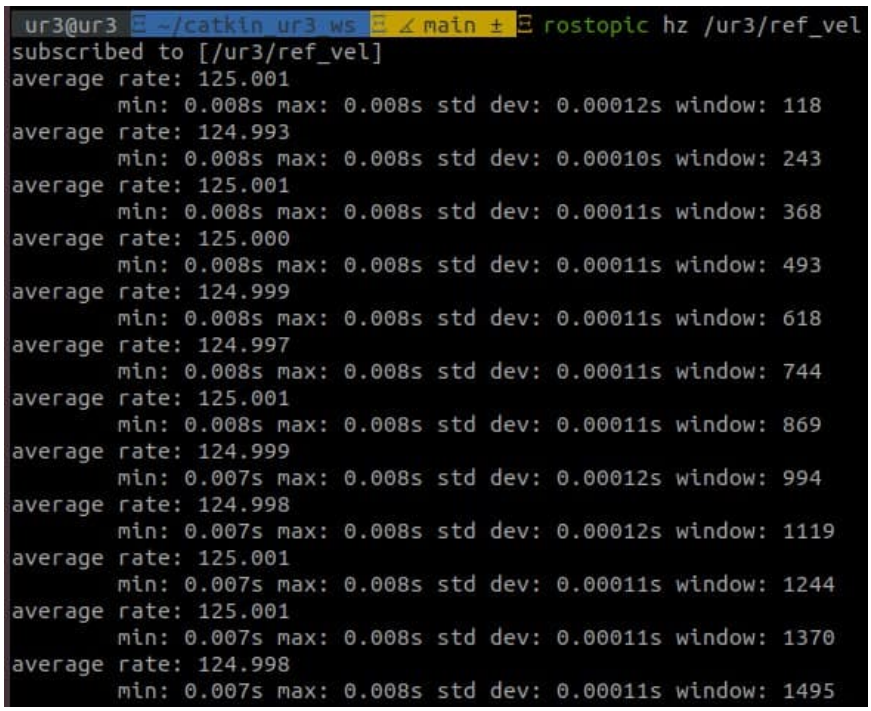


Figura 4.18: Comportamento da velocidade de execução do laço do controlador

## 4.5 Conclusão

Ao longo desse Capítulo buscou-se modelar a junta Elbow joint do robô UR3 e implementar um controle de posição pautado em limitações de deslocamento, velocidade e outros requisitos de projeto.

A partir das seção de resultados, foi observado que os parâmetros de overshoot, tempo de pico e erro estacionário (coletados experimentalmente usando a Interface de Comunicação [4]) não possuem grandes discrepâncias em relação ao valores impostos na seção de especificação dos sistema (seção 4.2). Pode-se afirmar que obteve-se sucesso no processo de modelagem, identificação do sistema e projeto de controle de posição.

Como passos futuros pode-se expandir esse estudo de controle para respostas transitórias mais rápidas e regiões de deslocamento maiores (mais próximos dos limites máximos do robô).

# Capítulo 5

# Conclusão

Neste trabalho foi apresentado a construção de uma plataforma para pesquisa e desenvolvimento para o robô manipulador UR3 presente no LARA usando o *framework* ROS. Para isso, foi feita uma pesquisa sobre o funcionamento dos robôs manipuladores da *Universal Robots* a fim de estabelecer uma comunicação via software usando ROS para gerar um suporte as aplicações de robótica, visto que, os robôs da *Universal Robots* possuem uma arquitetura relativamente fechada para pesquisa, que, até o momento, a *Universal Robots* não oferece suporte para entrada de torque nas juntas do UR3 mas permite a leitura dos torques. Mesmo com essas limitações referentes a atuação de baixo nível nas juntas to UR3, foi desenvolvida uma Interface de Comunicação 2.2.3 que objetiva padronizar as unidades dos sinais de entra e saída do manipulador e estabelecer um padrão de desenvolvimento de *software* baseado no *framework* ROS.

Para os parâmetros de **Denavit-Hartenberg** e para as orientações das juntas do UR3 foram encontrados resultados satisfatórios usando a plataforma de desenvolvimento proposta. A partir desses resultados, que trazem um comparativo com a documentação do fabricante [28], podemos concluir que o método usado foi adequado, visto que o maior erro percentual encontrado foi inferior a 1%.

Outra contribuição deste trabalho de graduação, foi a elaboração de uma documentação a respeito dos aspectos construtivo com relação ao robô físico e *software* utilizado na implementação da plataforma de pesquisa, visando acelerar a curva de aprendizagem de novos usuários pesquisadores no laboratório LARA. Além disso, com base nos resultados obtidos nos capítulos 3, 4 e na documentação referente a realização de experimentos simplificados nos Apêndices A, B, C e D, é notório que a replicabilidade dos trabalhos de graduação necessitam de documentação com foco em um primeiro contato com o tema aborda da foma mais simplificada, tentando replicar a mesma filosofia dos famosos **"*Hello World*"** tão recorrentes nas matérias de computação quando o aluno é um iniciado naquela área de conhecimento.

Todo o material desenvolvido (roteiros dos experimentos, programas da Interface de Comunicação e do pacote **ur3_control**, PIBIT que mostrada detalhes da parte construtiva da Interface de Comunicação) pode ser acessado no github[1].

[1] `https://github.com/lara-unb/catkin_ur3_ws`

Apesar de o UR3 ser uma plataforma mais voltada para o uso industrial, o UR3 presente no LARA pode ser amplamente utilizado para estudos na área de controle e principalmente para interação com pessoas ou outros robôs.

Para trabalhos futuros com o UR3 sugerem-se:

1. implementar controladores cinemáticos envolvendo todas as juntas;
2. replicar e ampliar a plataforma de desenvolvimento para incluir o robô manipulador Meka A2 e os robôs móveis Pioneers presentes no LARA[2];
3. incluir o pacote ROS multimaster_fkie [32] na Interface de Comunicação (cf. Figura 2.17) para permitir a intercomunicação entre o UR3 e outros robôs via ROS;
4. identificar o modelo dinâmico com as mesmas abordagens de [14] e [13] utilizando a plataforma proposta no presente trabalho.

[2]Documentação disponível no endereço `https://cooprobo.readthedocs.io/en/latest/`

# REFERÊNCIAS BIBLIOGRÁFICAS


[1] ROBOTS, U. *DYNAMIC FORCE CONTROL*. Disponível em: <https://www.universal-robots.com/articles/ur/programming/urscript-dynamic-force-control/>.

[2] JOSEPH, L.; CACACE, J. *Mastering ROS for Robotics Programming - Second Edition: Design, Build, and Simulate Complex Robots Using the Robot Operating System*. 2nd. ed. [S.l.]: Packt Publishing, 2018. ISBN 1788478959.

[3] SPONG, M.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot Modeling and Control*. Wiley, 2005. ISBN 9780471649908. Disponível em: <https://books.google.com.br/books?id=A0OXDwAAQBAJ>.

[4] MATOS, R. R. Interfaciamento amigável para robôs manipuladores. In: *26º CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UNB E 17º DO DF*. Brasília, Brasil: [s.n.], 2020. p. 1–12.

[5] Python. *Time access and conversions*. Disponível em: <https://docs.python.org/3/library/time.html>.

[6] Education and Training. *Simple Machines*. Disponível em: <https://www.education.vic.gov.au/school/teachers/teachingresources/discipline/science/continuum/Pages/simpmachines.aspx>.

[7] COLLABORATIVE Robots and Industrial Revolution 4.0 (IR 4.0). In: 2020 International Conference on Emerging Trends in Smart Technologies (ICETST). [S.l.: s.n.].

[8] WOMACK, J. P.; JONES, D. T.; ROOS, D. *The Machine that Changed the World*. New York: Rawson Association, 1990. ISBN 0-89256-350-8.

[9] JESSOP, B. Fordism and post- fordism: A critical reformulation. *Fordism and Post Fordism: A Critical Reformation*, p. 46–69, 01 1992.

[10] Automni Team. *Tecnologia na Logística: o uso da automação em Centros de Distribuição*. Disponível em: <https://automni.com.br/tecnologia-na-logistica-o-uso-da-automacao-em-centros-de-distribuicao/>.

[11] Automni Team. *automação pode aumentar a produtividade e promover ambientes mais alegres*. Disponível em: <https://automni.com.br/voce-sabia-que-automacao-pode-aumentar-a-produtividade-e-promover-ambientes-mais-alegres-so-que-nem-tudo-sao-flores/>.

[12] ROBOTS, U. *User Manual UR3/CB3*. [S.l.].

[13] GARCIA, P. *Estimação de Força de Interação no Braço Robótico UR3*. 2019. Trabalho de Graduação em Engenharia de Controle e Automação, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF.

[14] CESARINO, P. S. D. N. *Modelagem, Identificação e Design de Controle para o Robô Manipulador UR3*. 2020. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação PPGEA.TD-001/11, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 90p, Brazil.

[15] YOUNG, I. *DIRECT DRIVES IN LIGHTWEIGHT ROBOTS*. Disponível em: <https://www.kollmorgen.com/en-us/service-and-support/knowledge-center/success-stories/direct-drives-in-lightweight-robots/>.

[16] SPECIFIE, E. *Harmonic Drive AG. Universal Robots Given a Helping Hand*. Disponível em: <https://www.engineeringspecifier.com/mechanical-components/universal-robots-given-a-helping-hand>.

[17] BOSCARIOL, P. et al. Energy optimization of functionally redundant robots through motion design. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 10, n. 9, p. 3022, 2020.

[18] SE, S. D. *We are supplying Universal Robots with HFUS-2SH Series Gears to help optimise production processes*. Disponível em: <https://twitter.com/harmonicdriveuk/status/560399204397645825>.

[19] AKSIM. *Supports Universal Robots for smart factory automation*. Disponível em: <https://www.renishaw.com/en/aksim-supports-universal-robots-for-smart-factory-automation–40389>.

[20] ROBOTS, U. *The URScript Programming Language*. [S.l.].

[21] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Disponível em: <https://www.ros.org>.

[22] ROSSUM, G. V.; JR, F. L. D. *Python reference manual*. [S.l.]: Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[23] w3schools.com. *C++ Class Methods*. Disponível em: <https://www.w3schools.com /cpp/cpp_class_methods.asp>.

[24] Dorian Scholz, Dirk Thomas. *rosbag*. Disponível em: <http://wiki.ros.org/rosbag>.

[25] ANDERSEN SIMON RASMUSSEN, F. E. L. S. T. S. T. T. *ur_robot_driver*. Disponível em: <http://wiki.ros.org/Industria>.

[26] Michael Carroll. *dynamic_reconfigure*. Disponível em: <http://wiki.ros.org/dynamic_reconfigure>.

[27] DENAVIT, J.; HARTENBERG, R. S. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics*, v. 22, p. 215–221, June 1955.

[28] Denavit Hartenberg Parameters - DH Parameters. *DH PARAMETERS FOR CALCULATIONS OF KINEMATICS AND DYNAMICS*. Disponível em: <https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/>.

[29] FARIA, C. et al. Automatic denavit-hartenberg parameter identification for serial manipulators. In: *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*. [S.l.: s.n.], 2019. v. 1, p. 610–617.

[30] MATLAB. *version 9.5.0.9 (R2018b)*. Natick, Massachusetts: The MathWorks Inc., 2018.

[31] ROBOTS, U. *UR3 Technical specifications*. Disponível em: <https://www.universal-robots.com/media/240787/ur3_us.pdf>.

[32] Alexander Tiderko. *ultimaster_fkie*. Disponível em: <http://wiki.ros.org/multimaster$_f$*kie*>.

[33] Dorian Scholz, Dirk Thomas. *rqt_plot*. Disponível em: <http://wiki.ros.org/rqt_plot>.

# Apêndice

# Apêndice A

# Experimento 1: Ligar e desligar a plataforma de desenvolvimento

Este experimento objetiva fornecer um primeiro contato do usuário com o sistema robótico do manipulador UR3. Ele permitirá que o usuário seja capaz de:

- realizar o processo de inicialização do robô manipulador UR3;
- realizar o processo de inicialização do Linux PC;
- rodar um experimento simples;
- realizar o processo de desligamento do sistema.

## A.1 Setup: Robô manipulador UR3 e Controller

A Figura A.1 mostra a plataforma de desenvolvimento de aplicações para o braço manipulador UR3, presente no LARA, usando o ROS.

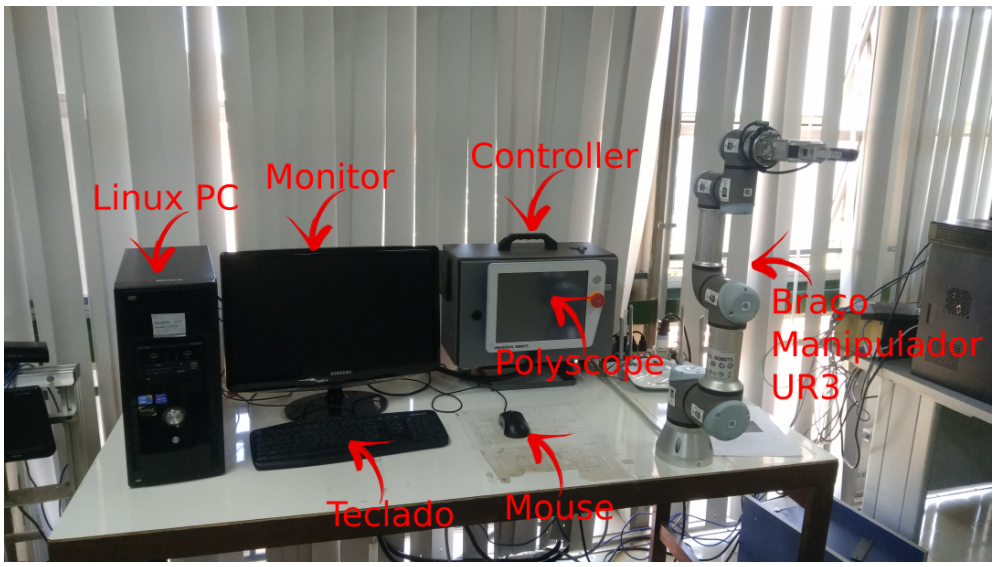


Figura A.1: Sistema completo

O computador do LARA, designado para fechar uma malha de controle com o robô UR3, é apresentado na Figura A.1 e é denominado **Linux PC**. O roteador dedicado ao sistema é chamado **roteador multi-robôs**.

A comunicação entre o roteador multi-robôs e o **Linux PC** é feita usando protocolo TCP/IP usando cabo (veja fotos da seção A.4.4.3).

### A.1.1 Setup: Robô manipulador UR3

Primeiro, ligamos o robô UR3 apertando o botão **POWER** que é mostrado em destaque por um retângulo vermelho na Figura A.2.


Figura A.2: Botão power

Depois de apertar o botão power, aparecerá a tela igual a Figura A.3

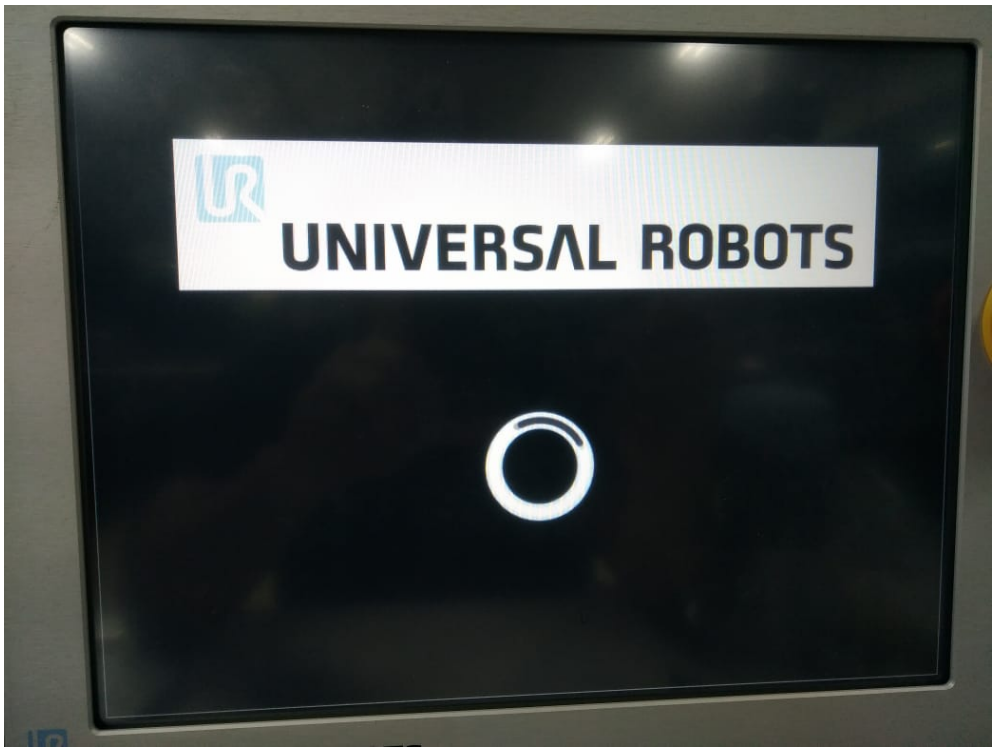


Figura A.3: Tela de carregamento

Depois que o sistema do robô carregar, aparecerá a tela igual a Figura A.4

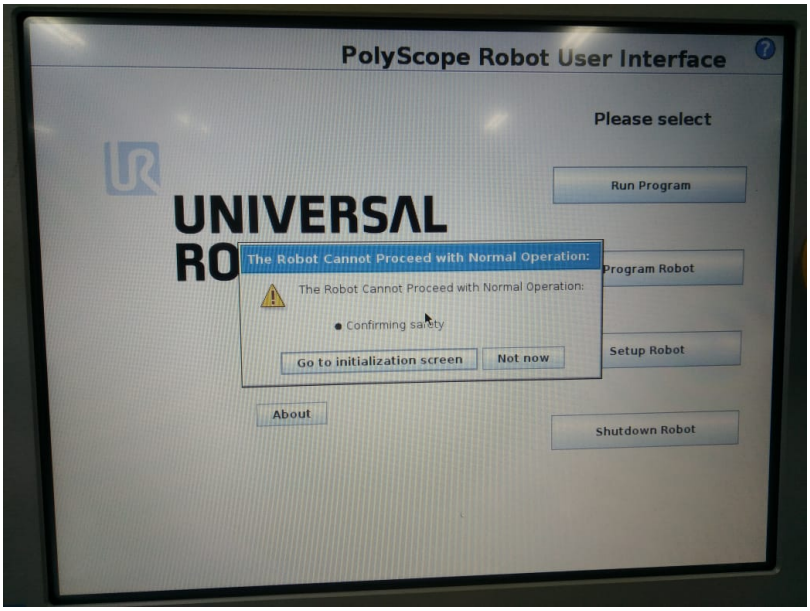


Figura A.4: Tela de apresentação

Agora, para ligar o braço robótico, clique em **Go to initialization screen**, que se encontra no centro da Figura A.4. Neste momento, temos a tela da Figura A.5.

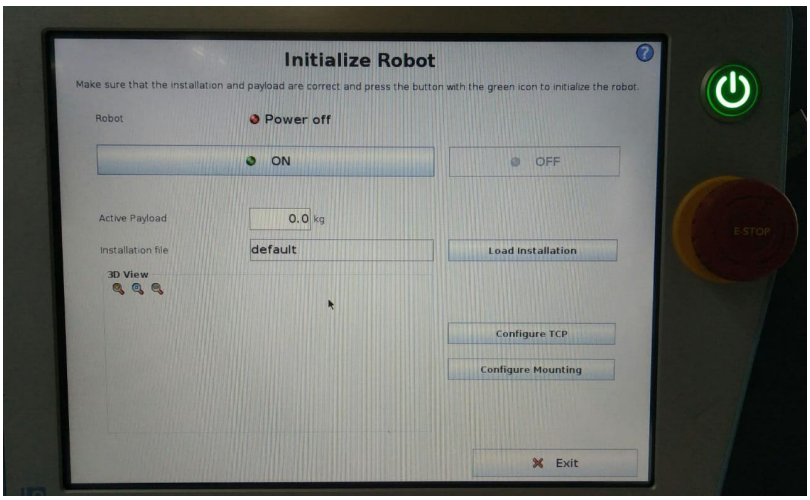


Figura A.5: Tela de inicialização

Para ligar o robô, pressione o botão **ON**. A tela que aparecerá será a da Figura A.6

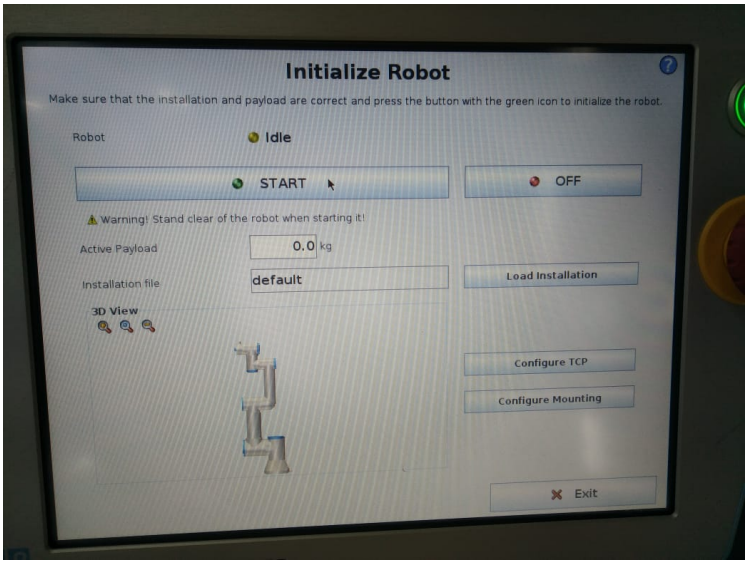


Figura A.6: Manipulador ligado. Destravamento: apertar START

Agora o robô está ligado. Para destravar as juntas aperte o botão **START** na Figura A.6.

**O robô fará um som de destravamento**, que é esperado, e entrará no modo de operação normal que pode ser visto na Figura A.7.

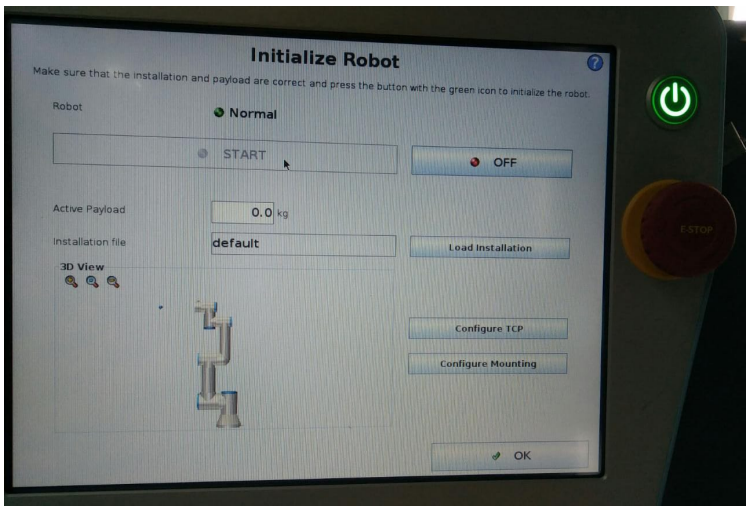


Figura A.7: OK

No canto inferior direito da tela (Figura A.7), aperte **OK**.

Agora, aparecerá uma nova tela (Figura A.8) como algumas opções de funcionalidades do robô. Deve-se clicar em **Program Robot** que está circulado em vermelho.

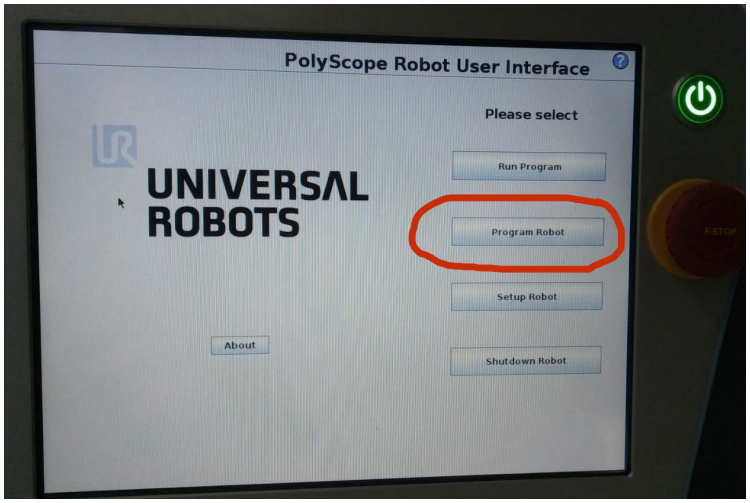


Figura A.8: Program Robot

Depois de apertar **Program Robot** aparecerá uma tela semelhante a Figura A.9. Aperte **Move**.

Pronto. Finalizamos nosso Setup do robô e a tela final é semelhante a Figura A.10.

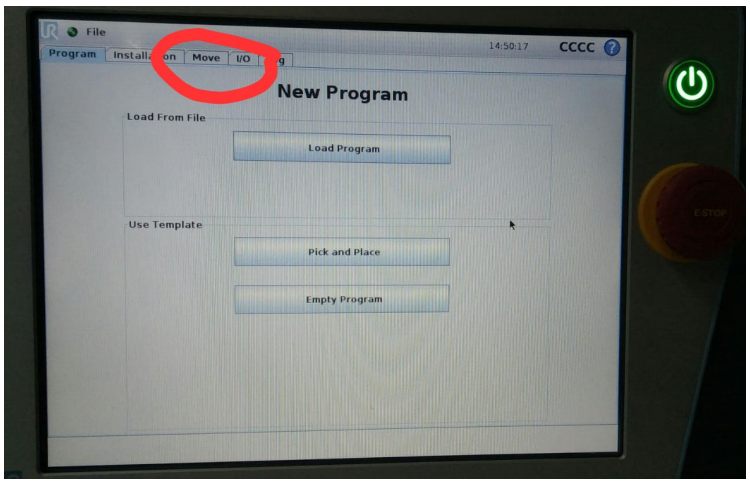


Figura A.9: MOVE

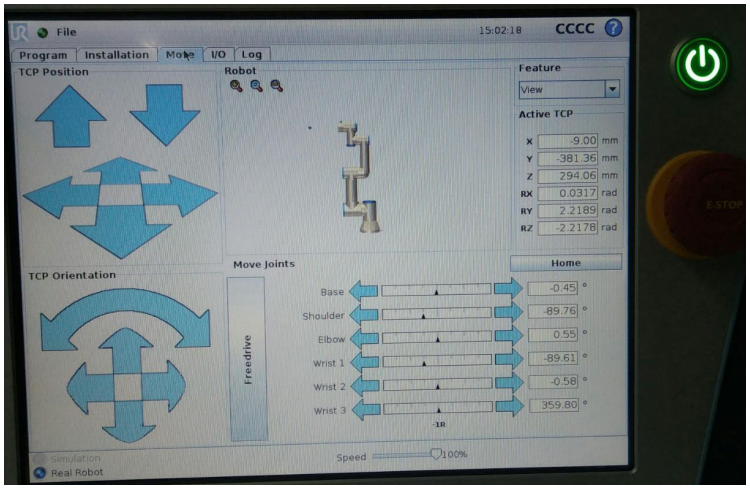


Figura A.10: Tela final para o setup do UR3

### A.1.2 Setup do Computador: Linux PC

Agora vamos ligar o computador Linux PC, que é usado para fechar uma malha de controle com o robô. O usuário do computador é **ur3** e senha **ur3**.

Abra a aplicação chamada **Terminator** circulada em vermelho na barra de tarefa do ubuntu 18.04, que é exatamente como a Figura A.11.

Depois de abrir o **Terminator**, vamos usar o comando **git clone** para baixar para o Linux PC o workspace catkin_ur3_ws, localizado no repositório github do LARA denominado lara-unb e localizado em https://github.com/lara-unb: digite no **Terminator** o comando[1]

**git clone https://github.com/lara-unb/catkin_ur3_ws.git**

[1]Em alguns terminais podem surgir na forma $ git clone https://github.com/lara-unb/catkin_ur3_ws.git. Neste caso, retirar o $

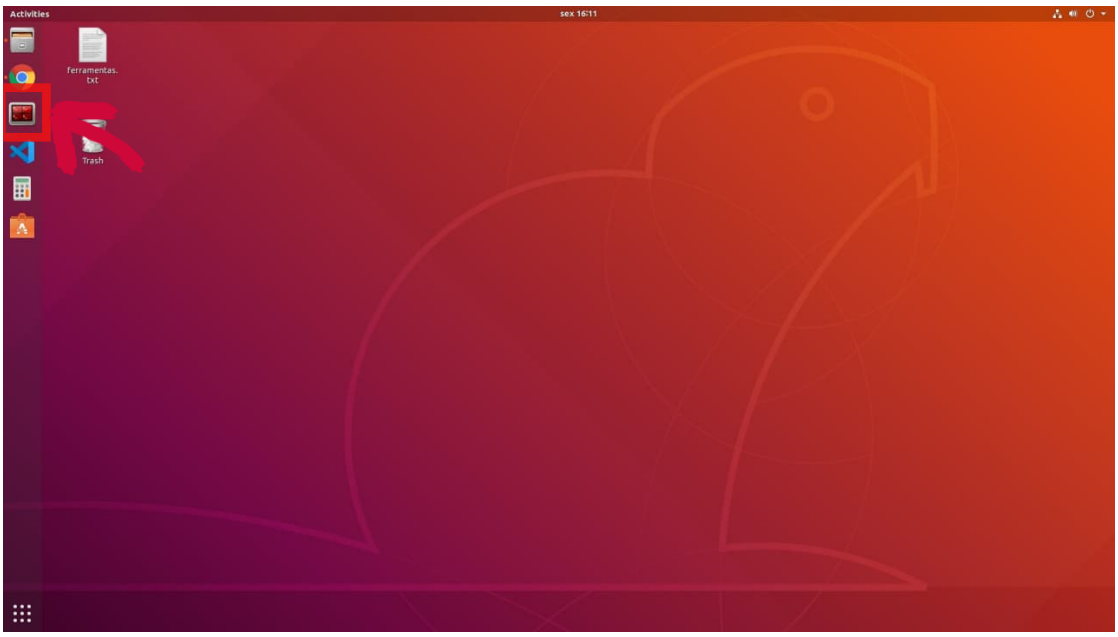


Figura A.11: Terminator

Na situação onde o **workspace cakin_ur3_ws** não exista no Linux PC, o workspace será baixado e deve surgir mensagens como mostrado na figura A.12.

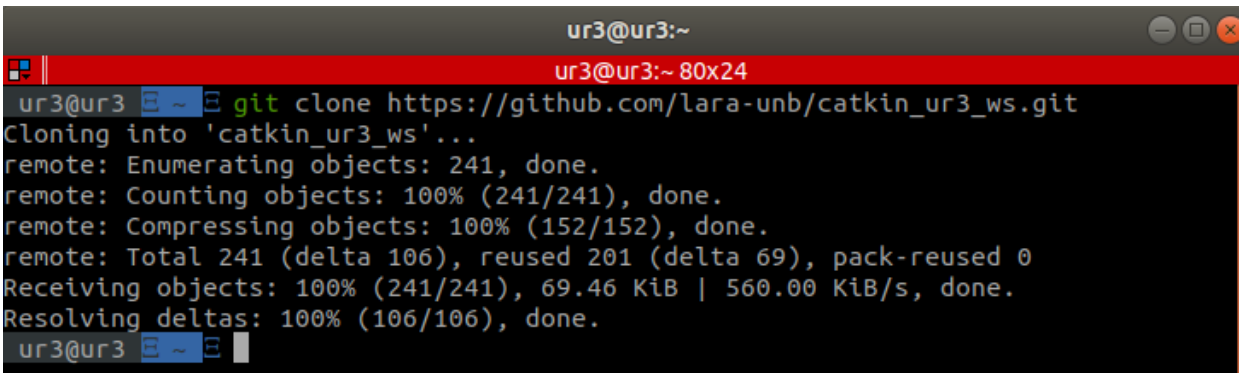


Figura A.12: git clone

Caso o **workspace cakin_ur3_ws** já exista no Linux PC, o usuário receberá uma mensagem de que o **workspace** já exite conforme Figura A.13

Com o workspace baixado, o passo seguinte consiste em entrar no diretório workspace usando o seguinte comando

**cd ~/catkin_ur3_ws**

Com isso, a tela do Terminator deve ser parecida com a da Figura A.14

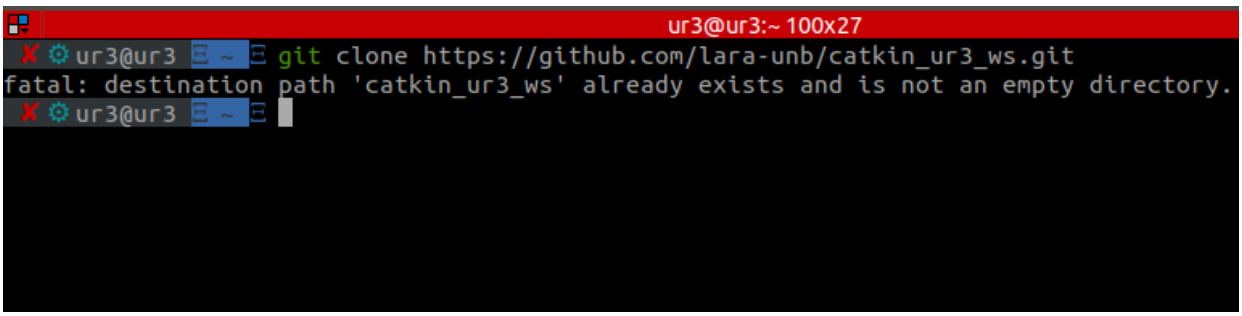


Figura A.13: Workspace existente

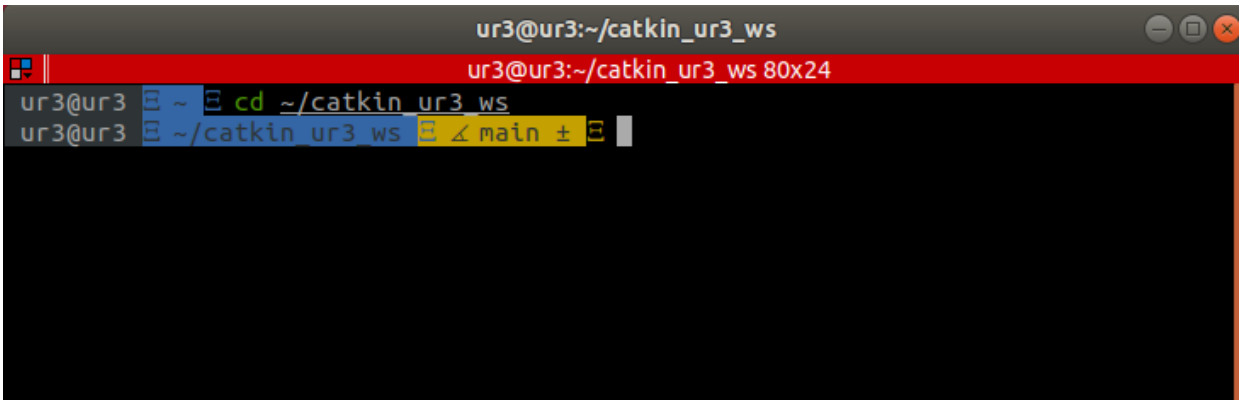


Figura A.14: catkin_ws

A seguir – para compilar os códigos da Interface de Comunicação, demos e mensagens e gerar os executáveis – use o comando:

**catkin_make**

O usuário receberá uma mensagem como a da figura A.15 abaixo.

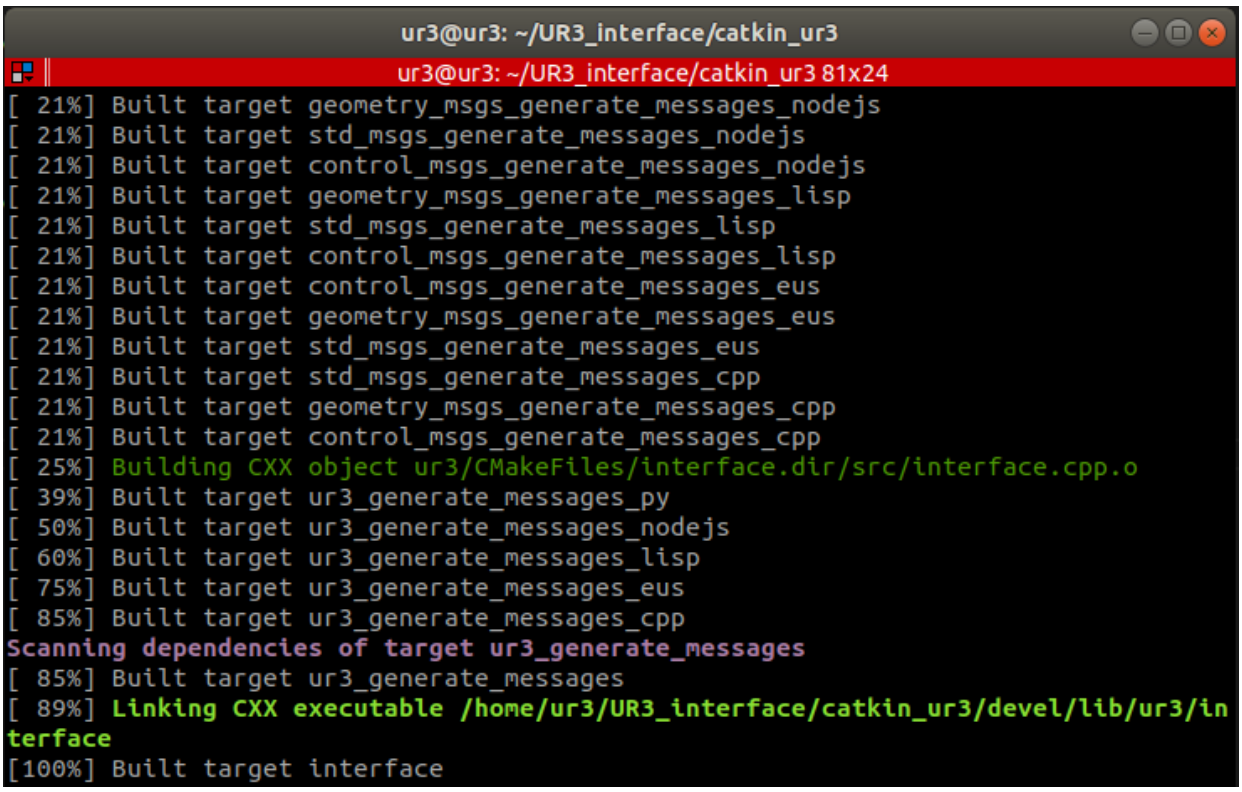


Figura A.15: Compilação

A parte de setup do computador para podermos usar o ROS está pronta. No próximo passo vamos aprender como lançar os pacotes ROS (Robot Operating System).

## A.2 Inicialização do ROS

Os arquivos executáveis já estão prontos (feito no último passo da seção anterior). Nesta seção, vamos apresentar a sequência de comandos necessários para que se lance a Interface de Comunicação.

Como primeiro passo, vamos abrir mais três seções do Terminator. Use Ctrl+shift+O para dividir o Terminator em seção horizontal e Ctrl+shift+E para dividir em seção vertical. Como resultado, teremos 4 (quatro) seções como mostrado na figura A.16 abaixo.


Figura A.16: Seções do Terminator

Cheque se as novas seções estão no mesmo workspace[2] conforme indicado na Figura A.16.

Uma vez que todas as seções estejam trabalhando no mesmo workspace catkin_ur3_ws, vamos rodar em cada seção, o comando

**source devel/setup.zsh**

para que **todas** as seções abertas sejam diretórios reconhecidos pelo ROS.

Para iniciar a Interface de Comunicação use o comando mostrado abaixo **na Seção 1**:

**roslaunch ur3 real_ur3.launch**

Depois de executar o comando acima, o usuário receberá uma resposta do Terminator, com a lista de parâmetros, as etapas de inicialização do UR3 e uma mensagem (em verde) dizendo que o nó ROS **Communication Interface** da Interface de Comunicação está rodando (vide Figura A.17)

[2]Caso alguma seção esteja em um workspace diferente dos mostrados na Figura A.16, execute nesta seção o seguinte comando **cd ~/catkin_ur3_ws**

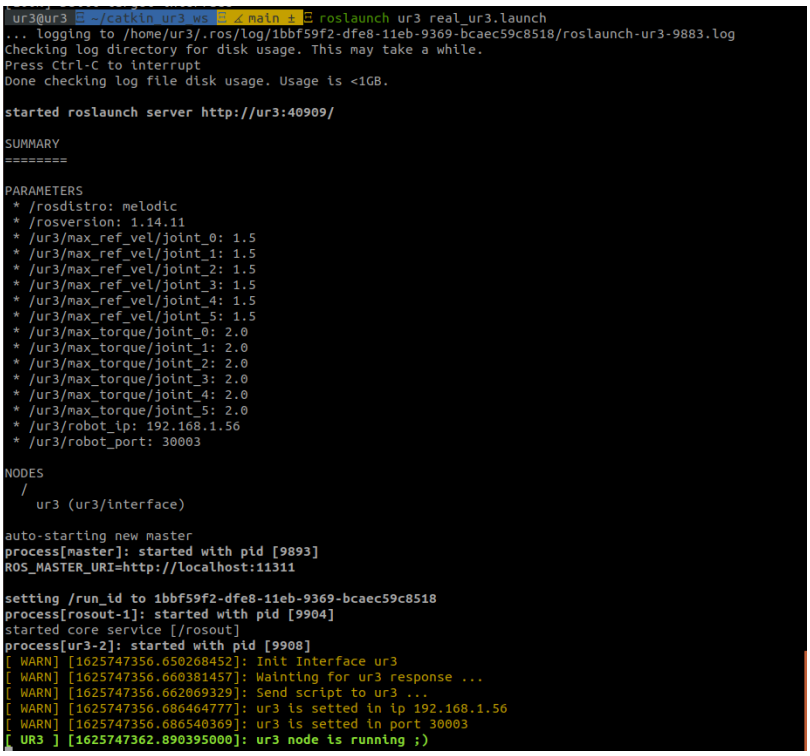


Figura A.17: Log do Launch ur3

Neste ponto, o nó ROS **Communication Interface**, que permite a intercomunicação entre o Linux PC e o Controller, está rodando. Com isso, a Interface de Comunicação está ativa, ou seja, capturando os dados dos estados das juntas do robô UR3, publicando estes dados no tópico **/ur3/arm** e esperando uma referência de velocidade no tópico **/ur3/ref_vel**.

## A.3 Demostração do experimento 1

Este experimento consiste em fazer um movimento com 1 minuto de duração da junta M32 ( **Elbow Joint**). O movimento é periódico entre as posições 0° e 45° para a junta M32 e é obtido usando-se referência de velocidade.

Para esta pequena demostração, foi feito, em linguagem **Python**, um nó ROS chamado **exp1** para publicar as referências de velocidade para as juntas do robô[3].

Para lançar o experimento 1, ou seja, executar o nó ROS **exp1**, vá para a seção 2 no Terminator e execute o seguinte comando

**roslaunch demos_ur3 exp1.launch**

Para iniciar a publicação das velocidades de referência para o robô, foi feito um serviço ROS chamado de **/demos_ur3/start_exp1** e para chamar esse serviço basta rodar o seguinte co-

[3]O arquivo que representa o nó ROS é chamado de **demo_exp1.py** e está armazenado no pacote **demos_ur3**.

mando na seção 3 do Terminator com *flag* **data** com o valor **true** (Olhe para robô depois que executar o comando, pois ele começará a fazer o movimento do experimento):

**rosservice call /demos_ur3/start_exp1 "data: true"**

Depois disso é esperado que o robô UR3 faça um movimento repetitivo parecido com o movimento do seguinte vídeo que está postado no youtube:

`https://youtu.be/pkN1GS0wX_o`

Caso queira visualizar os dados em tempo real dos estados da junta, citados em 2.2.3.2, siga os passos do experimento de visualização de dados no Apêndice C.

Caso queira parar o experimento, basta chamar o mesmo serviço com *flag* **data** com o valor **false** ,na Seção 3, ou espere o experimento acabar.

**rosservice call /demos_ur3/start_exp1 "data: false"**

Caso queira repetir o experimento, execute, na seção 4, o comando abaixo na para coloca o UR3 na posição de descanso.

**rosservice call /ur3/reset "{}"**

Depois rode o comando abaixo, na seção 3, para iniciar o experimento.

**rosservice call /demos_ur3/start_exp1 "data: true"**

Com a execução do nó **exp1** encerrado, caso desejado, pode-se mover as juntas do UR3 pressionando as setas, demarcados com retângulo vermelho representado na Figura A.18, apresentadas na tela touchscreen da interface **Polyscope** . Porém a conexão da caixa de controle do UR3 com a Interface Comunicação do Linux PC será desfeita e aparecerá uma mensagem como na figura abaixo.

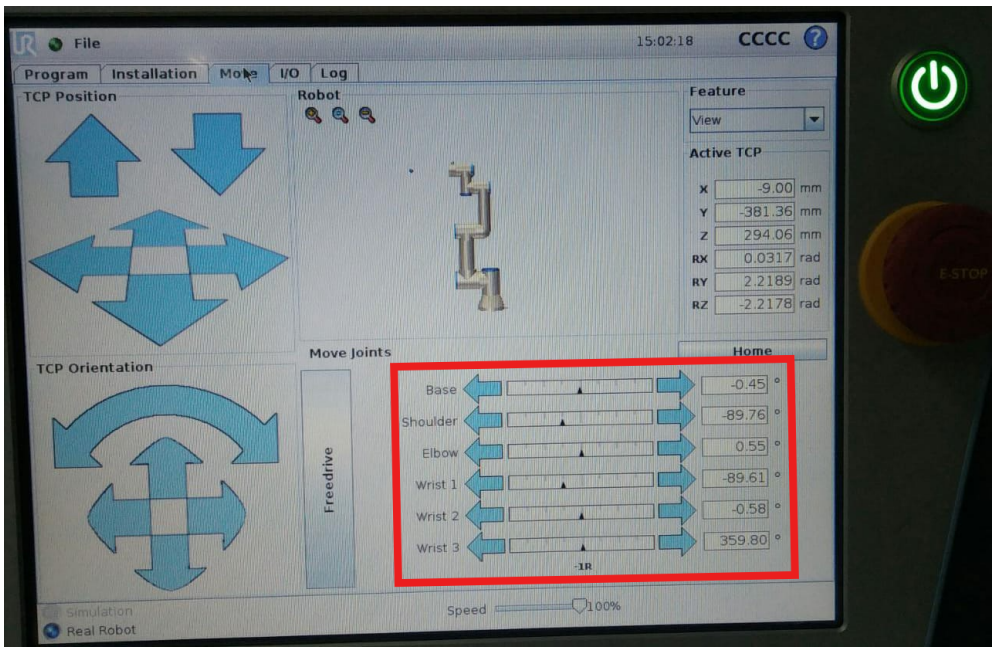


Figura A.18: Setas do Polyscope

```
process[ur3-2]: started with pid [14040]
[ WARN] [1626178241.983887313]: Init Interface ur3
[ WARN] [1626178241.996398715]: Wainting for ur3 response ...
[ WARN] [1626178241.997812734]: Send script to ur3 ...
[ WARN] [1626178242.021061366]: ur3 is setted in ip 192.168.1.56
[ WARN] [1626178242.021148202]: ur3 is setted in port 30003
[ INFO] [1626178247.021900740]: Opening Socket Communication ...
[ UR3 ] [1626178248.217183000]: Communication Interface node is running ;)
[ INFO] [1626178478.285735025]: Closing Socket Communication ...
[ WARN] [1626178479.287904061]: ur3 is setted in ip 192.168.1.56
[ WARN] [1626178479.287987280]: ur3 is setted in port 30003
[ INFO] [1626178484.288815973]: Opening Socket Communication ...
[ INFO] [1626178485.481423549]: UR3 was resetted !
[ UR3 ] [1626178485.481497000]: Communication Interface node is running ;)
[ INFO] [1626178532.917976842]: Closing Socket Communication ...
[ WARN] [1626178533.920110138]: ur3 is setted in ip 192.168.1.56
[ WARN] [1626178533.920184362]: ur3 is setted in port 30003
[ INFO] [1626178538.920799657]: Opening Socket Communication ...
[ INFO] [1626178540.119212121]: UR3 was resetted !
[ UR3 ] [1626178540.119275000]: Communication Interface node is running ;)
[ERROR] [1626178577.114141147]: The connection with ur3 was broken. Please, reset the Interface
Communication
[ WARN] [1626178635.598781723]: ur3 is setted in ip 192.168.1.56
[ WARN] [1626178635.598858587]: ur3 is setted in port 30003
```

Figura A.19: Mensagem de erro

Para reconectar a Interface de comunicação no **Linux PC** basta rodar o seguinte comando

**rosservice call /ur3/reset "{}"**

## A.4 Desligamento do sistema

### A.4.1 Certificar se o robô está na posição HOME

Caso o braço manipulador não esteja na posição HOME, como mostra a Figura A.20, use o comando de **reset** em alguma seção disponível do Terminator.

Comando **reset** mostrado logo abaixo para por o UR3 na posição HOME.

**rosservice call /ur3/reset "{}"**

Figura A.20: Braço manipulador UR3 na posição HOME

### A.4.2 Desligamento do polyscope

Simplesmente apertar o botão on-off do Polyscope Figura A.21 que é mostrada logo abaixo.

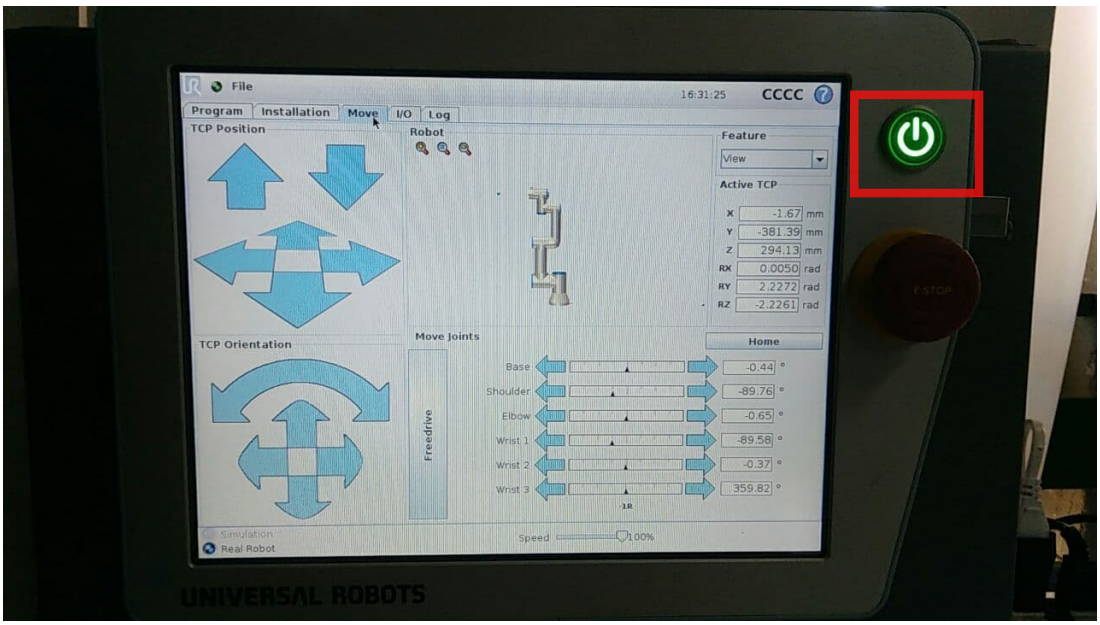


Figura A.21: Botão on-off no desligamento

Aparecerá um POPUP chamado *shutdown* similar ao da Figura A.22.

Pressione o botão com a marcação retangular em vermelho chamado **Power off**.

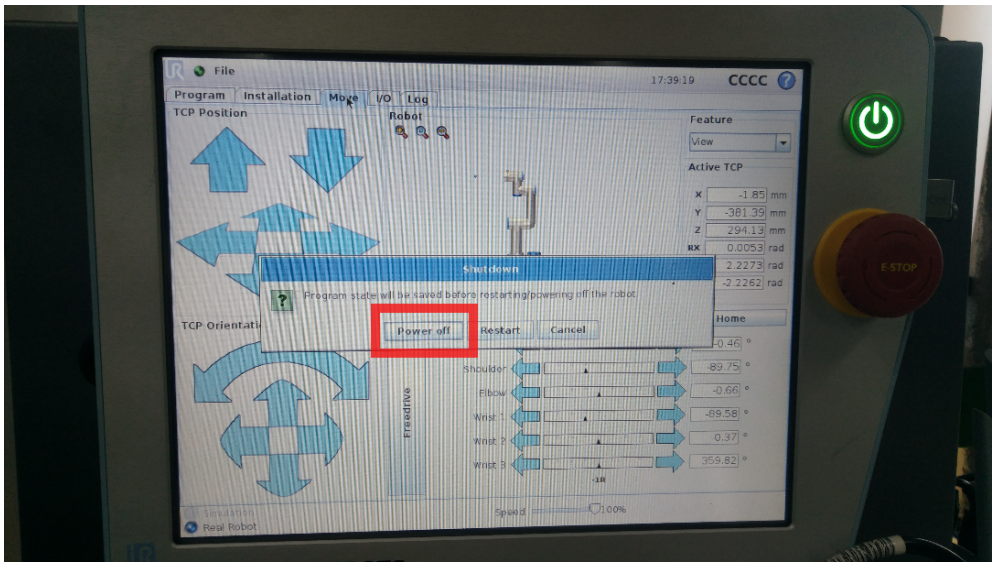


Figura A.22: POPUP shutdown

Feito isso o robô será desligado.

### A.4.3 Desligamento da Interface de Comunicação Linux PC

#### A.4.3.1 Na seção 1 (que roda a interface de comunicação) usar o comando Crtl+C pelo teclado.

#### A.4.3.2 Fechar o Terminator como qualquer outro aplicativo Lynux.

### A.4.4 IMPORTANTE!!

Antes de ir embora, manter o sistema sempre como na foto abaixo.

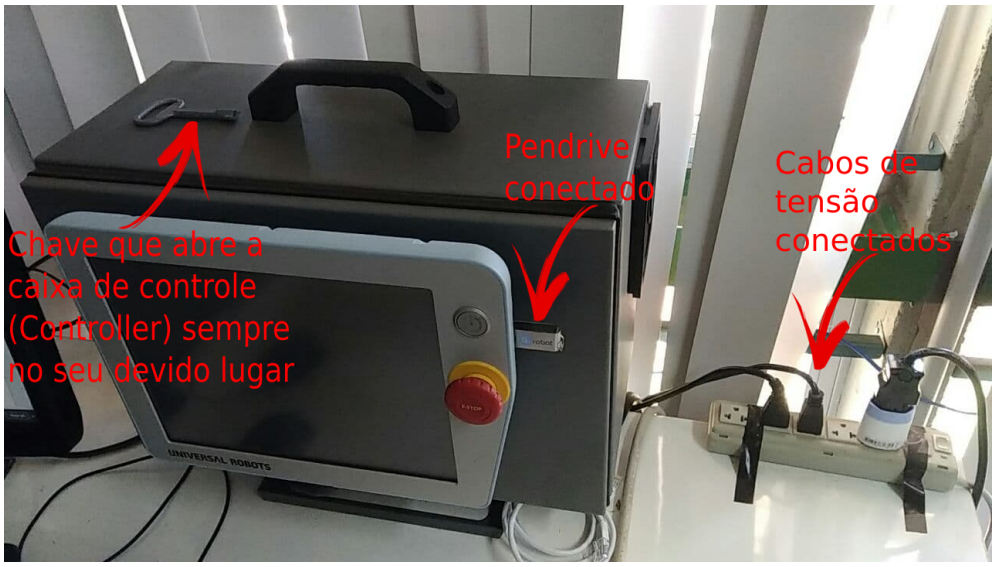


Figura A.23: Cabos de tensão, pendrive e chave

#### A.4.4.1 Não retirar o pendrive

O pendrive fornece os drivers da garra. O fabricante recomenda deixar o pendrive sempre conectado ao polyscope

#### A.4.4.2 Não retire a chave do lugar

A chave é utilizada por usuários avançados para acessar a caixa de controle (Controller). Não abra a caixa de controle a menos que saiba exatamente o que está fazendo.

#### A.4.4.3 Manter os cabos no lugar

Manter os cabos de rede como nas fotos A.24, A.25 e A.26

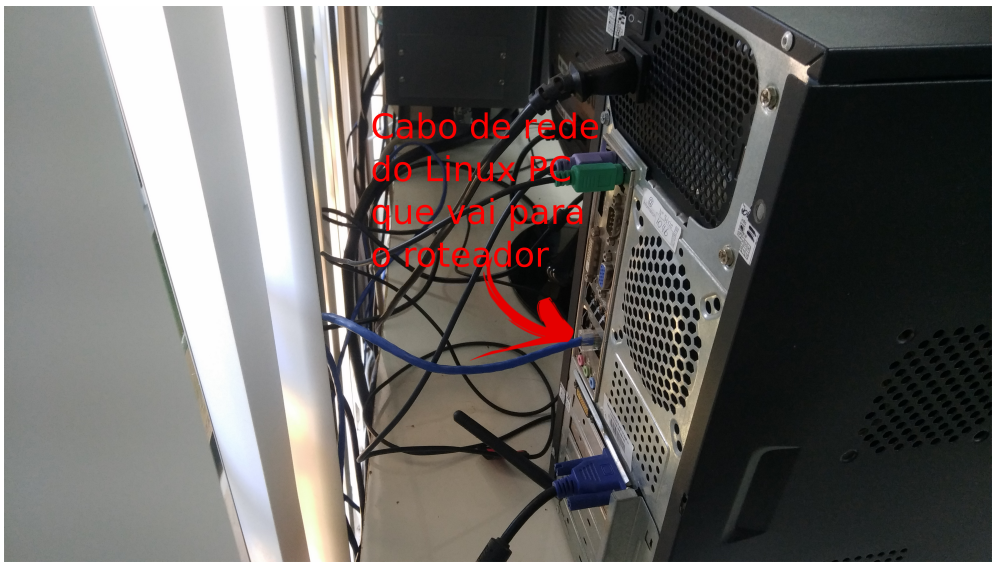


Figura A.24: Cabo de rede do linux PC


Figura A.25: Cabo de rede do Controller

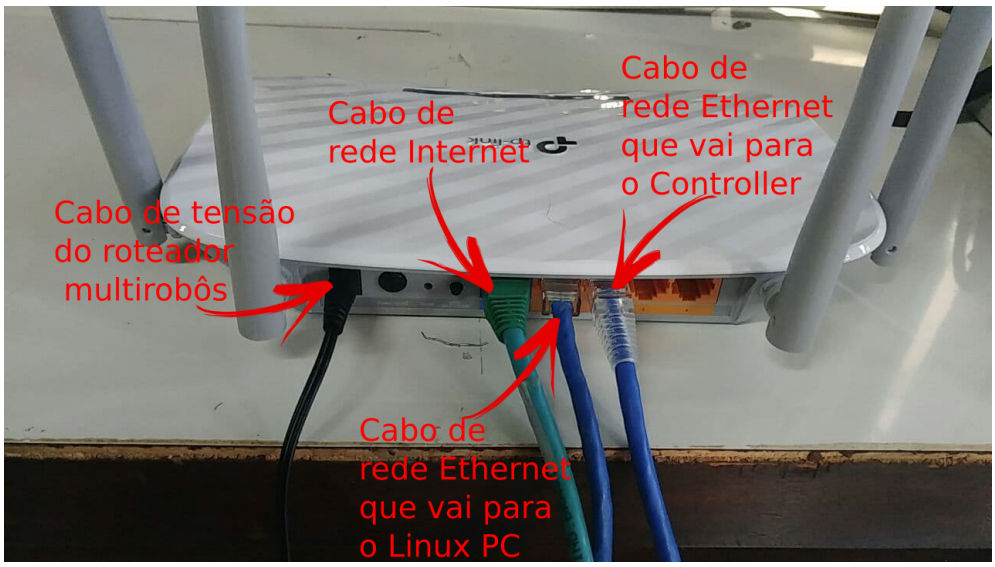


Figura A.26: Cabos de rede do roteador

# Apêndice B

# Experimento 2: Usando controlador do Capítulo 4

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

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

**cd ~/catkin_ur3_ws**

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

**source devel/setup.zsh**.

Como explicado na seção A.1.2, Interface de Comunicação está em funcionamento, ou seja, ela já está capturando os dados dos estados das juntas do robô UR3, publicando no tópico **/ur3/arm** e esperando uma referência de velocidade no tópico **/ur3/ref_vel**.

## B.1 Gerando velocidades de referência para o UR3

Neste secão vamos explicar o processo de como gerar uma referência de velocidade para umas das juntas do robô UR3. Para isso, será necessário que o usuário tenha em mãos a ferramenta Matlab para rodar o código mostrado abaixo (o Matlab está instalado no computador reservado para o UR3 ).

### B.1.1 Explicação do *script* de geração de velocidade

```
% Arquivo para gerar referencia de velocidade para as juntas do robo UR3
clear ;
close all;
periodo = 2; % Em pi rad
taxa_de_amostragem = 125; % Em Hz do processo
duracao = 4; %  Tempo de duracao da onda de entrada em pi segundos
amplitude = 0.5; % Amplitude da onda de referencia em rad
% Foi usado a funcao senoide, mas pode ser a funcao de sua preferencia.
% Evite gerar referencias que tenham trasicoes rapidas
tempo = (0:(1/taxa_de_amostragem):duracao*pi);
ref_vel = amplitude*sin(periodo*tempo);
% ref_vel ser  a vari vel que cont m os dados de velocidade para o robo
figure('units','normalized','outerposition',[0 0 1 1]);
plot(tempo, ref_vel,'LineWidth',1.5); % plotando a referencia de velocidade
grid on ;
ylabel('Amplitude(rad)'); xlabel('tempo(s)')
lg = legend('Refer ncia de velocidade'); lg.FontSize = 11;
ti = title('Onda senoidal de refer ncia'); ti.FontSize = 20;
csvwrite('ref_vel.csv', ref_vel);% escrevendo em um arquivo do tipo csv
```

Nesse código existem alguns parâmetros que o usuário pode alterar conforma sua necessidade. São eles:

- Período da função seno (linha 4 do código matlab)
- O tempo de duração da referência de velocidade (linha 6 do código matlab)
- A amplitude do sinal (linha 5 do código matlab)
- A função geradora do sinal (linha 11 do código matlab) que nesse caso é a função seno.

Após rodar esse *script* Matlab, será mostrado uma imagem semelhante Figura B.1 onde é plotado os dados de referência de velocidade.

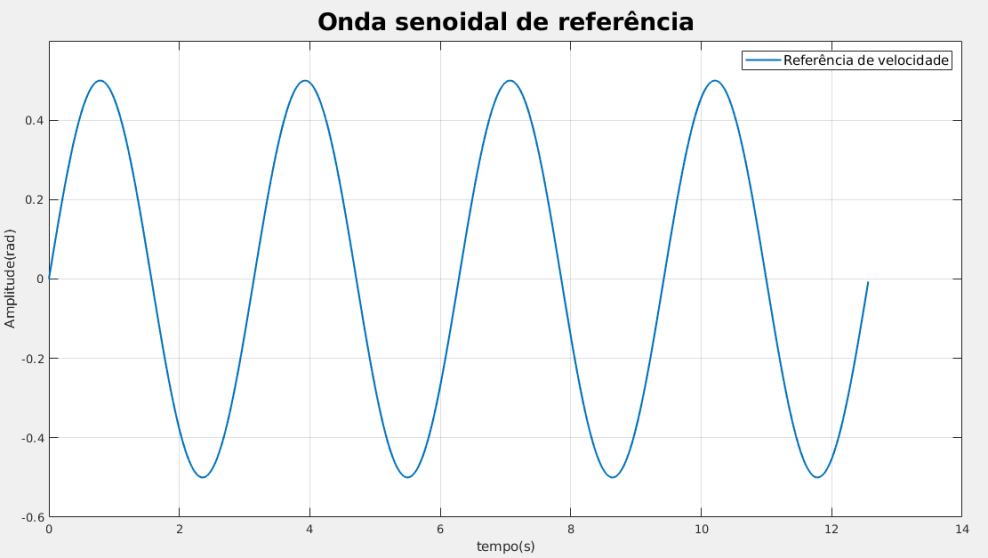


Figura B.1: Referência de Velocidade

Na linha 19 do *script* Matlab é onde acontece a criação do arquivo que será usado pela Interface de Comunicação 2.2.3 , que nesse caso chama-se **ref_vel.csv**, e pode ser visto na pasta que é usado no Matlab exemplificada na FiguraB.2 .


Figura B.2: ref_vel.csv no diretório do Matlab

### B.1.2 Movendo o aquivo CSV para pasta csv do pacote demos_ur3

Dentre os pacotes da Interface de Comunicação, foi projetado um pacote que é específico para enviar referências de velocidade para o UR3. Para isso, basta utilizar o arquivo .csv, gerado no passo anterior, e colocá-lo dentro do pacote **demos_ur3** na pasta **cvc** (pasta selecionada na Figura B.3). Para realizar todo o manuseio do robô e da Interface de Comunicação para o experimento de envio e coleta de dados, basta seguir os passos do Apêndice A.

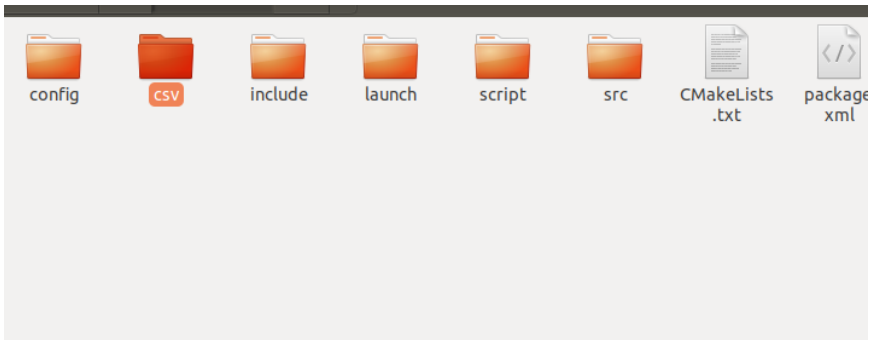


Figura B.3: Pasta csv selecionada

### B.1.3 Alterando o arquivo de setup

Agora vamos informar para o pacote **demos_ur3**, nas configurações, que o arquivo que queremos que seja lido para mandar as referências de velocidade, seja o arquivo que acabamos de criar na seção anterior B.1.2. Para isso, basta ir na pasta **config** (pasta selecionada na Figura B.4) do pacote **demos_ur3** e mudar a variável **csv_name** que se encontra no aquivo **setup1.yaml** para **ref_vel.csv** como mostra a Figura B.5.

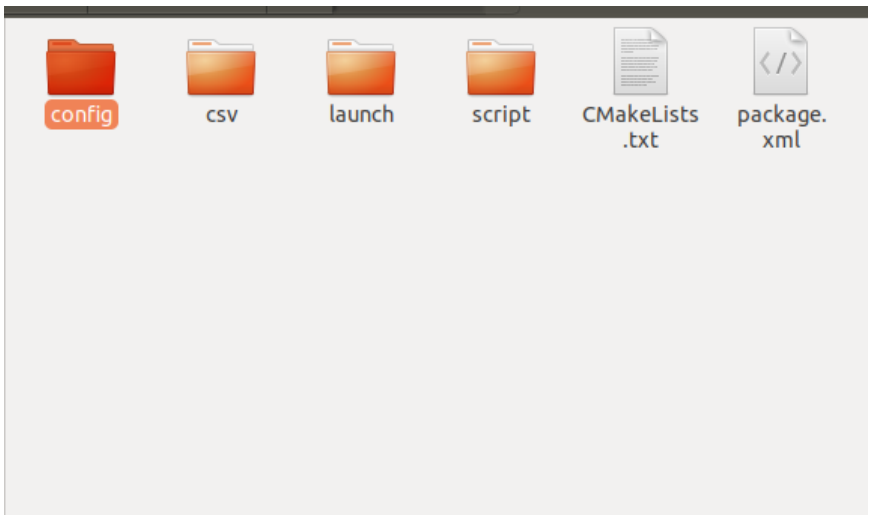


Figura B.4: Pasta config selecionada

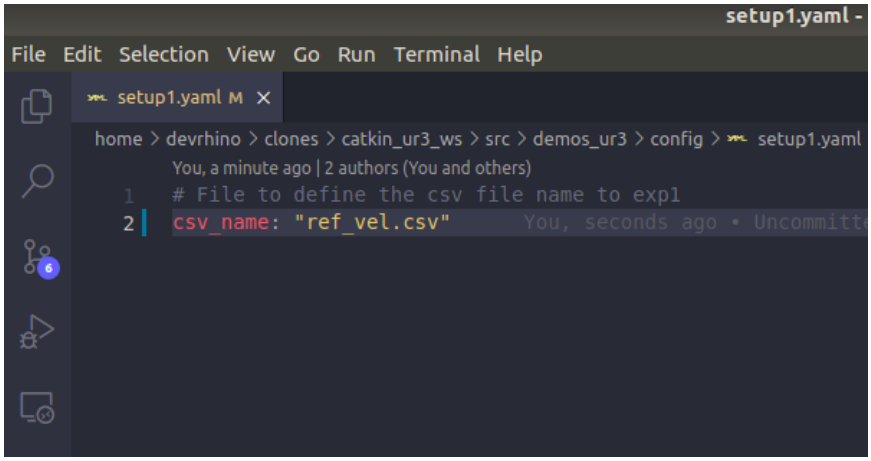


Figura B.5: Arquivo setup1

Feito isso, basta executar os mesmos comandos do Apêndice A referente a inicialização da Interface de Comunicação e execução do experimento.

## B.2 Usando o controlador

Neste experimento o usuário tem a opção de enviar ondas de referência de posição de forma offline, como é feito no experimento 1 mostrado no Apêndice A, ou mandar o comando especificando a posição da junta via terminal.

No arquivo **setup2.yaml** ( Figura B.7), localizado na pasta **config** (pasta selecionada na Figura B.6) do pacote **demos_ur3**, há alguns parâmetros que serão comentados em itens a seguir.

- **csv_name**: No parâmetro **csv_name** o leitor deve escrever no sub-parâmetro **your_wave**

o nome de um arquivos **csv** que tenha ondas de referência de posição geradas, explicado no seção B.1, para realizar seu próprio experimento. Os sub-parâmetros **sine** e **square** não devem ser alterados, pois foram feitos para fazer demostrações de funcionamento do UR3.

- **ref_type**: No parâmetro **ref_type** o usuário decide se quer que as mensagens de referência, para as juntas, sejam enviadas online ou offline, ou seja, caso o sub-parâmetro **online** seja dado com **True** o usuário enviará a referência de posição pelo terminal. Por outro lado, caso o sub-parâmetro **online** seja dado com **False**, o sub-parâmetro **type** irá informar qual onda de referência de posição será enviada para o UR3, que na Figura B.7 mostra que é a senoidal.

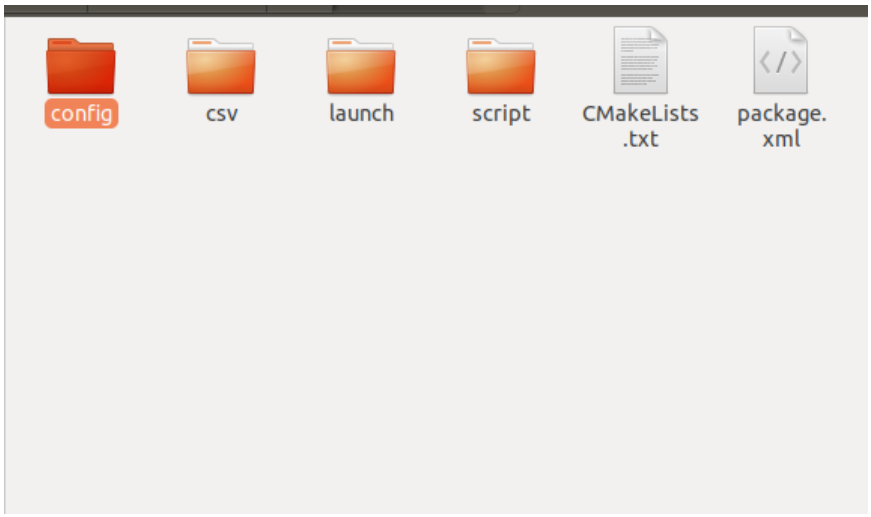


Figura B.6: Pasta config selecionada

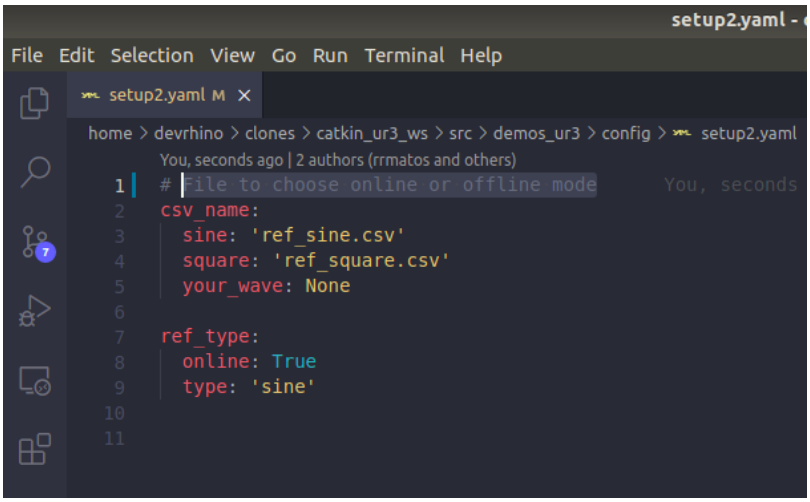


Figura B.7: Arquivo de setup para o experimento 2

### B.2.1 Experimento offline

No experimento **offline** o usuário deve colocar o sub-parâmetro **online** como **False**, escolher qual onda de referência será enviada ao UR3 e colocá-la no sub-parâmetro **type**. As opções de onda de referência são:

- **sine**: Onda de referência será uma senoidal.
- **square**: Onda de referência será uma onda quadrada.
- **your_wave**: Uma onda gerada pelo próprio usuário.

Para este experimento foi feito, em **Python**, um nó chamado **exp2** em ROS para publicar as referências de velocidade para o robô. O arquivo em questão é chamado de **demo_exp2.py**, pertencente ao pacote **demos_ur3**. Para ser executado, abra outra aba no terminal, vá para o **workspace** e digite o comando que torna essa aba um diretório ROS.

**source devel/setup.zsh**

Na sequência inicie o nó do experimento com o comando a seguir.

**roslaunch demos_ur3 exp2.launch**

Neste ponto o nó **Exp2** está rodando e esperando a chamada do serviço **/demos_ur3/start_exp2**.

Para chamar o serviço **/demos_ur3/start_exp2** abra outra aba no terminal, vá para o **workspace** e digite o comando que torna essa aba um diretório ROS

**source devel/setup.zsh**

e rode o comando de chamada do serviço com a flag **data** como **true**.

**rosservice call /demos_ur3/start_exp2 "data: true"**

Para parar o experimento basta chamar o serviço **/demos_ur3/start_exp2** com a flag **data** como **false** ou esperar o experimento acabar.

**rosservice call /demos_ur3/start_exp2 "data: false"**

### B.2.2 Experimento online

No experimento **online** o usuário deve colocar o sub-parâmetro **online** como **true**. Feito isso, o usuário não precisa se preocupar com os outros parâmetros.

Para este experimento foi feito, em **Python**, um nó chamado **exp2** em ROS para publicar as referências de velocidade para o robô. O arquivo em questão é chamado de **demo_exp2.py**, pertencente ao pacote **demos_ur3**, e para ser executado, abra outra aba no terminal, vá para o **workspace** e digite o comando que torna essa aba um diretório ROS.

**source devel/setup.zsh**

Na sequencia inicie o nó do experimento com o comando a seguir.

**roslaunch demos_ur3 exp2.launch**

Neste ponto o nó **Exp2** está rodando e esperando a chamada do serviço **/demos_ur3/start_exp2**.

Para chamar o serviço **/demos_ur3/start_exp2** abra outra aba no terminal, vá para o **workspace** e digite o comando que torna essa aba um diretório ROS

**source devel/setup.zsh**

e rode o comando de chamada do serviço com a flag **data** como **true**.

**rosservice call /demos_ur3/start_exp2 "data: true"**

O envio de referência via terminal também é feito via serviço do ROS. O nome do serviço que envia os dados de referência de posição é **/demos_ur3/target_position** e nele o leitor teve passar como argumento do parâmetro **newFloat** a posição, em radianos, para a junta do UR3.

O comando a seguir é um exemplo de como o serviço é chamado passando uma posição de referência de 0.2 rad.

**rosservice call /demos_ur3/target_position "newFloat: 0.2"**

Para parar o experimento basta chamar o serviço **/demos_ur3/start_exp2** com a flag **data** como **false** ou esperar o experimento acabar.

**rosservice call /demos_ur3/start_exp2 "data: false"**

Depois disso é esperado que o robô UR3 faça um movimento repetitivo parecido com o movimento do seguinte vídeo que está postado no youtube:

`https://youtu.be/pkN1GS0wX_o`

# Apêndice C

# Experimento 3: Visualização dos dados dos estados das juntas, sinais de entrada e gravação dos dados

## C.1 Visualização dos estados das juntas do UR3

Este experimento objetiva fornecer um primeiro contato do usuário ao sistema robótico do manipulador UR3. Ele permitirá que o usuário seja capaz de

- Visualização usando rqt_plot [33];
- Gravação dos dados em um arquivo rosbag [24];
- Exportação dos dados gravados para o matlab [30];

Assumindo que o leitor tenha feito o processo de setup do ambiente de experimento, como é demonstrado no Apêndice A da seção A.1.1 até a seção A.1.2, vamos direto para o experimento em si.

Como explicado na seção A.1.2, Interface de Comunicação está em funcionamento, ou seja, ela já está capturando os dados dos estados das juntas do robô UR3, publicando no tópico **/ur3/arm** e esperando uma referência de velocidade no tópico **/ur3/ref_vel**.

## C.2 Visualização usando rqt_plot

Abra uma nova seção no **Terminator** e rode o comando abaixo para ir para o diretório do **workspce**.

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

**cd ~/catkin_ur3_ws**

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

**source devel/setup.zsh**.

Em seguida, vamos rodar o comando abaixo para abrir **rqt_plot**. Uma janela idêntica a Figura C.1 será aberta.

**rqt_plot**.

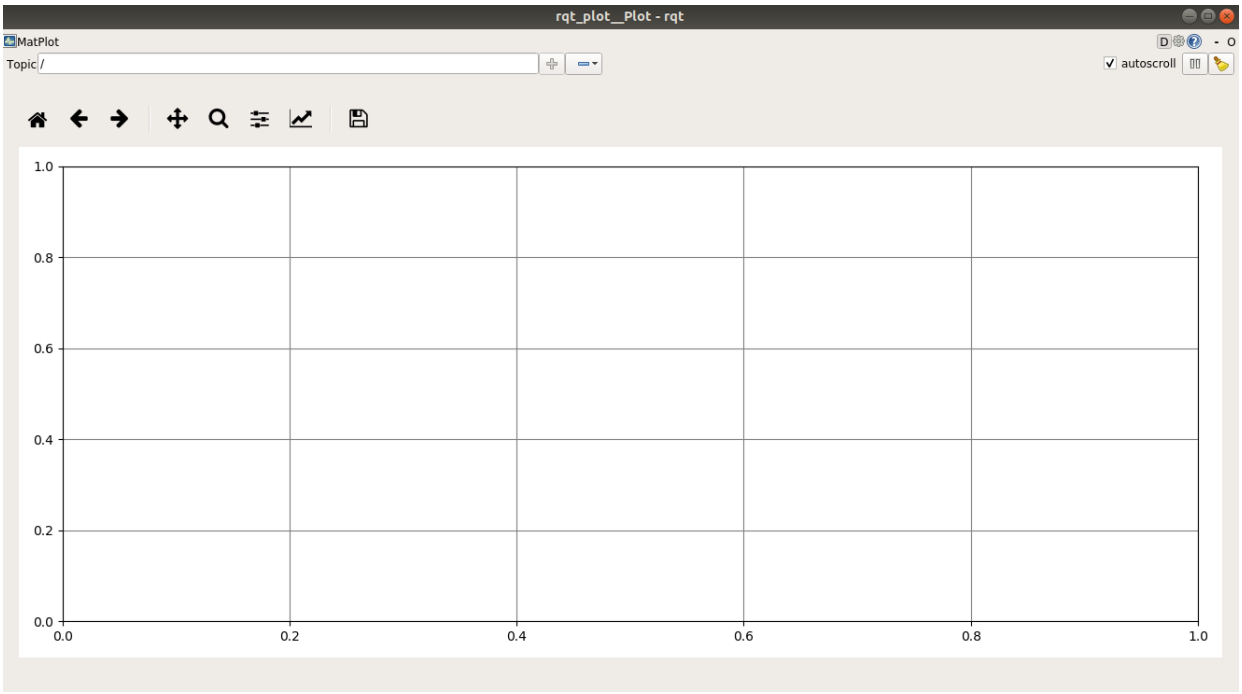


Figura C.1: rqt_plot

A interface gráfica do **rqt_plot** funciona de forma similar a um osciloscópio. O eixo das ordenadas faz a medida do sinal que é colocado como entrada (será explicado como colocar um sinal para visualização mais adiante) e o eixo das abscissa faz a medida do tempo em segundos.

### C.2.1 Visualização dos estados das juntas do UR3

Como já descrito na seção 2.2.3.2, existe um tópico chamado de **/ur3/arm** (mostrado na Figura C.2 usando o comando **rostopic list**) onde podemos acessar os estados das juntas do UR3 (Posição, velocidade e Torque).

Agora, usando a interface do **rqt_plot**, vamos acessar os estados de uma das juntas, usando as legendas abaixo, sendo colocado a referência para o tópico na caixa de texto em vermelho chamada **Topic** Figura C.3.

- **i = 0**: junta **Base**

- **i = 1**: junta **Shoulder**
- **i = 2**: junta **Elbow**
- **i = 3**: junta **Wrist1**
- **i = 4**: junta **Wrist2**
- **i = 5**: junta **Wrist3**

- **Posição da junta i, com i = 0 a 5**: /ur3/arm/position[i]
- **Velocidade da junta i, com i = 0 a 5**: /ur3/arm/velocity[i]
- **Toque da junta i, com i = 0 a 5**: /ur3/arm/effort[i]

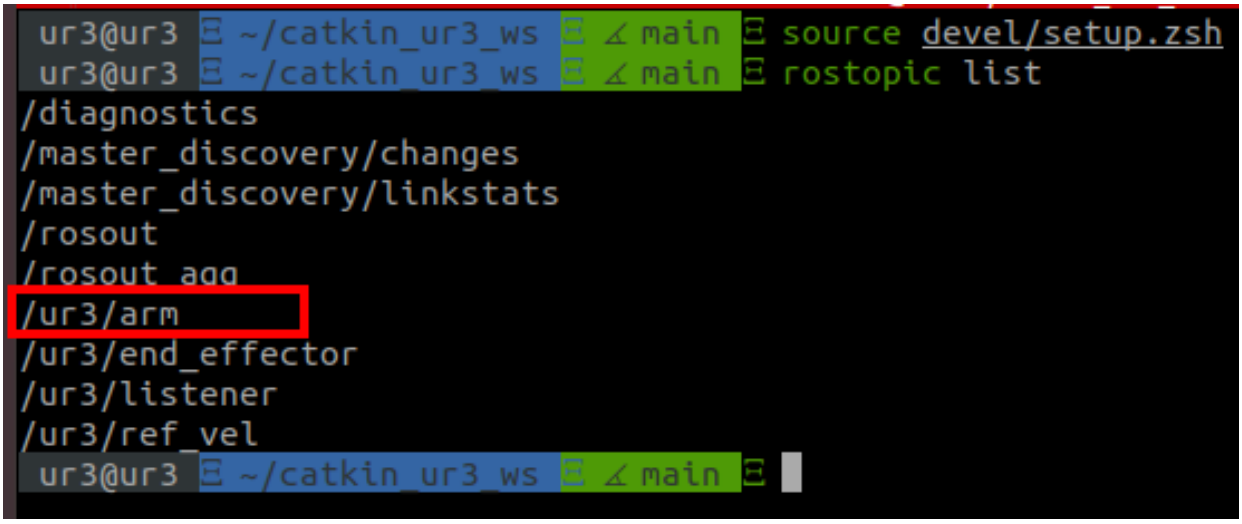


Figura C.2: Tópico /ur3/arm

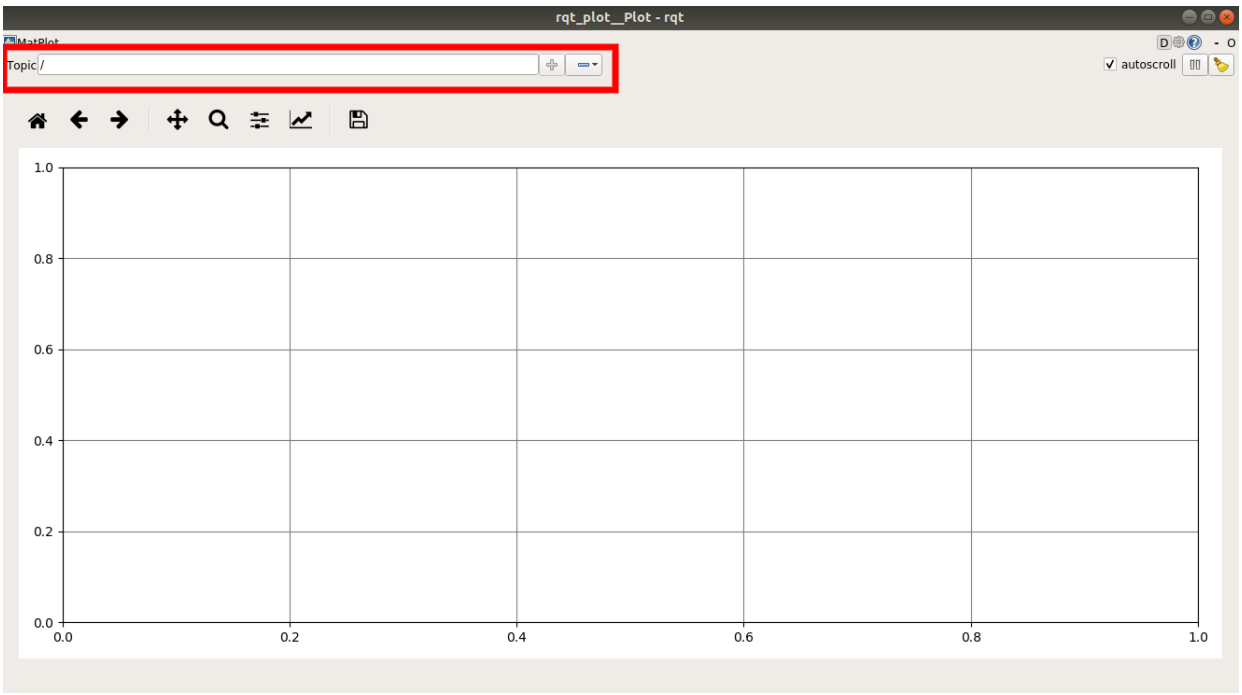


Figura C.3: caixa de texto Topic

Depois de adicionar uma das referências para uma das ondas, que no exemplo da Figura C.4 é a posição da **Base**, clique no botão demarcado pelo retângulo verde na Figura C.4.

Como a junta não está em movimento, o sinal aparecerá imóvel como mostra a Figura C.5.

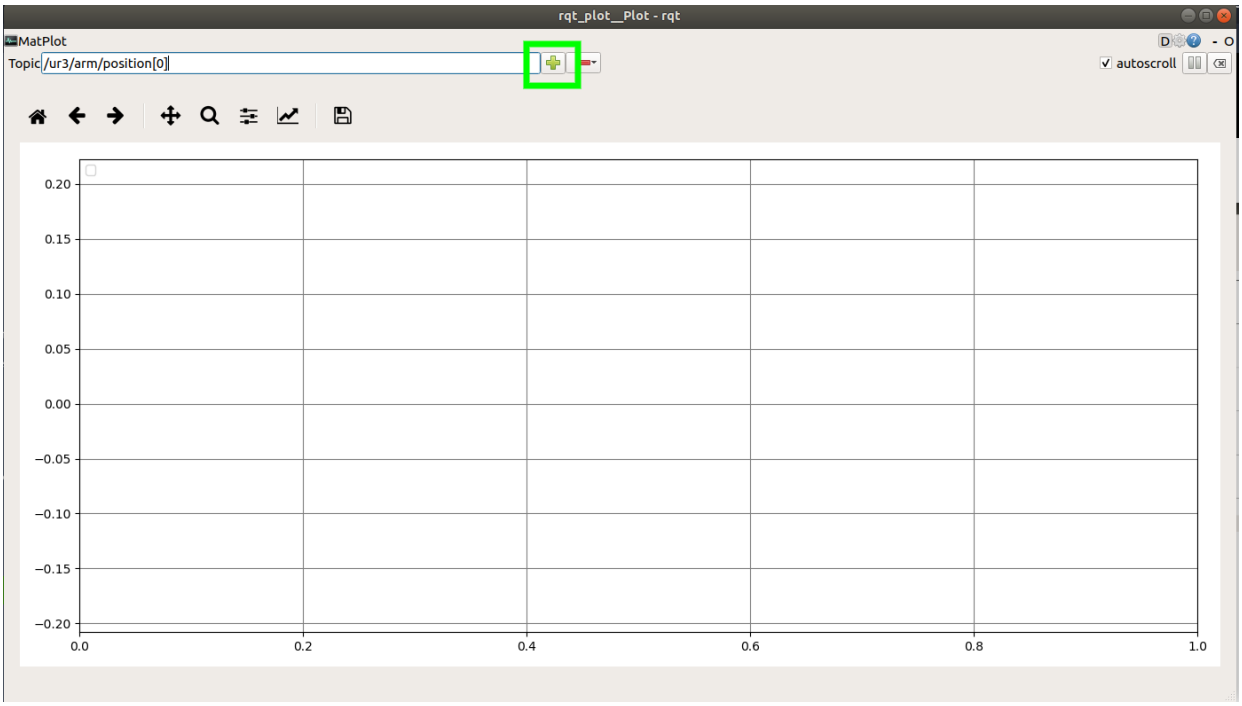


Figura C.4: Adicionando uma onda no rqt_plot

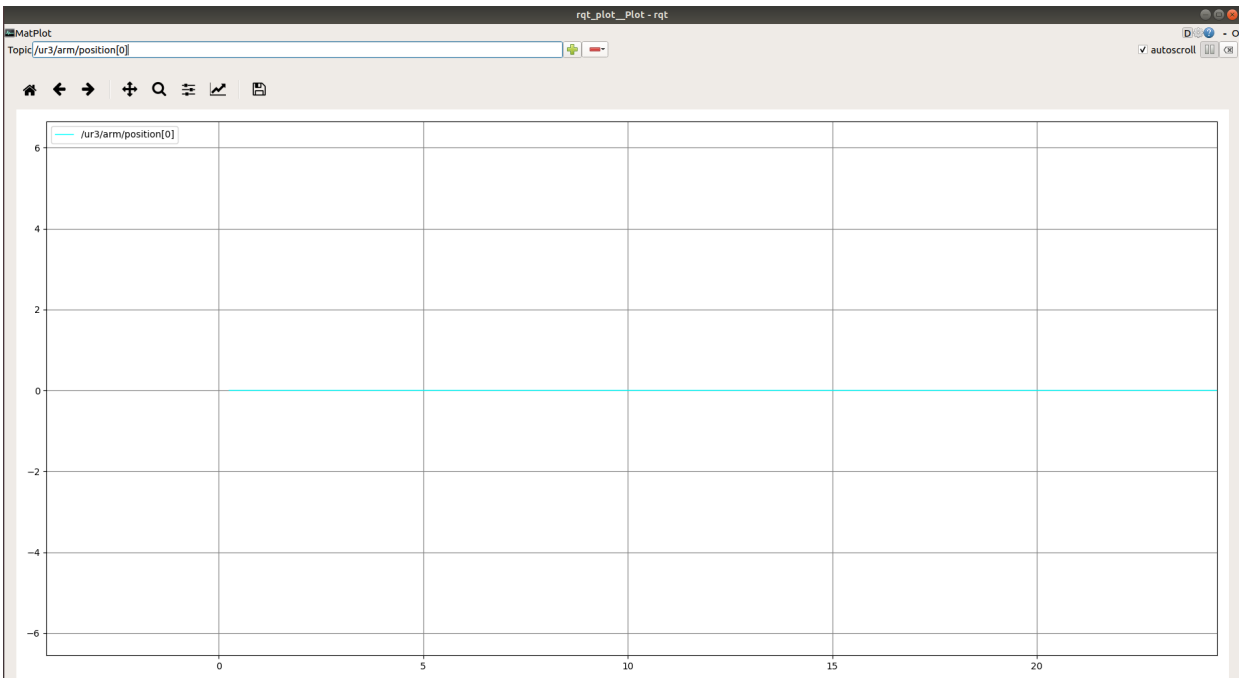


Figura C.5: Sinal de posição da Base Adicionado

Seguindo o mesmo processo usado para adicionar o sinal de posição, vamos adicionar os estados de velocidade e torque da junta da base. Para isso, basta escrever o comando abaixo na caixa de texto **Topic** e clicar no ícone mais.

Lembre-se, só é permitido adicionar um sinal por vez, ou seja, adicione o sinal de velocidade e depois adicione o sinal de torque.

**ur3/arm/velocity[0]**
**ur3/arm/effort[0]**

Depois de adicionar os sinais que foram mencionados nos parágrafos acima, abra uma nova seção no terminator e rode os comandos a seguir para fazer a junta **Base** iniciar um movimento.

**source devel/setup.zsh**

Na sequência inicie o nó do experimento com o comando a seguir.

**roslaunch demos_ur3 exp3.launch**

Neste ponto o nó **Exp3** está rodando e esperando a chamada do serviço **/demos_ur3/start_exp3**.

Para chamar o serviço **/demos_ur3/start_exp2** abra outra aba no terminal, vá para o **workspace** e insira o comando que torna essa aba um diretório ROS

**source devel/setup.zsh**

e rode o comando de chamada do serviço com a flag **data** como **true**.

**rosservice call /demos_ur3/start_exp3 "data: true"**

Para parar o experimento basta chamar o serviço **/demos_ur3/start_exp3** com a flag **data** como **false** ou esperar o experimento acabar.

**rosservice call /demos_ur3/start_exp3 "data: false"**

Depois desse processo o usuário verá uma imagem semelhante a da Figura C.6.

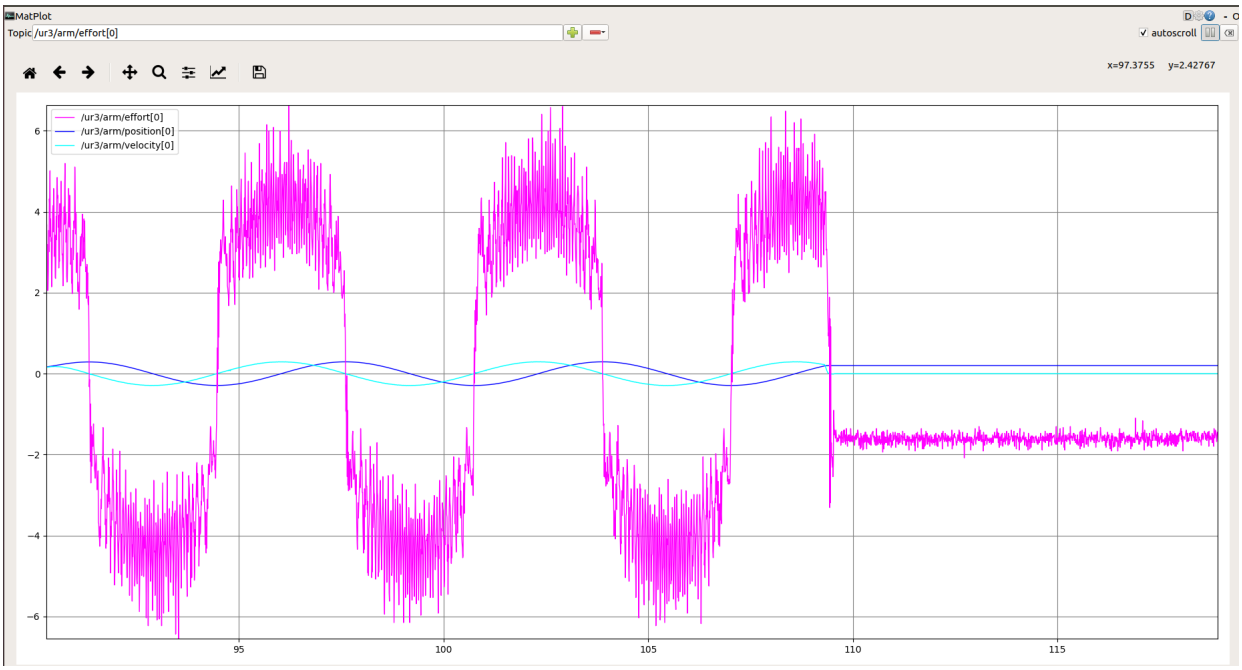


Figura C.6: Sinais de saída para junta Base

### C.2.2 Visualização dos sinais de entrada

Assim como para visualizar os sinais de saída do UR3, visualizar o sinais de entrada da junta **Base** (referência de posição e sinal de controle que é uma referência de velocidade para o controlador interno do UR3), basta ir no **rqt_plot** e adicionar o sinal referente ao tópico **/ur3/ref_pos** e o sinal referente ao tópico **/ur3/ref_vel**.

Para isso, escreva os comandos abaixo na caixa de texto **Topico** do **rqt_plot** e clique no ícone mais.

**/ur3/ref_pos/data[0]**
**/ur3/ref_vel/data[0]**

Depois desse processo o usuário verá uma imagem semelhante a da Figura C.7.

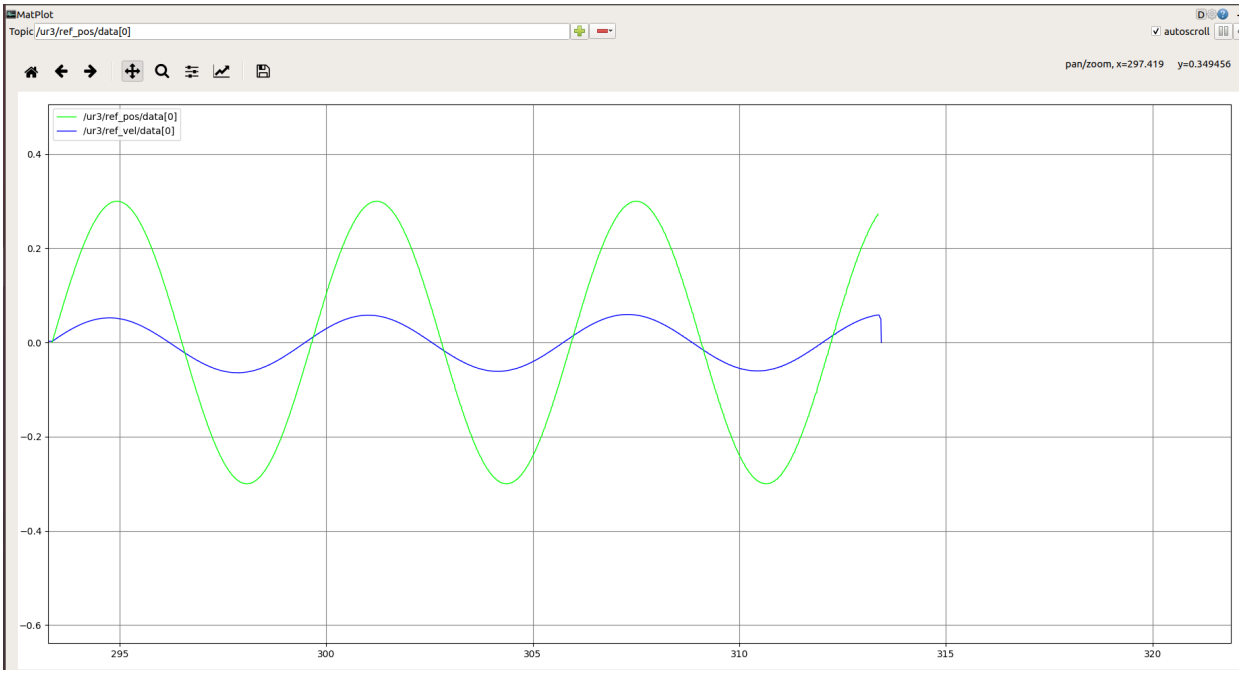


Figura C.7: Sinais de entrada para junta Base

Caso o usuário insira todas as ondas (entrada e saída da junta Base), ele verá uma imagem semelhante a da Figura C.8.

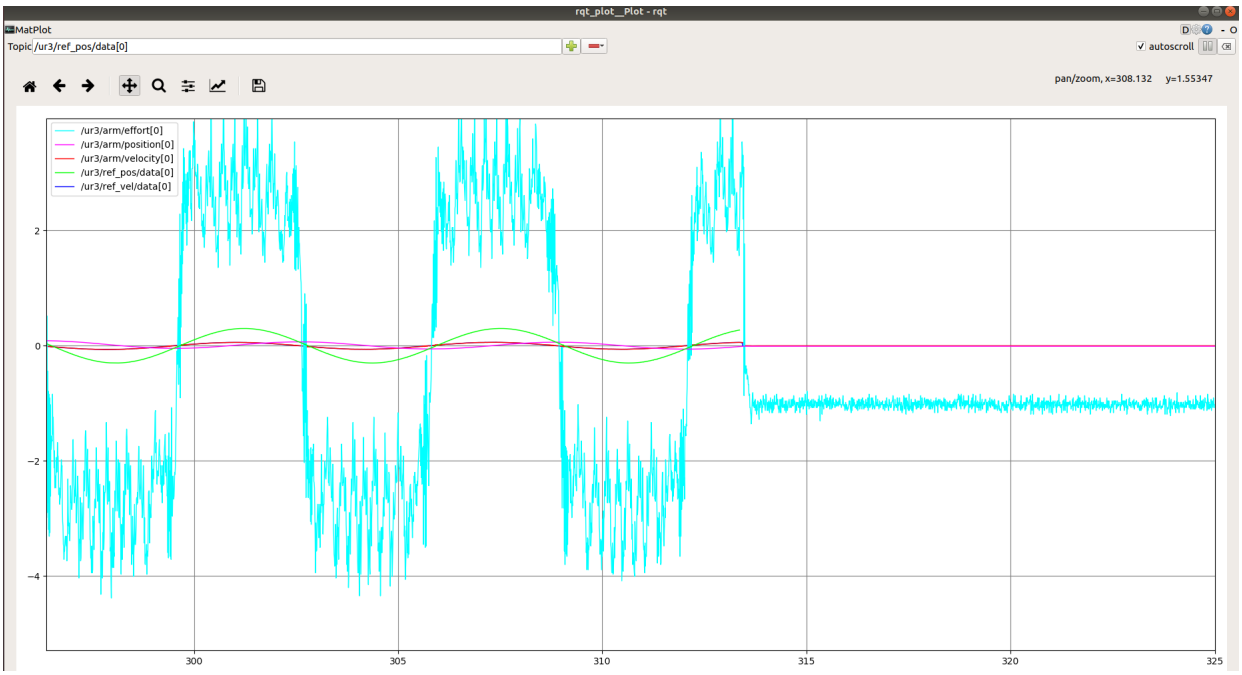


Figura C.8: Sinais de entrada e saída para junta Base

### C.2.3 Gravação dos dados em um arquivo rosbag

Para realizar o gravação dos dados em um sistema de arquivo **bag**, primeiro vamos esclarecer qual os tópicos ROS que são de interesse apara uma analise mais geral dos sinais de entrada e saída do UR3.

Nos itens abaixo, encontra-se a discrição de cada tópico de interesse. Para mais detalhe, consulte o capítulo referente a Interface de Comunicação 2.2.3.2.

- **ur3/arm**: nesse tópico está contido todos os sinais de saída referente as juntas do UR3 (Posição, Velocidade e Torque).
- **ur3/ref_pose**: o tópico **ur3/ref_pos** armazeno os os sinais de referência de posição para todas as juntas do UR3.
- **ur3/ref_vel**: o tópico **ur3/ref_vel** armazeno os os sinais de referência de velocidade para todas as juntas do UR3.
- **/ur3/end_effector**: este tópico armazena os dados do end_effector do UR3, são eles:
  - **abertura da pinça do end_effector**
  - **posição no espaço tridimensional com relação a base do UR3**
  - **orientação no espaço tridimensional com relação a base do UR3**
  - **velocidade linear no espaço tridimensional com relação a base do UR3**

Como dito antes, os dados gravados serão armazenados em um arquivo **bag**[24] e serão armazenados, com o prefixo **exp**, na pasta **bags** (Figura C.9) presente no **workspace catkin_ur3_ws** com a nomenclatura sendo prefixo mais a data e hora de realização do experimento.

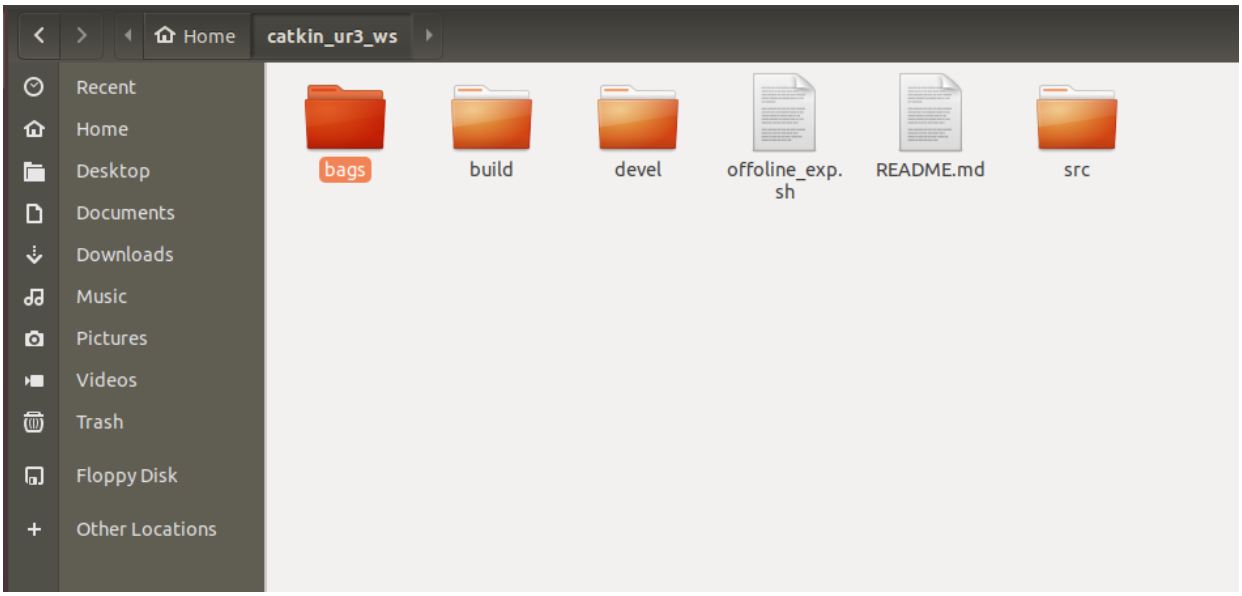


Figura C.9: Pasta bags selecionada

Para gravar os dados do experimento, abra uma nova seção no terminator, vá para o **workspace** e insira o comando que torna essa aba um diretório ROS.

**source devel/setup.zsh**

Inicie a gravação dos dados com o comando abaixo.

**rosbag record -o bags/exp /ur3/arm /ur3/ref_pos /ur3/ref_vel /ur3/end_effector**

Para parar a gravação dos dados, pressione Ctrl+C na mesma seção do terminator que foi iniciado a gravação. Depois disso, aparecerá um arquivo com os dados do experimento como mostrado na FiguraC.10 (nesse exemplo experimento foi realizado na data 25/09/2021 às 10:47:32)

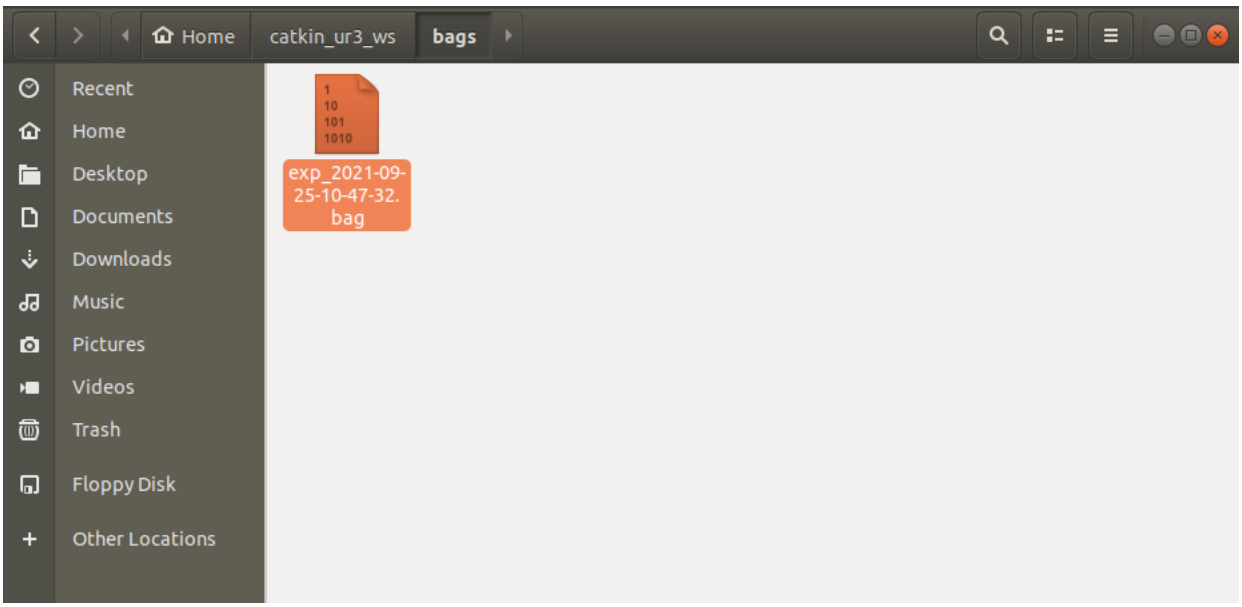


Figura C.10: Pasta bags selecionada

### C.2.4 Exportação dos dados gravados para o matlab

Para fazer a exportação dos dados gravados pelo ROS para o **matlab** da forma mais simples possível, foi feito um script na linguagem matlab para agilizar o processo de exportação de dados.

O script matlab é nomeado com **export_rosbag** e se encontra na pasta **matlab_scripts** dentro do **workspace** como mostra a Figura C.11.

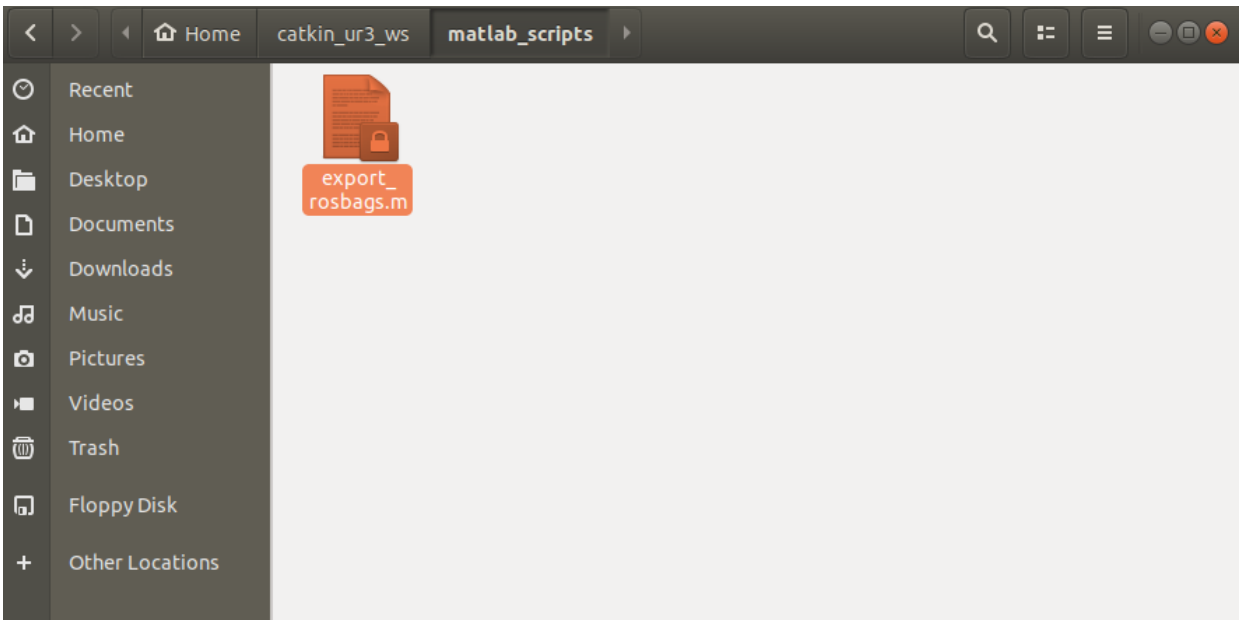


Figura C.11: Export rosbag

Para fazer a exportação dos dodos para o matlab, abra o matlab e rode o script **matlab_scripts** no formato descrito no comando abaixo.

**[time, input_pose, input_vel, output_pose, output_vel, output_effort, end_effector] = export_rosbags('exp_2021-09-25-10-47-32')**

A FiguraC.12 mostra como é a execução do comando acima (retângulo em vermelho) e as variáveis exportadas (retângulo em verde).

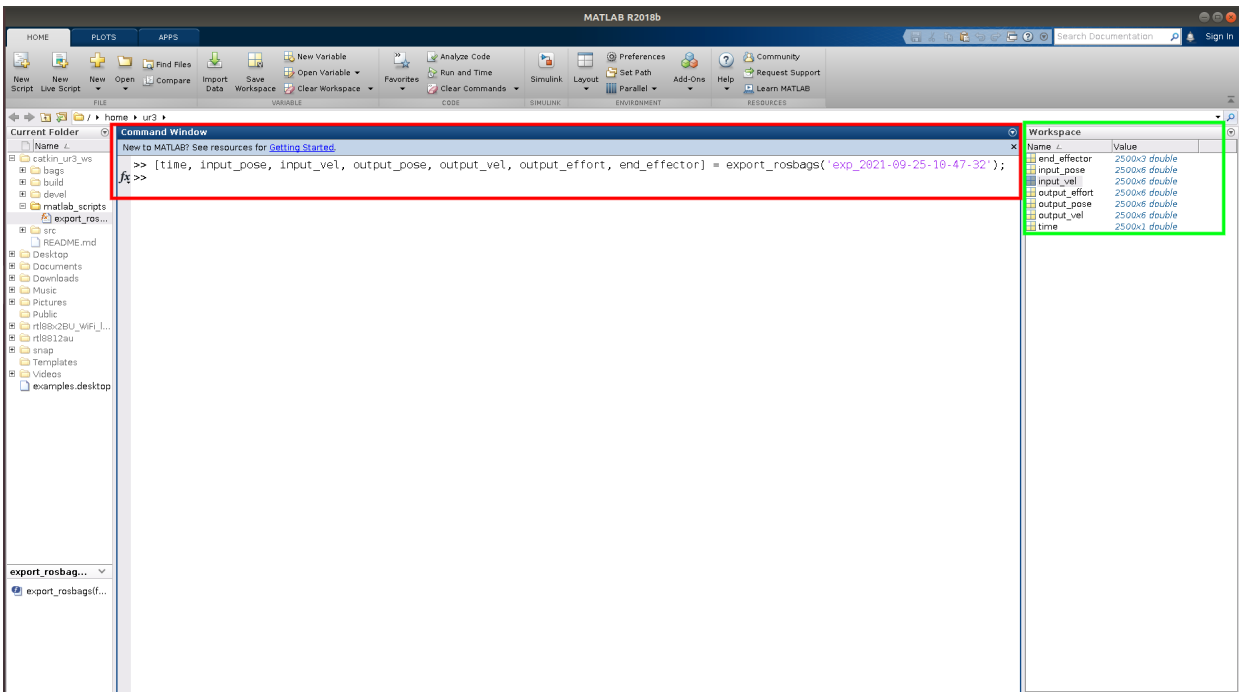


Figura C.12: Comando para exporta os dados para o matlab

Descrição de cada variável do workspace do matlab (retângulo verde na Figura C.12).

- **time**: vetor com os valores de tempo discretizado a cada 0.008 segundos.
- **input_pose**: vetor entrada de posição angular de tamanho 6, em radianos, para cada junta do UR3, sendo o primeiro elemento a posição da junta da base.
- **input_vel**: entrada de velocidade angular de tamanho 6, em rads/s, para cada junta do UR3, sendo o primeiro elemento a velocidade da junta da base.
- **output_pose**: saída de posição angular de tamanho 6, em radianos, de cada junta do UR3, sendo o primeiro elemento a posição da junta da base.
- **output_vel**: saída de velocidade angular de tamanho 6, em rads/s, de cada junta do UR3, sendo o primeiro elemento a velocidade da junta da base.
- **output_effort**: saída de torque de tamanho 6, em N.m, de cada junta do UR3, sendo o primeiro elemento a torque da junta da base.
- **end_effectror**: vetor de de saída da posição tridimensional do end_effector com tamanho 3 com relação a base do UR3. A primeira posição do vetor é a coordenada x, a segunda y e a terceira a coordenada z, todas com a unidade em metros.

# Apêndice D

# Experimento 4: Como construir um controlador Avanço Atraso usando python

## D.1 Introdução

Nesse experimento vamos descrever como o usuário pode construir um controlador incorporando a lei de controle vista no Capítulo 4. A lei de controle abordada no Capítulo 4, descrita na equação 4.15 e replicada como a equação D.1, tem como entrada o erro de posição e como saída (sinal de controle), uma referência de velocidade para as juntas do UR3.

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

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

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

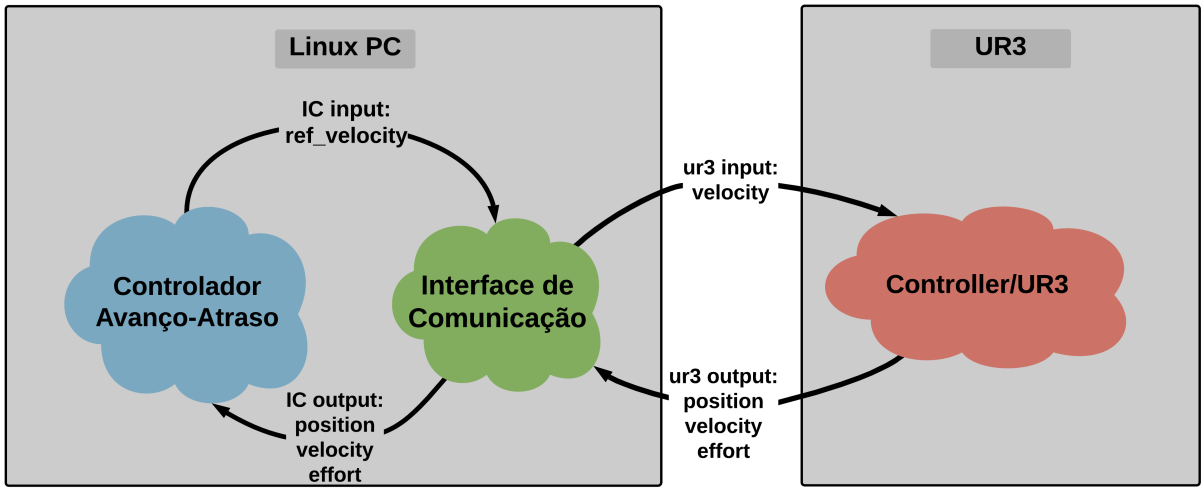


Figura D.1: Fluxo de dados entre o controlador a Interface de Comunicação e o UR3

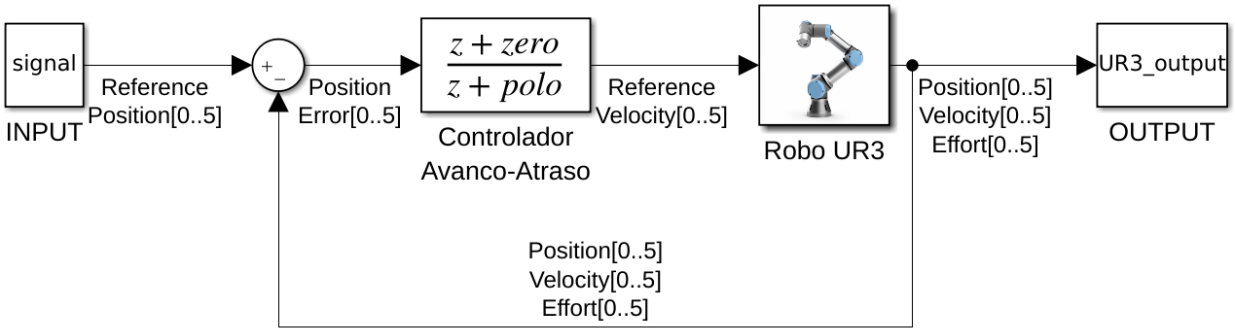


Figura D.2: Esquemático da malha de controle

## D.2 Pacote ur3_controls

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

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

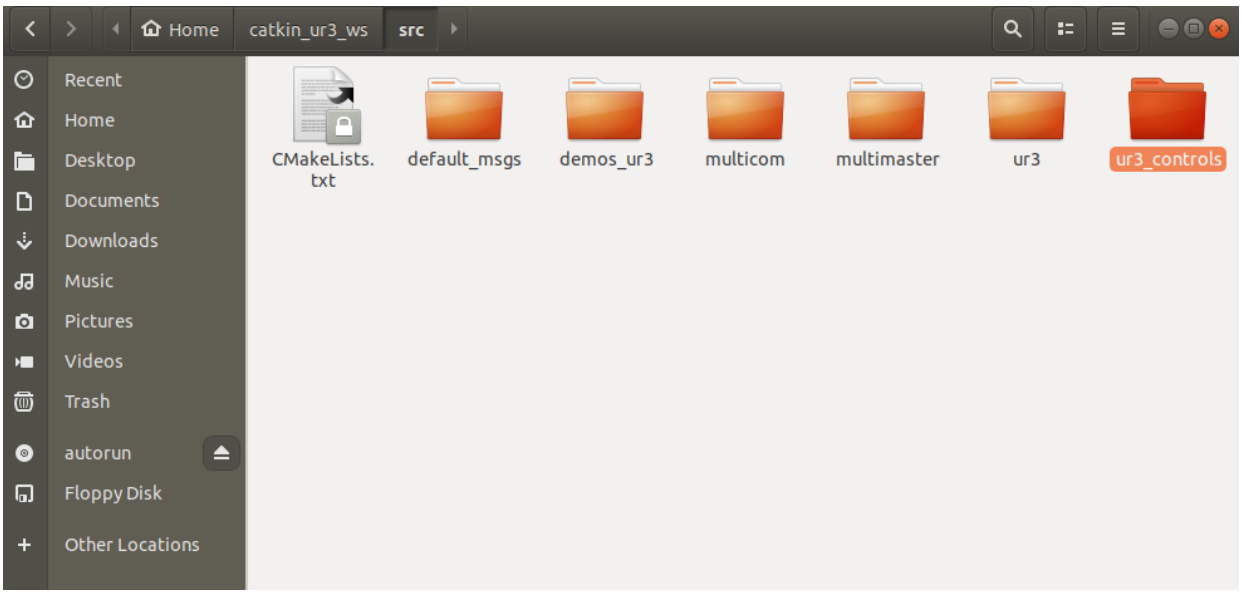


Figura D.3: Pasta selecionada mostrando o pacote ur3_controls

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

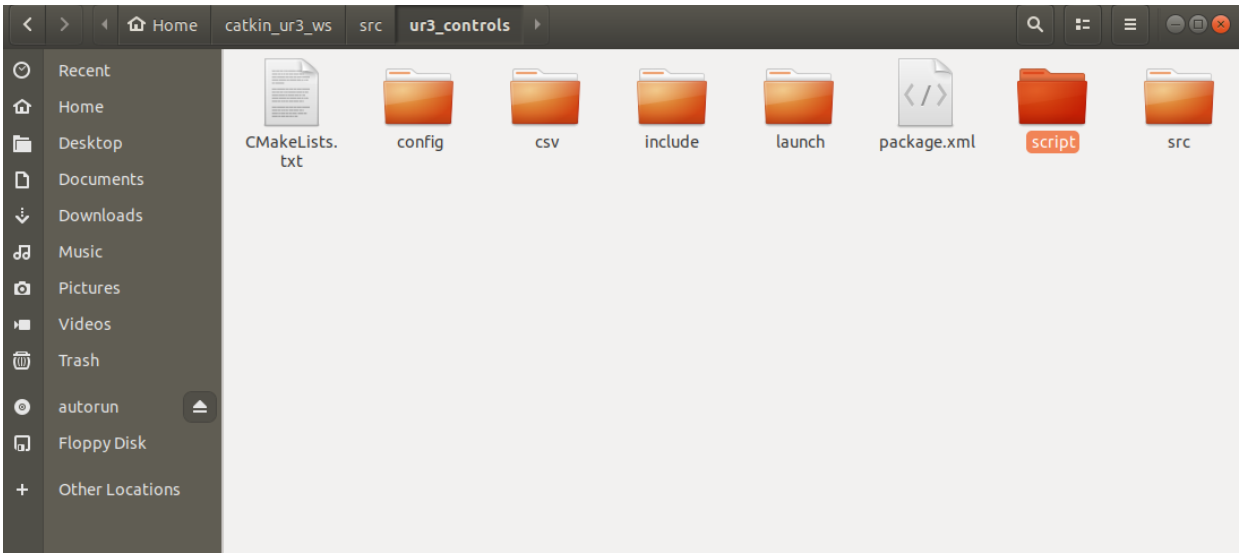


Figura D.4: Pasta script selecionada

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

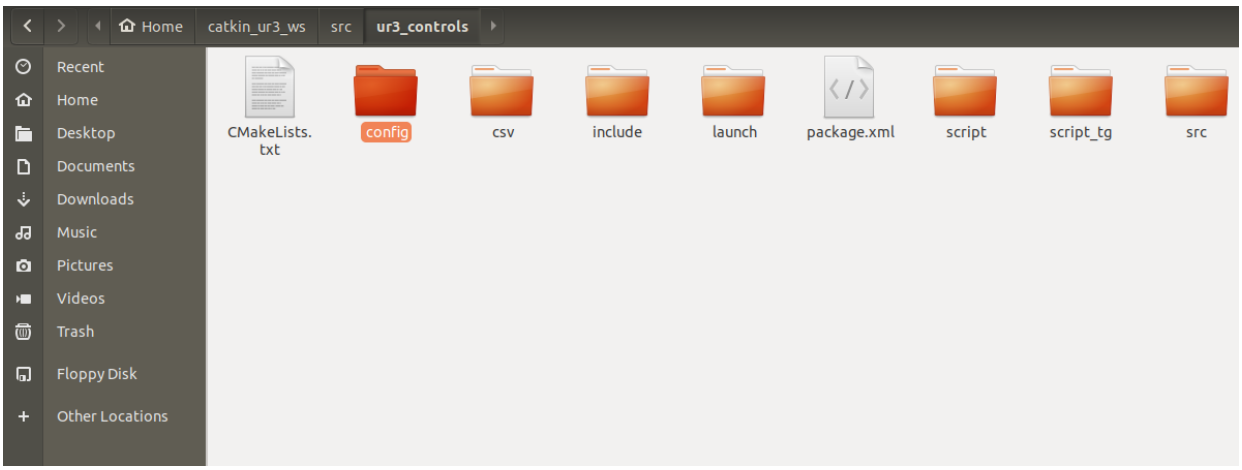


Figura D.5: Pasta config selecionada

A Figura D.6 mostra dentro da pasta **script**. Dentro dela há um arquivo que não pode ser alterado (o arquivo **main.py**). O arquivo chamado de **controle_avaco_atraso.py** será onde iremos implementar o controlador descrito na equação 4.15.

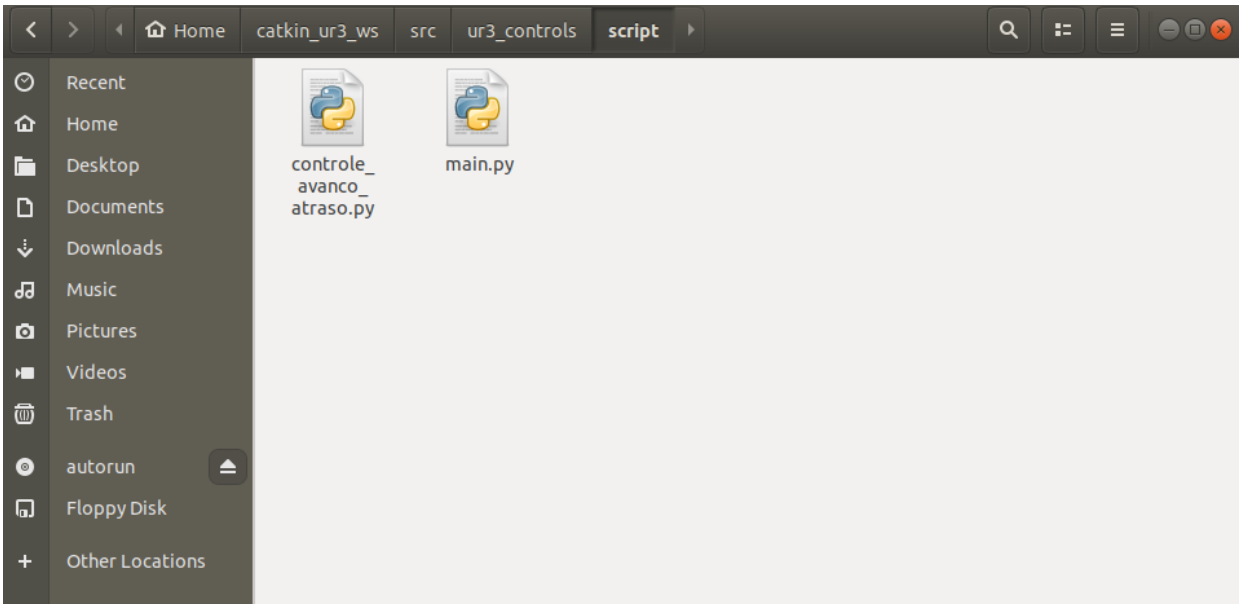


Figura D.6: Dentro Pasta script

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

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

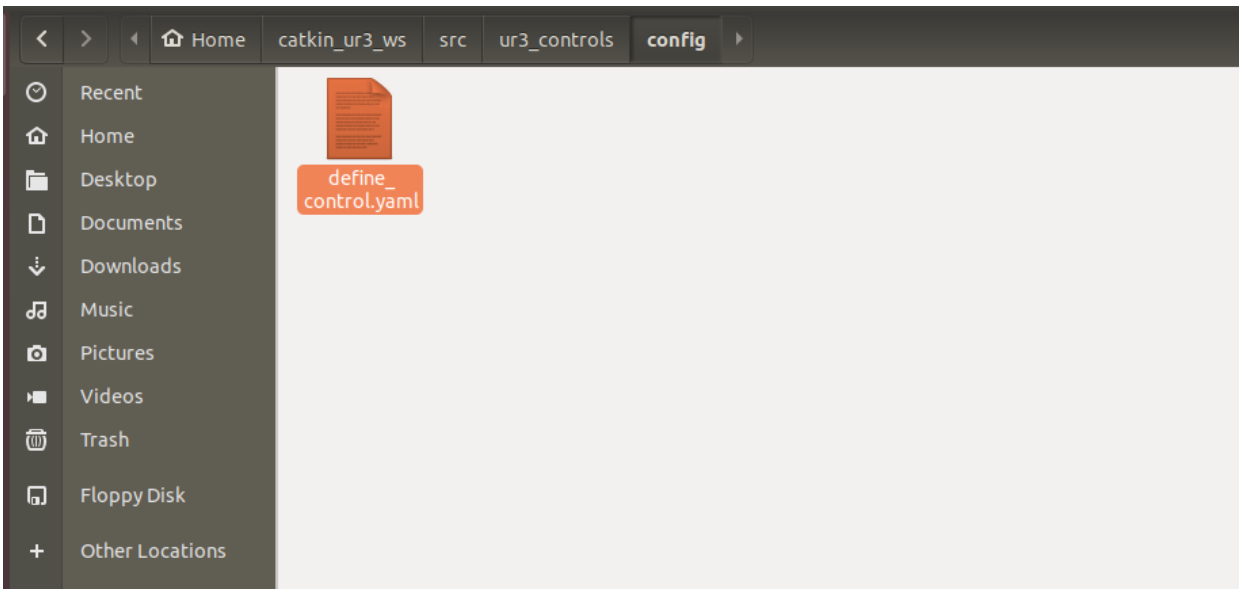


Figura D.7: Dentro Pasta script

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

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

### D.3.1 Inicio do código e a classe

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

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

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

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

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

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

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

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

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

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

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

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

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

### D.3.2 Lei de Controle

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

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

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

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

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

## D.4 Estrutura do arquivo de configuração

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

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

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

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

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

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

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

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

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

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

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

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

### D.4.1 Arquivo de configuração define_control.yaml

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

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

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

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

### D.4.2 Código completo do controlador Avanço-Atraso

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

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

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

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

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

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

## D.5 Como rodar o controlador proposto nesse experimento

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

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

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

**cd ~/catkin_ur3_ws**

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

**source devel/setup.zsh**.

### D.5.1 Experimento offline

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

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

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

**roslaunch ur3_controls main.launch**.

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

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

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

**cd ~/catkin_ur3_ws**

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

**source devel/setup.zsh**.

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

A junta **wrist3** começará a se mover

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

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

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

### D.5.2 Experimento online

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

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

**roslaunch ur3_controls main.launch**.

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

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

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

**cd ~/catkin_ur3_ws**

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

**source devel/setup.zsh**.

Inicialize o controlador com o comando abaixo.

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

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

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

**cd ~/catkin_ur3_ws**

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

**source devel/setup.zsh**.

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

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

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

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

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

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

Caso queria parar o experimento, envie o comando abaixo.

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

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

**rosrun rqt_gui rqt_gui -s reconfigure**.

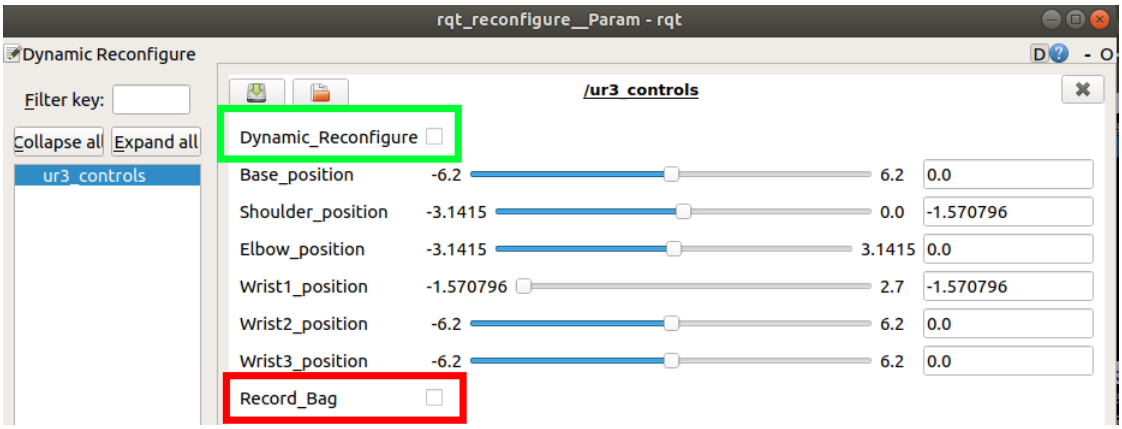


Figura D.8: Interface ur3_contros

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

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