Centro Federal de Educação Tecnológica de Minas Gerais
Campus Divinópolis
Graduação em Engenharia Mecatrônica

Thalles Oliveira Campagnani

Arranjo Cliente Servidor para um Robô Industrial com Controladora Aberta

Divinópolis
2022

Thalles Oliveira Campagnani

Arranjo Cliente Servidor para um Robô Industrial com Controladora Aberta

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação do curso de Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.
Eixo de Formação: Computação.

Orientador: Prof. Dr. Renato de Sousa Dâmaso

Divinópolis
2022

Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG / Divinópolis
Curso de Engenharia Mecatrônica

Monografia intitulada “Arranjo Cliente Servidor para um Robô Industrial com Controladora Aberta ”, de autoria do graduando Thalles Oliveira Campagnani, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Renato de Sousa Dâmaso

Prof. Dr. Daniel Alves Costa

Prof. Dr. Adriano Nogueira Drumond Lopes

Coordenador do Curso de Engenharia Mecatrônica

Prof. Dr. Marlon Antônio Pinheiro

Divinópolis

Maio de 2022

Dedico essa monografia a classe trabalhadora, a qual trabalho prol de um dia automatizarmos completamente os meios de produção com objetivo de libertar o ser humano do trabalho alienante, monótono e repetitivo, deixando esse trabalho para as máquinas, de forma que no futuro, quando os meios de produção pertencerem aos trabalhadores, vivamos em uma sociedade em que trabalhemos apenas com trabalho criativo "de cada um segundo a sua capacidade", e recebamos "a cada um segundo a sua necessidade".

Agradecimentos

Agradeço, ao CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico - pelo apoio financeiro dado a essa pesquisa através da concessão de bolsa do PIBTI - Programa Institucional de Bolsas de Desenvolvimento Tecnológico e Inovação. Agradecemos também a COMAU Robotics pelo suporte técnico e orientações dadas. E por fim ao CEFET-MG por ceder o espaço físico e equipamentos para o desenvolvimento das atividades.

A anarquia econômica da sociedade capitalista como existe hoje é, na minha opinião, a verdadeira fonte do mal. Estou convencido de que existe apenas um caminho para eliminar esses graves males, e esse é o estabelecimento de uma economia socialista, acompanhada por um sistema educacional orientado para objetivos sociais.

Albert Einstein

Resumo

O presente trabalho teve o objetivo de investigar a possibilidade de se fornecer mais opções para o desenvolvimento de aplicações para um robô industrial com controladora aberta disponível no laboratório de Robótica da unidade de Divinópolis do CEFET-MG. Para isso, foi realizado o desenvolvimento de um arranjo cliente servidor, onde o programa servidor deve ser executado no computador industrial da controladora aberta do robô. O trabalho também prevê a possibilidade do desenvolvimento de aplicações no programa cliente em qualquer linguagem de programação, qualquer arquitetura e qualquer sistema operacional, para ser executado em um terceiro dispositivo. Dessa forma, o desenvolvimento desse arranjo cliente servidor deverá estender a liberdade que a solução da fabricante proporciona aos programadores de seus robôs industriais. Para tanto, foi necessário desenvolver: um protocolo de comunicação que utiliza o TCP/IP para enviar e receber variáveis referentes aos ângulos das juntas do robô; um algorítimo que armazene na memória os ângulos de referência das juntas do robô oriundos do programa no terceiro dispositivo e que envie os dados sensoriais armazenados na memória para esse programa; um algoritmo que use a biblioteca fornecida pela fabricante para ler dados sensoriais, salvando-os na memória e enviando referências das juntas do robô armazenadas na memória para a controladora; resolver o problema gerado pelo assincronismo entre a comunicação do arranjo e controladora e entre a comunicação do arranjo e do programa no terceiro dispositivo. Realizou-se, então, uma pesquisa de finalidade aplicada, objetivo exploratório, sob o método hipotético-dedutivo, com abordagem quantitativa, realizada com procedimentos experimentais. Diante disso, verificou-se que foi possível realizar a comunicação a uma taxa estável máxima de 2 ms, programar e executar o exemplo de programa cliente em qualquer linguagem de programação, arquitetura e sistema operacional escolhidos. Isso levou à constatação de que o objetivo de estender a liberdade que a solução da fabricante proporciona aos programadores de robôs industriais em sua controladora aberta foi atingido.

Palavras-chave: Robô Industrial, Controladora aberta, Programação de manipulador, Arranjo cliente servidor.

Abstract

The present work aimed to investigate the possibility of providing more options for the development of applications for an industrial robot with an open controller available in the Robotics laboratory of the CEFET-MG unit in Divinópolis. For this, the development of a client-server arrangement was carried out, where the server program must be executed in the industrial computer of the open controller of the robot. The work also foresees the possibility of developing applications in the client program in any programming language, any architecture and any operating system, to be executed in a third device. Thus, the development of this client-server arrangement should extend the freedom that the manufacturer's solution provides to the programmers of its industrial robots. Therefore, it was necessary to develop: a communication protocol that uses TCP/IP to send and receive variables referring to the robot joint angles; an algorithm that stores in the memory the reference angles of the robot joints coming from the software in the third device and that sends the sensorial data stored in the memory to that software; an algorithm that uses the library provided by the manufacturer to read sensory data, saving it in memory and sending the robot joint references stored in memory to the controller; solve the problem generated by the asynchronism between array and controller communication and between array and software communication on the third device. Then, a research of applied purpose, exploratory objective, was carried out under the hypothetical-deductive method, with a quantitative approach, carried out with experimental procedures. Therefore, it was verified that it was possible to communicate at a maximum stable rate of 2 ms, program and run the client software example in any chosen programming language, architecture and operating system. This led to the realization that the objective of extending the freedom that the manufacturer's solution provides to industrial robot programmers in its open controller was achieved.

Keywords: Industrial Robot, Open controller, Manipulator programming, Client server arrangement.

Sumário

Lista de figuras

Lista de acrônimos e notações

API	Interface de Programação de Aplicativo, do inglês *Application Programming Interface*
CNC	Controle Numérico Computadorizado, do inglês *Computer Numeric Control*
eORL	Biblioteca Robótica Aberta Realística Melhorada, do inglês *enhanced Open realistic Robot Library*
EPL	Ethernet PowerLink
ISO	Organização Internacional de Normalização, do inglês *International Organization for Standardization*
LTS	Suporte de longa duração, do inglês *Long Term Support*
Open LPC	Computador Pessoal com Linux do Open, do inglês *Open Linux Personal Computer*
PC	Computador pessoal, do inglês *Personal computer*
ROS	Sistema Operacional Robótico, do inglês *Robot Operation System*
SBC	Computador de placa única, do inglês *Single board computer*
TCP/IP	Protocolo de Controle de Transmissão / Protocolo de Internet, do inglês *Transmission Control Protocol / Internet Protocol*
TP	Terminal de programação, comumente chamado de *teaching pendant*

Capítulo 1

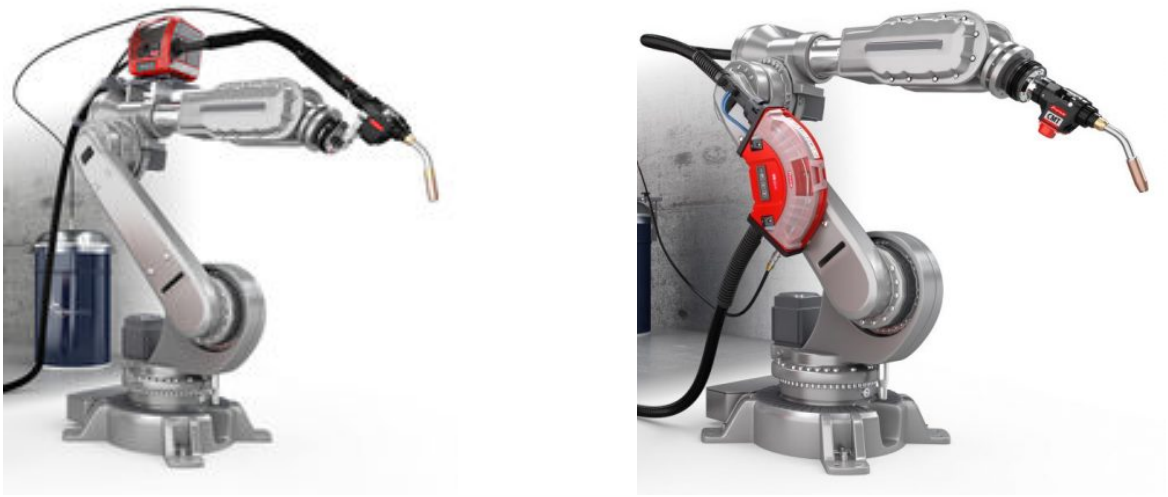
Introdução

1.1 Definição do problema e motivação

Um robô industrial é um sistema robótico usado para manufatura, sendo que sua utilização contribui para o aumento da flexibilidade de sua célula de produção, de forma semelhante às máquinas de Controle Numérico Computadorizado, do inglês *Computer Numeric Control* (CNC). Assim, caso o objeto a ser produzido seja atualizado ou mesmo trocado por um novo modelo, geralmente demanda-se uma mudança na programação e, as vezes, da ferramenta. Já as máquinas dedicadas a uma tarefa específica de fabricação são mais rígidas e difíceis de serem adaptadas a mudanças no produto. É bem verdade que máquinas dedicadas ou específicas geralmente apresentam maior volume de produção. Em células de produção robotizadas, é comum o uso de manipuladores alimentando e retirando peças de máquinas tais como CNCs, prensas, cortadoras, etc. É comum seu emprego com a manipulação de objetos em tarefas de montagem, acabamento ou retirando objetos de esteiras para embalagem (*pick and place*) ou paletização (*palletizing*) (SPONG; HUTCHINSON; VIDYASAGAR, 2005).

De acordo com a Organização Internacional de Normalização, do inglês *International Organization for Standardization* (ISO), um robô industrial é definido como "um manipulador controlado automaticamente, reprogramável, multifuncional, programável em três ou mais eixos, que pode ser fixo ou móvel para uso em aplicações de automação industrial". Existem algumas formas de classificação de robôs, segundo seu número e tipo de juntas, que podem ser rotacionais (R) ou prismáticas (P). Assim, tem-se os robôs cartesianos (que são PPP), ou os robôs cilíndricos (RPP), os esféricos (RRP) e os antropomorfos (RRR),

para a parte do posicionamento. Tais arranjos são frequentemente complementados por mais três juntas rotacionais, numa configuração de punho esférico, para fornecer orientação generalizada dentro do espaço de trabalho do robô. Outra classificação separa os robôs industriais em manipuladores de propósito geral e manipuladores de uso específico, como é mostrado na Figura 1.1. Um robô de propósito geral é basicamente um suporte de ferramentas articulado e programável, capaz de movimentar e posicionar a ferramenta que for fixada em sua extremidade (flange) com considerável precisão. Já um manipulador de uso específico, terá seu projeto otimizado para a realização de determinada tarefa, tendo uma ferramenta específica, além de cabos e mangueiras integradas a sua estrutura mecânica. Um exemplo de robô de uso específico é o de soldagem a arco (SPONG; HUTCHINSON; VIDYASAGAR, 2005).



(a) (b)

Figura 1.1 – Robô de propósito geral (a) e robô dedicado à soldagem (b).

Com relação às partes constituintes, um robô industrial é dividido em sua estrutura mecânica, na unidade controladora e no Terminal de programação, comumente chamado de *teaching pendant* (TP). Podem integrar o sistema robótico diferentes ferramentas, dispositivos auxiliares, como trilhos para movimentação do robô, e dispositivos periféricos, como mesas móveis do tipo pórtico.

Os robôs industriais tradicionais possuem a unidade controladora como uma arquitetura fechada de controle e de geração de trajetória fornecida pela fabricante. Esse tipo de solução segue o paradigma a qual deve ser priorizada a robustez e confiabilidade do funcionamento e da execução das tarefas, além de atender a certos requisitos normativos e legais que regem as indústrias. A controladora desses robôs é programada via uma lin-

guagem dedicada, normalmente derivada de uma linguagem de programação de propósito geral acrescentada das funcionalidades necessárias aos robôs, como os tipos de comandos MOVE. Logo, cada fabricante de robô possui sua própria linguagem dedicada, os robôs ABB são programados na linguagem RAPID, os robôs Kuka em KRL, os robôs Comau em PDL2, entre outros. A linguagem PDL2 é derivada da linguagem PASCAL, por exemplo.

Portanto, as controladoras fechadas buscam atender os requisitos de confiabilidade e também de simplicidade, demandando pouco conhecimento ou treinamento para operar ou programar um robô. No entanto, é perceptível que essa forma fechada dificulta a realização de novas implementações e atividades de pesquisa com os robôs industriais. Ao mesmo tempo que estabelece, em certo ponto, uma dependência com os fabricantes e/ou com as empresas parceiras quando surge a necessidade de uma mudança ou expansão da célula de produção robotizada.

Existem alguns fabricantes de robôs industriais, como a Comau, a Universal Robots, a Staubli e a Kuka, que possuem a opção de controladora aberta, isto é, controladoras que permitem a sua programação utilizando linguagens de propósito geral, como C ou C++. A vantagem de usar esse tipo de linguagem, em comparação com o uso de uma linguagem dedicada à robótica, é que as possibilidades de uso são expandidas. Por exemplo, é possível criar seu próprio gerador de trajetória, utilizar visão computacional para controlar a pose do robô, realizar um controle de força, substituir os controladores das juntas do robô, entre outras possibilidades. Isso vai mais ao encontro das novas demandas da Indústria 4.0, bem como dos pesquisadores e desenvolvedores. Por exemplo, se surge a demanda por acrescentar uma malha de controle com visão computacional, poder-se-a integrar uma ou mais câmeras que tenham um bom custo benefício, e não câmeras de um fornecedor específico, que geralmente apresentam um custo mais elevado.

É importante ter a compreensão de que uma controladora aberta geralmente não vai ao encontro dos principais anseios das indústrias que empregam robôs. Esses anseios estão mais relacionados ao cumprimento, por longos períodos, de tarefas que geralmente superam a força humana, mantendo uma precisão e repetibilidade que refletem na qualidade dos bens produzidos. Assim, a indústria demanda que a programação desse equipamento seja facilmente aprendida por seus funcionários que irão operá-lo, sendo que uma linguagem dedicada à robótica é concebida justamente com essa intenção. Esse acaba não sendo o caso do C e C++, já que, por serem de propósito geral, possuem muito mais funcionali-

dades do que apenas mover o robô por uma trajetória, ligar ou desligar uma ferramenta, etc. Além disso, como comentado anteriormente, o paradigma da controladora fechada oferece mais robustez e confiabilidade em seu funcionamento e na execução das tarefas.

A Comau desenvolveu uma controladora aberta que é uma variante de sua controladora padrão. Essa abertura é realizada mediante a adição de um pequeno hardware de comunicação e de um programa ao PLC da controladora com objetivo de implementar a comunicação com um computador industrial fornecido pela fabricante. Neste computador são executados os programas desenvolvidos pelo usuário em C ou C++ com a possibilidade de controlar a posição, velocidade ou torque das juntas do robô, ao mesmo tempo que lê os seus dados sensoriais de posição e de estimativa da velocidade. Dessa forma, a controladora aberta continua trabalhando da mesma forma que a controladora fechada, com a exceção de quando é ativada a modalidade aberta através da aplicação mencionada acima, feita na linguagem PDL2. Quando isso acontece, a aplicação do usuário no computador industrial passa a comandar o robô. Logo, a solução da Comau pode ser considerada uma solução híbrida, permitindo ao usuário alternar entre usar a linguagem robótica para controlar o robô, ou usar a linguagem C/C++ para controlar o mesmo através da sua implementação em conjunto com recursos do sistema Open.

Apesar dessa solução inovadora, ela possui certas limitações. Por exemplo, a linguagem de programação do desenvolvimento das aplicações é restrita à linguagem C ou C++, ao hardware fornecido pela fabricante (o computador industrial) tanto em relação a potência computacional quanto à arquitetura do mesmo, ficando restrito também ao sistema operacional fornecido pela fabricante.

Assim sendo, pareceu relevante a tentativa de se investigar se seria possível dar mais liberdade no desenvolvimento de aplicações para um robô industrial com controladora aberta, onde um terceiro dispositivo ficaria responsável por realizar o processamento necessário à geração de trajetória e enviar as referências de posição ao computador industrial fornecido pela fabricante, que por sua vez repassaria as referências para a controladora do robô. Assim, tal computador industrial ficaria responsável por manter uma comunicação mais segura e controlada, recebendo as referências do terceiro dispositivo computacional e repassando para a controladora do robô.

A proposta deste Trabalho de Conclusão de Curso é desenvolver um arranjo cliente servidor para o computador industrial que possibilitará o desenvolvimento de aplicações

em qualquer linguagem de programação, qualquer arquitetura e qualquer sistema operacional para um terceiro dispositivo que seja capaz de se conectar com o mesmo por uma Interface de Programação de Aplicativo, do inglês *Application Programming Interface* (API) desenvolvida usando o protocolo Protocolo de Controle de Transmissão / Protocolo de Internet, do inglês *Transmission Control Protocol / Internet Protocol* (TCP/IP), sendo o arranjo cliente servidor responsável apenas por ser uma ponte de comunicação entre a controladora do robô e o programa rodando no terceiro dispositivo. Assim, dando mais liberdade aos desenvolvedores de aplicações para a Indústria 4.0 e para os pesquisadores universitários, atendendo inclusive a demanda do Laboratório de Robótica do CEFET-MG, o qual possui uma controladora aberta desse tipo, a C5G Open.

Partiu-se da hipótese de que seria possível desenvolver um programa para o computador industrial, que se comunicasse com a controladora do robô por meio do protocolo fornecido pela fabricante ao mesmo tempo, porém em taxas diferentes, e que se comunicasse com um programa em um terceiro dispositivo via API desenvolvida usando protocolo TCP/IP, sendo este terceiro dispositivo rodando um sistema operacional diferente, com uma arquitetura diferente e tal programa em uma linguagem de programação diferente da exigida pela solução da fabricante. Assim, para viabilizar o teste dessa hipótese, realizou-se uma pesquisa de finalidade aplicada, objetivo exploratório, sob o método hipotético-dedutivo, com abordagem quantitativa, realizada com procedimentos experimentais.

1.2 Objetivos

O objetivo geral do trabalho é desenvolver um programa com arranjo do tipo cliente servidor que estenda a liberdade da solução de controladora aberta que a fabricante proporciona aos programadores de robôs industriais.

1.2.1 Objetivos específicos

Para atingir o objetivo geral, é necessário desenvolver:

- Uma API que usa o protocolo de comunicação TCP/IP para enviar e receber variáveis referentes aos ângulos de juntas do robô.
- Uma função que salve na memória os ângulos de referência das juntas do robô oriundos do programa no terceiro dispositivo, e que envie os dados sensoriais armazenados

na memória para tal programa.

- Uma função que use a solução da fabricante para ler dados sensoriais, salvando esses dados na memória, e enviar referências das juntas do robô armazenadas na memória.

- E resolver o problema gerado pelo assincronismo das comunicações entre arranjo controladora e arranjo com o programa no terceiro dispositivo.

1.3 Organização do texto

O Capítulo 2 apresenta a revisão teórica do tema e assuntos relacionados ou tratados.

O Capítulo 3 apresenta o desenvolvimento dos programas do arranjo cliente servidor, incluindo um exemplo de programa cliente para realizar os testes e coletar os resultados experimentais.

Já o Capítulo 4 apresenta os resultados experimentais e discute as principais consequências deles.

Ao final deste trabalho, são apresentadas no Capítulo 5 as conclusões relacionadas ao objetivo de estender a liberdade que a solução da fabricante proporciona aos programadores de robôs industriais.

Capítulo 2

Fundamentação

2.1 Controladora Robótica Aberta

Algumas tentativas feitas pela academia foram relatadas na literatura no sentido de abrir a estrutura de controle de alguns robôs industriais (ROBERTI et al., 2010). Recentemente, alguns fabricantes de robôs perceberam essa demanda por parte das instituições de ensino e pesquisa e, de forma semelhante ao que vem acontecendo com os programas de código aberto do Sistema Operacional Robótico, do inglês *Robot Operation System* (ROS) passaram a oferecer a opção de abertura do sistema de controle de robôs (KUBUS D. NILSSON; JOHANSSON, 2010), isto é, uma controladora aberta.

Uma opção para atender essa demanda foi desenvolvida pela fabricante de robôs industriais *Comau*, adicionando uma arquitetura complementar à controladora de seus robôs, a *Comau C5G*. Essa extensão, denominada *Comau C5G Open Controller*, é feita através da conexão da controladora a um computador industrial externo, denominado Computador Pessoal com Linux do Open, do inglês *Open Linux Personal Computer* (Open LPC), por intermédio de uma conexão Ethernet PowerLink (EPL), como é mostrado na Figura 2.1 (ANTONELLI et al., 2010).

A Figura 2.2 permite a visualização da solução fornecida pela fabricante, sendo que, a conexão *Ethernet* TCP/IP precisa ser intermediada por um modem, roteador ou *switch* não fornecido nessa extensão.

Assim, é possibilitado à controladora do robô industrial enviar dados sensoriais e receber referências de um programa a ser programado pelo usuário, a taxas de até 1 pacote a cada 0,4 ms (1 pacote contém todas as leituras de sensores do robô). Isso possibilita a

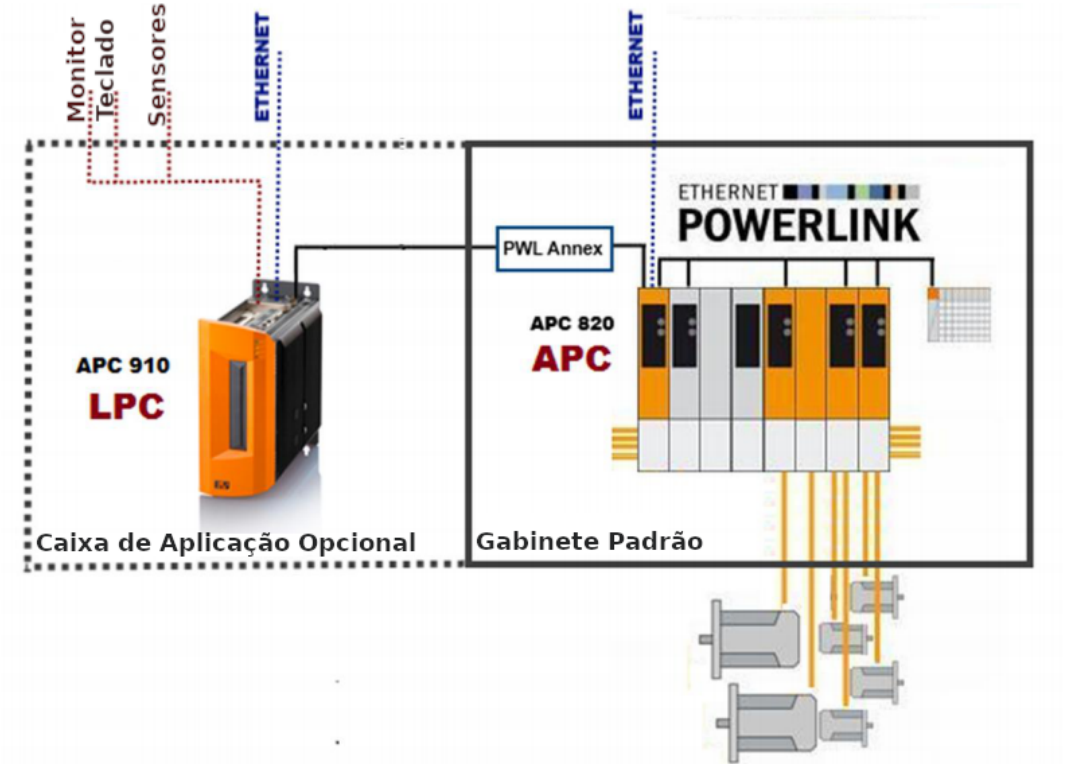


Figura 2.1 – Visão geral sobre o hardware do sistema C5G-Open.
Adaptado de (FERRARA, 2013)

criação de programas que implementem malhas de controle adicionais como: controle de força, visão computacional, entre outros. (FERRARA, 2013)

Entretanto, existem algumas limitações, por exemplo: o programa *User Application* (na Figura 2.2), que implementa essa malha de controle adicional, precisa ser compilado e executado nesse computador industrial externo, rodando na distribuição GNU *Linux Mint 17* de arquitetura x86, conectado à controladora por um cabo de rede especial chamado EPL (visto na Figura 2.2). Esse programa precisa ser feito em linguagem C/C++ e utilizar a biblioteca proprietária de código fechado desenvolvida pela *Comau* chamada Biblioteca Robótica Aberta Realística Melhorada, do inglês *enhanced Open realistic Robot Library* (eORL), para controlar a comunicação.

Além disso, uma utilização inadequada por um usuário que deixe a conexão com o Open LPC habilitada do lado da controladora e que não tenha a adequada resposta do lado do Open LPC, por exemplo, poderá produzir um erro de comunicação que acabará sendo armazenado em seus registros e impedirá a liberação dos *drivers* da controladora pelo seu módulo de gerenciamento de segurança, chamado C5G-SDM (Módulo de Distribuição e Segurança, em português) (COMAU_ROBOTICS, 2010). Vale registrar que tal evento aconteceu em uma oportunidade com a controladora C5G do Laboratório de Robótica do

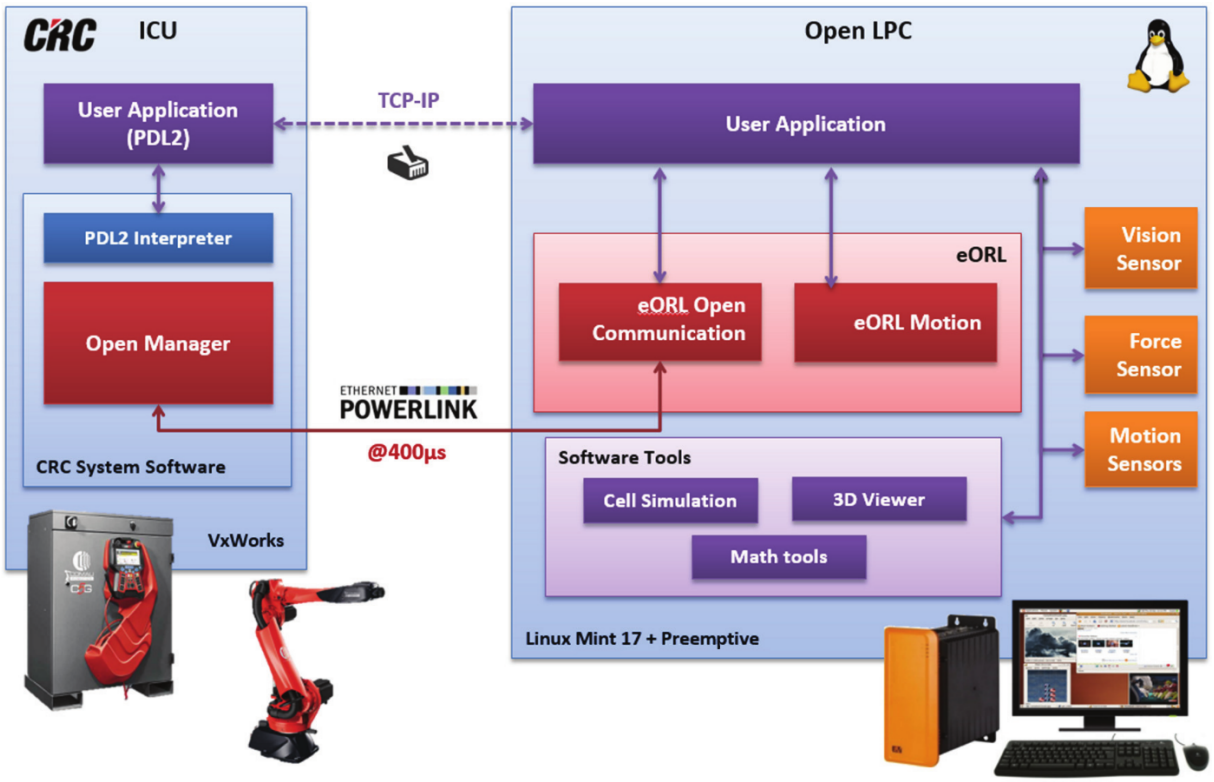


Figura 2.2 – Visão geral do programa do sistema C5G-Open.
Fonte: (COMAU_ROBOTICS, 2019)

CEFET-MG.

Por fim, nem todos os programas desejados/necessários podem ser instalados no Open LPC, ou a versão necessária/desejada da aplicação, devido ao fato que os drivers para o hardware de comunicação EPL presente no Open LPC precisam de uma versão mais antiga da distribuição Ubuntu (ou derivados dela, como Linux Mint) para funcionar. Logo, o desenvolvimento de aplicações fica limitado. Um exemplo de limitação de instalação é o ROS (BISSON, 2014). Atualmente é executado no Open LPC a versão 17 da distribuição *Linux Mint*, que é baseado na versão Suporte de longa duração, do inglês *Long Term Support* (LTS) de 2014 da distribuição Ubuntu. Em consequência, todos os programas distribuídos e versões de bibliotecas do sistema operacional GNU são versões ultrapassadas e, geralmente, não oferece suporte à instalação de novos programas ou de novas versões de bibliotecas, como é o caso do ROS2.

2.1.1 Abertura da malha de controle

Para utilizar o sistema *Open* é necessária a instalação da biblioteca eORL no Open LPC na mesma versão da eORL da controladora (COMAU_ROBOTICS, 2019). Além disso, é

necessário habilitar o modo Open nos eixos desejados através da controladora. Para isso, é usado o programa *Crcopen* disponível na controladora, conforme indicado na Figura 2.3.

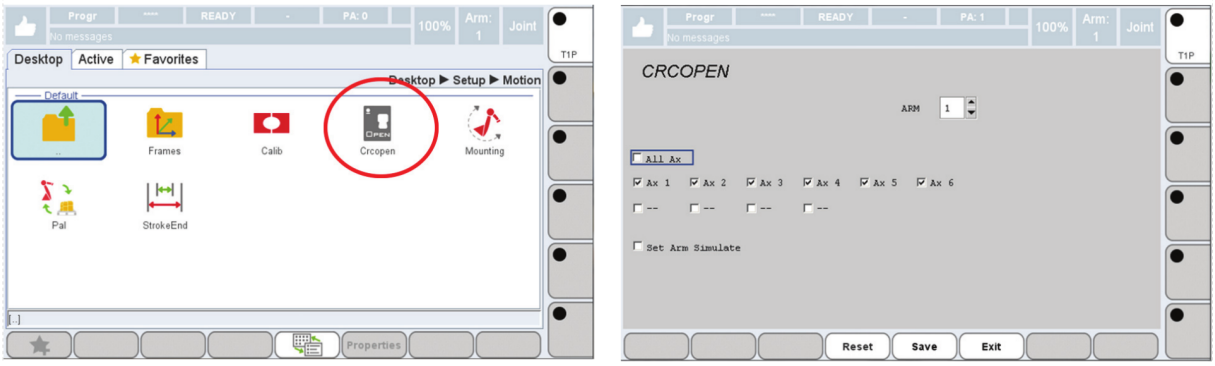


Figura 2.3 – Procedimento para habilitar o modo Open dos eixos do robô (COMAU_ROBOTICS, 2019).

Depois de selecionar quais eixos serão habilitados no sistema Open, é necessário reiniciar a controladora. Depois de reiniciada, cada eixo fica por padrão na modalidade "CRCOPEN_LISTEN", que significa que o Open LPC está apto somente para ler os valores dos sensores dos eixos habilitados. Além disso, a controladora só permitirá ligar os motores do robô se o Open LPC estiver ligado, conectado à controladora, executando algum programa que usa a biblioteca eORL e se esse programa já tiver iniciado a comunicação com a controladora. Na Figura 2.4 é possível ver o sistema de controle implementado pela fabricante na controladora *C5G Open* e quais dados podem ser enviados para o Open LPC.

Existem outras modalidades que podem ser alteradas durante a execução do código em PDL2. Na versão de programa instalada atualmente na controladora estão disponíveis as modalidades de controle de posição:

- CRCOPEN_POS_ABSOLUTE
- CRCOPEN_POS_RELATIVE
- CRCOPEN_POS_ADDITIVE

A diferença entre elas pode ser lida no manual (COMAU_ROBOTICS, 2019). A modalidade utilizada no escopo deste trabalho é a CRCOPEN_POS_ABSOLUTE, que significa que o Open LPC tem a capacidade de enviar as referências diretamente para o controlador de posição. Na Figura 2.5 é possível ver o sistema de controle interno da controladora e quais dados são enviados para o Open LPC. Além disso, é possível ver

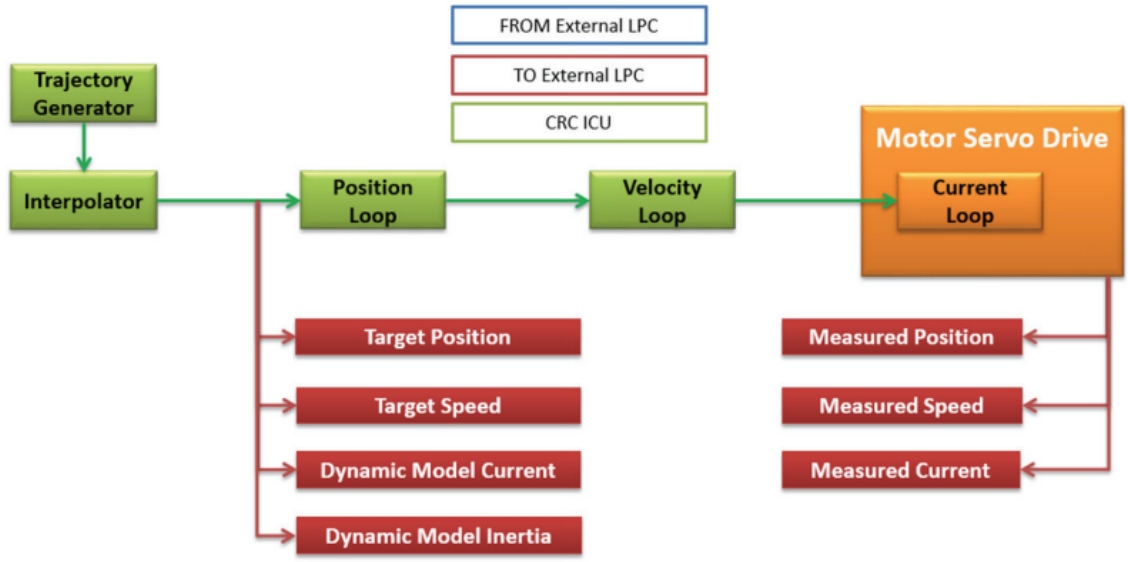


Figura 2.4 – Sistema de controle em modo CRCOPEN_LISTEN (COMAU_ROBOTICS, 2019).

também quais dados a controladora recebe do Open LPC, ou seja, a posição angular de referência de cada junta.

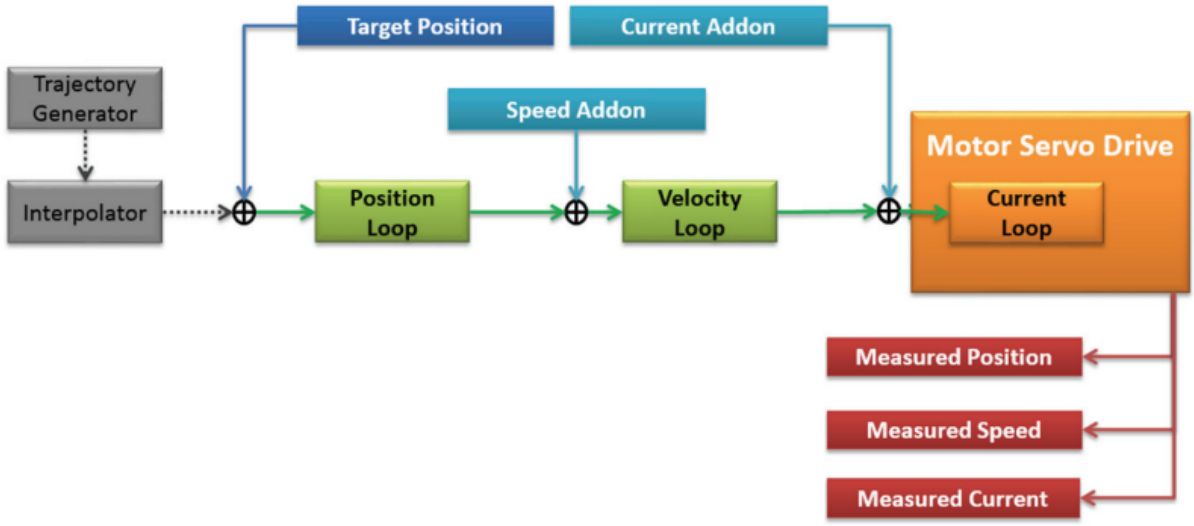


Figura 2.5 – Sistema de controle em modo CRCOPEN_POS_ABSOLUTE (COMAU_ROBOTICS, 2019).

Para trocar a modalidade dos eixos é necessário executar uma função durante a execução de um código em PDL2 na controladora. Portanto, é possível executar movimentações com códigos PDL2, e depois trocar a modalidade dos eixos do robô para o Open LPC assumir o controle da movimentação do robô. Depois que o programa no Open LPC executa a tarefa, ele pode enviar um comando para a controladora avisando. Assim ela

voltará a executar o restante do código PDL2 de antes. Ou seja, é possível alternar entre movimentação pelo código PDL2 na controladora e movimentação pelo código C/C++ no Open LPC quantas vezes forem necessárias. A Figura 2.6 ilustra a alternância de modalidades durante o cumprimento de uma tarefa.

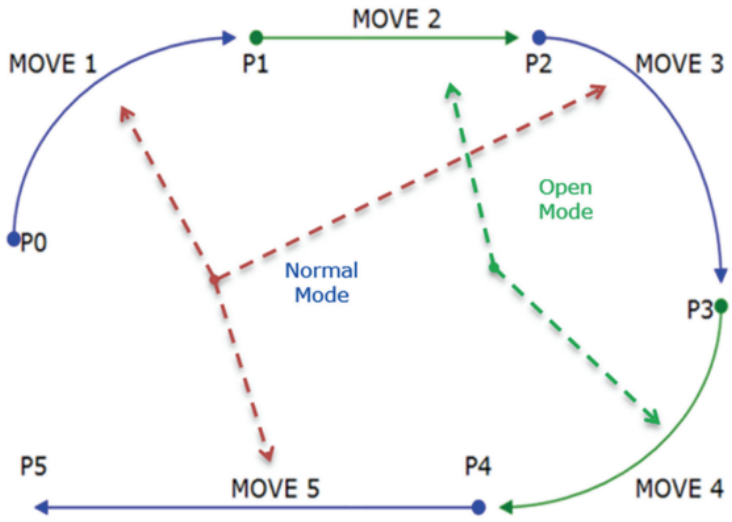


Figura 2.6 – Exemplo de cumprimento de uma tarefa com alternância de modalidades (COMAU_ROBOTICS, 2019).

A função em código PDL2 que altera a modalidade dos eixos é a *ru_crcopen_set_modality()*. A partir do momento que a modalidade é alterada, o programa no Open LPC já está apto a usar as funções da eORL para enviar as referências do controlador de posição. No caso, a função da eORL em C que envia as referências em graus para a controladora é a *ORLOPEN_set_absolute_pos_target_degree()*, e a função que realiza a leitura dos sensores de posição é a *ORLOPEN_get_pos_measured()*.

2.2 Robô Virtual no *Open LPC*

Somente essas funções não bastam para controlar a movimentação do robô, pois a controladora não aceita referências de posições descontínuas ou que gerem uma resposta brusca nos atuadores, causando um aviso de alerta e desligando o sistema Open.

Para enviar as referências de posição para a controladora é preciso gerar uma trajetória suave e enviar a cada instante uma posição diferente da mesma. O usuário pode optar por programar um gerador de trajetória se ele quiser. No entanto, a fabricante implementou o gerador de trajetória da controladora dentro da eORL, disponibilizando assim para

os programadores utilizarem no desenvolvimento dos programas no Open LPC. Dessa forma, acaba se evitando que cada usuário tenha necessariamente que programar o seu gerador de trajetória, ao mesmo tempo que possibilita que o usuário consiga repetir a mesma movimentação da controladora no sistema Open.

Para utilizar a implementação do gerador de trajetória é necessário iniciar um robô virtual através da eORL no Open LPC, o que é feito automaticamente durante o início da conexão com a controladora com a função *ORLOPEN_initialize_controller()*. Depois é preciso fazer o robô virtual ficar na mesma configuração (ângulos) de juntas que o robô real com a função *ORLOPEN_sync_position()*, estabelecer qual é a nova posição que o robô deve ir, bem como os parâmetros de movimentação, através da função *ORL_set_move_parameters()*. E, então, quando executada a função *ORL_get_next_interpolation_step()*, será obtida a posição de referência do robô real que será enviada para a controladora, que é a posição atual do robô virtual. Desta maneira o robô real irá "seguir"o robô virtual.

Como o gerador de trajetória implementado na eORL é o mesmo implementado na controladora, de acordo com a fabricante, a movimentação do robô virtual no Open LPC será a mesma que a realizada por um comando equivalente em PDL2 na controladora. E como o robô real "segue"o robô virtual, é possível afirmar que a movimentação do robô real é praticamente a mesma que se o comando de movimentação tivesse sido realizado direto na controladora.

2.2.1 Modo de simulação da *eORL*

Existe uma outra maneira de se inicializar o robô virtual sem a conexão com a controladora, e é também através da função *ORL_initialize_controller()*. Para isso, é necessário primeiramente ter-se armazenado o arquivo de configuração do robô no Open LPC. Assim a eORL terá condições de realizar sua simulação do robô real sem a comunicação com a controladora.

É necessário definir uma taxa de amostragem para a simulação por meio da função *ORL_set_interpolation_time()*. É necessário definir também a posição atual do robô simulado, que no caso será a posição inicial, através da função *ORL_set_position()*. Agora é possível enviar comandos de movimentação para o robô virtual da mesma forma que anteriormente, ou seja, usando a função *ORL_set_move_parameters()*.

Cada vez que a função *ORL_get_next_interpolation_step()* é chamada, ela retorna a posição virtual partindo do pressuposto que o tempo passado desde a última chamada foi o tempo definido na função *ORL_set_interpolation_time()*. Portanto, caso for desejada uma simulação de tempo real, é necessário usar o relógio do sistema para garantir que o código seja executado na mesma taxa de amostragem definida previamente.

2.3 Modo de Simulação da Controladora

A fabricante da controladora desenvolveu um modo de simulação para que se possa realizar testes com segurança das tarefas a serem preparadas para o sistema Open. Este modo de simulação envia um comando para os controladores dos motores não atuarem nos mesmos e não lerem os sensores. Ao invés disso, ele irá usar a resposta esperada dos motores para estimar os valores dos respectivos sensores. Desta forma, o robô permanece seguro sem se mover até que seja reiniciado o robô sem o modo de simulação.

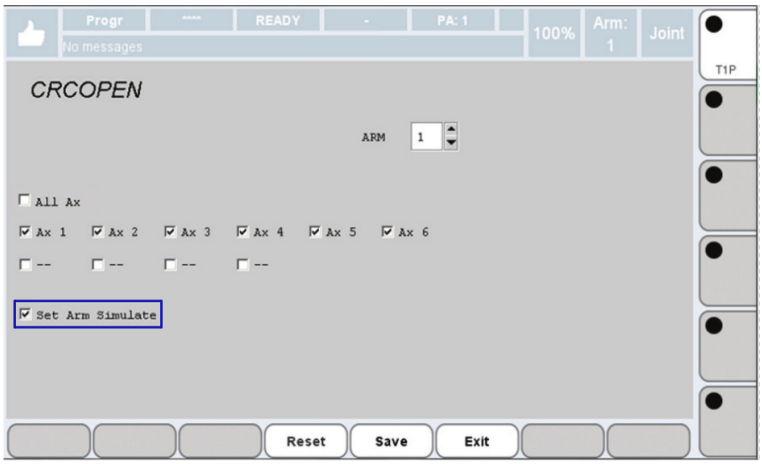


Figura 2.7 – Ativando o modo de simulação. Adaptado de (COMAU_ROBOTICS, 2019).

Depois que é ativado o modo de simulação e a controladora é reiniciada, é possível operá-la normalmente. É possível executar os programas em PDL2 presentes na controladora em modo de simulação, é possível usar o Open LPC para controlar a posição do robô pelo sistema Open. Todas as outras modalidades Open também estarão disponíveis no modo de simulação. A única diferença é que os freios dos motores do robô real permanecem sempre ativados e os controladores dos motores não mandam mais corrente elétrica para os motores. Ao invés isso, eles passam a usar seu poder computacional para realizar a simulação do robô.

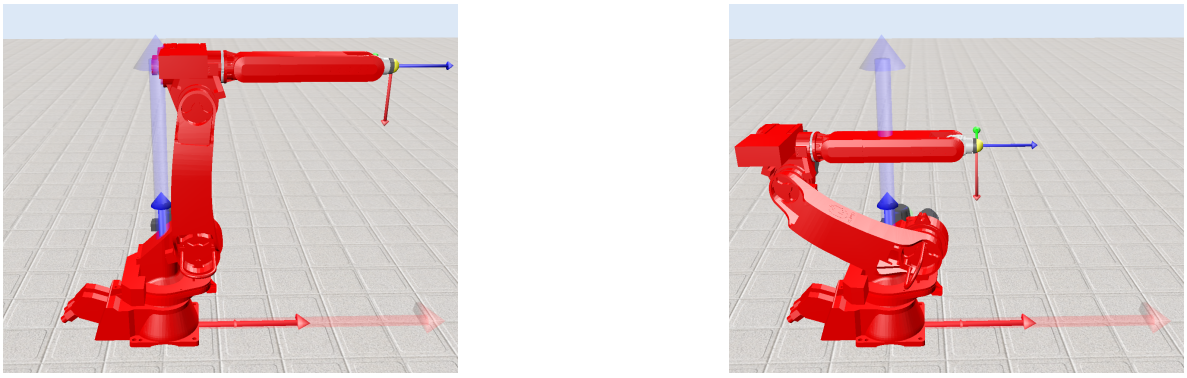
De acordo com a fabricante (COMAU_ROBOTICS, 2019), é possível acompanhar a posição do robô (real ou simulado) através:

- da tela do TP;
- do programa WinCRC;
- do programa Visual3D;
- do programa desenvolvido pelo usuário no Open LPC.

O programa *Visual3D* tem um diferencial em relação ao restante da lista que é oferecer uma visualização 3D da posição do robô em tempo real, ao contrário das outras alternativas que oferecem uma visualização numérica do valor angular de cada junta no formato de tabela, ou da posição cartesiana do efetuador do robô. Este programa pode ser executado em um computador qualquer que tenha capacidade de processamento para tal e se comunique diretamente com a controladora através da rede *Ethernet* padrão a qual a controladora esteja conectada (COMAU_ROBOTICS, 2019).

Este programa possui armazenado na memória vários modelos 3D de robôs da fabricante. Então, quando ele se comunica com a controladora pela primeira vez, ele recebe a informação de qual robô está conectado à controladora e, então, inicia a renderização 3D do modelo desse robô. Após esse processo, o programa passa a receber da controladora continuamente os valores angulares de cada junta do robô, seja ele real ou simulado. Ele passa a usar os valores que recebe da controladora para atualizar, em tempo real, a renderização 3D do robô na tela do computador, de forma a sincronizar os ângulos de cada junta. Na Figura 2.8 são mostradas capturas de tela do programa *Visual3D* exibindo o robô renderizado em dois momentos distintos.

O modo de simulação é de grande auxílio nos testes de programas desenvolvidos para o sistema Open, pois em modo de simulação é possível identificar, com segurança, se o teste simulado está ocorrendo conforme esperado. Já se esses primeiros testes forem realizados diretamente com robô real, haverá um risco de que possam ocorrer acidentes. E o programa *Visual3D* é de grande auxilio para visualizar se a simulação está ocorrendo conforme esperado pois além de oferecer uma visualização 3D do robô, é possível adicionar obstáculos no programa e verificar se ocorre colisão com esses obstáculos, por exemplo.



(a) (b)

Figura 2.8 – Modelo 3D do robô no programa Visual3D em duas configurações distintas de juntas.

2.4 Ponteiro Inteligente

Um ponteiro inteligente é um tipo de dado implementado na linguagem de programação *C++* a partir da revisão *C++11*. Esse tipo de dado fornece recursos adicionais aos ponteiros, como o gerenciamento automático de memória. No caso, eles controlam automaticamente a memória para a qual apontam e deletam automaticamente o objeto apontado após a utilização, por exemplo, porque o proprietário é uma variável local, e a execução sai do escopo da variável (CPPREFERENCE, 2015).

O Ponteiro Inteligente utilizando no desenvolvimento deste trabalho é o shared_ptr. Ele é um ponteiro compartilhado e tem a característica de nenhum shared_ptr específico possuir algum objeto. Todos os shared_ptr que apontam para o mesmo objeto colaboram para garantir sua existência e quando todos deixam de apontar para tal objeto ele é destruído automaticamente (MEYERS, 2015).

Capítulo 3

Desenvolvimento

3.1 O OpenServer

Visando oferecer mais liberdade e flexibilidade para o usuário, no que diz respeito à programação da malha de controle adicional e ao dispositivo que irá executá-la, foi desenvolvido um programa chamado OpenServer, isto é, o servidor do sistema Open. Este, tem também o objetivo de trazer mais segurança para a controladora do robô do laboratório do CEFET-MG em Divinópolis durante a execução de malhas de controle experimentais.

O OpenServer é executado no Open LPC e utiliza a biblioteca eORL para intermediar a comunicação entre a controladora do robô e o LPC, ou seja, utiliza a estrutura do sistema Open fornecida pela fabricante. Ele oferece uma API baseada em rede TCP/IP para se comunicar com um programa cliente em um terceiro dispositivo. Este programa cliente tem o papel de enviar as referências das juntas do robô para o OpenSever, que por sua vez tem que repassar estas referências para a controladora do robô. Ao mesmo tempo, o OpenSever tem o papel de receber da controladora os dados sensoriais do robô (posição angular das juntas, por exemplo) e repassar para o programa cliente. Como a comunicação entre controladora e o Open LPC pode ocorrer a uma taxa diferente da comunicação do Open LPC com o terceiro dispositivo, o OpenServer tem que conseguir realizar a comunicação de forma assíncrona.

A API foi baseada em TCP/IP porque além de ser um protocolo que garante a integridade e a ordem dos pacotes, é um protocolo já utilizado em projetos anteriores pelo discente. Logo, se torna mais rápido e fácil o desenvolvimento da parte servidor do pro-

grama.

O objetivo é possibilitar que o programa cliente possa ser escrito em qualquer linguagem de programação que suporte o protocolo TCP/IP, bem como que o programa possa rodar em qualquer sistema operacional que tenha suporte ao mesmo. Também é objetivo que o dispositivo no qual o programa cliente esteja rodando possa ser de qualquer arquitetura de processador, desta forma, permitindo que possam ser usados até dispositivos móveis como *smartphones* que utilizem a API do OpenServer.

Acredita-se que essa abordagem possa reduzir riscos ao equipamento por erro ou imperícia ao ser programada uma malha de controle usando recursos do sistema *Open* fornecidos pela fabricante. Isso porque o OpenServer pôde ser programado para manter uma comunicação segura com a controladora, mesmo que a conexão com o programa cliente seja perdida, quer seja ocasionada por um fechamento inesperado do programa cliente ou por um travamento. Já se este programa do usuário do sistema Open estivesse sendo executado no diretamente no LPC, a comunicação que ele mantinha com a controladora seria encerrada bruscamente. Além disso, o OpenServer é capaz de oferecer uma alternativa de API mais simples de ser aprendida e utilizada.

A estrutura padrão do sistema fornecida pela empresa fabricante é detalhada na Figura 3.1, onde: (1) é a conexão elétrica entre a controladora e o manipulador robótico; (2) representa a conexão *PowerLink* entre o Open LPC e a controladora; (3) é a conexão entre o roteador e a controladora; e (4) é a conexão entre o roteador e o Open LPC .

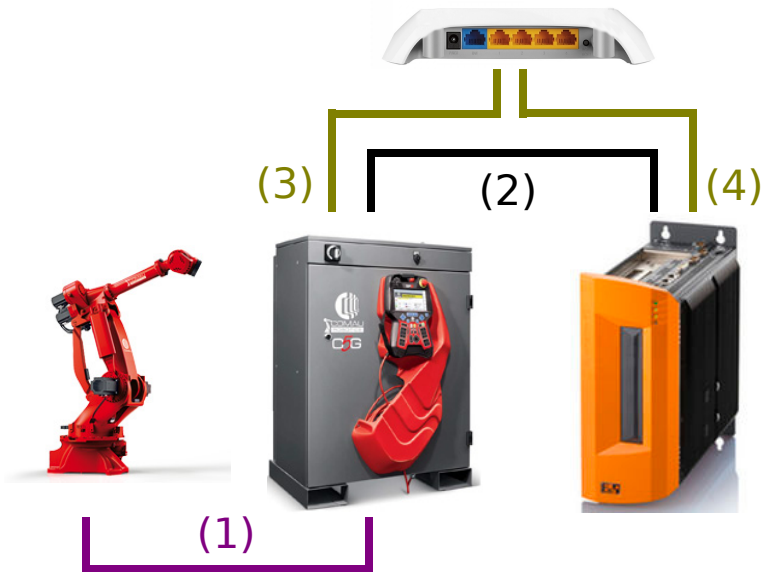


Figura 3.1 – Estrutura padrão do sistema C5G Open.

Como dito anteriormente, o OpenServer foi desenvolvido para utilizar-se a estrutura padrão do sistema fornecida pela empresa fabricante. Logo, foi programado dentro das limitações da plataforma, ou seja, na linguagem C++ usando a biblioteca eORL da fabricante, sendo usada a conexão EPL do Open LPC para fazer a comunicação com a controladora do robô e sendo compilado para o sistema operacional GNU com Kernel Linux de arquitetura x86. Além disso, ele usa a rede *Ethernet* convencional do Open LPC e bibliotecas padrão em C++ disponíveis para o sistema operacional GNU para realizar a comunicação via protocolo TCP/IP. A estrutura desenvolvida para execução do OpenServer passa então a ser conforme é mostrado na Figura 3.2, onde: (5) representa a conexão entre o roteador e o dispositivo externo, que pode ser feita através de cabeamento ou conexão sem fio; e (6) é a conexão entre roteador e rede institucional.

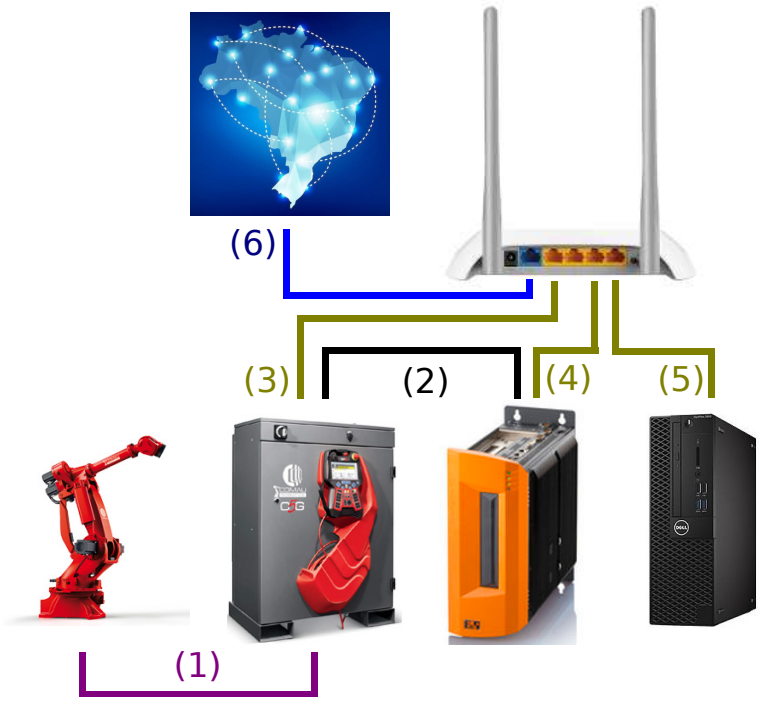


Figura 3.2 – Estrutura desenvolvida para execução do OpenServer.

A rede institucional do CEFET-MG oferece acesso à internet. No entanto, as portas do roteador são bloqueadas para acesso externo. Dessa forma, é criada uma rede interna segura onde os programas se comunicarão. Por esta razão, não foram implementadas soluções de criptografia de dados, bem como de autenticação. Isso ocorreu também pelo motivo de ser um ambiente acadêmico, controlado e sem trafego de dados sensíveis. A estrutura real do sistema pode ser visto na Figura 3.3. Estão disponíveis também dois computadores conectados à rede interna, que podem ser vistos na Figura 3.4, para a

execução de programas clientes.

Figura 3.3 – Estrutura do laboratório: robô, controladora e Open LPC.

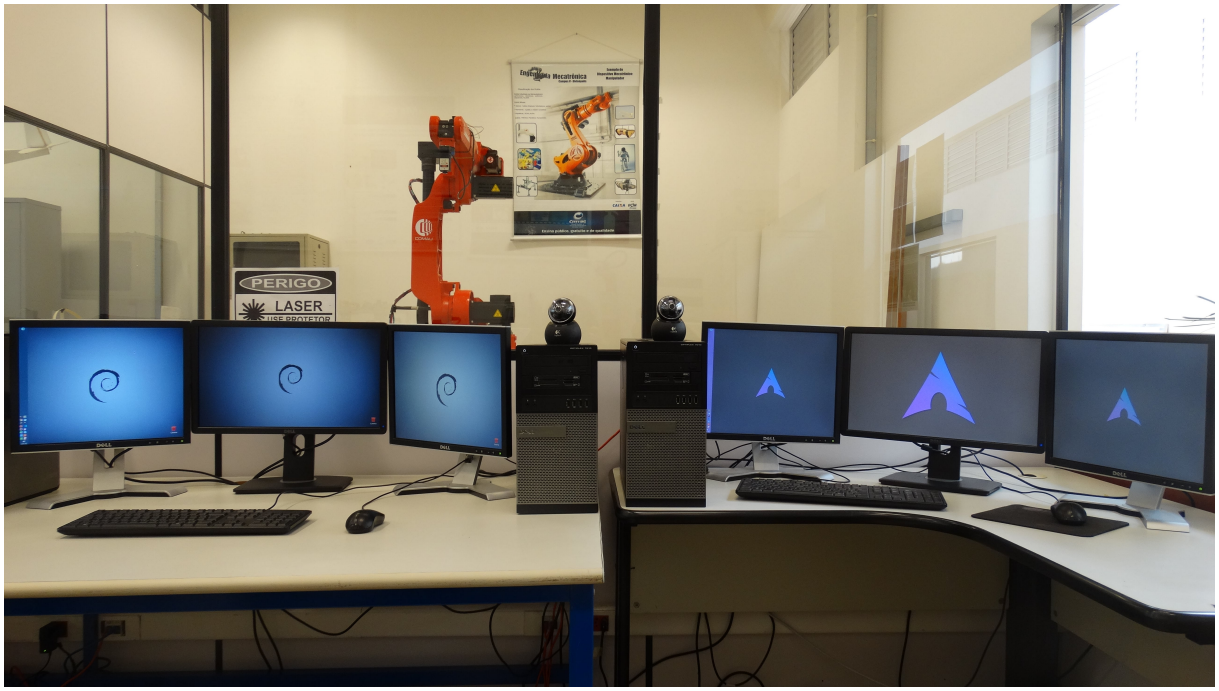


Figura 3.4 – Estrutura do laboratório: computadores onde serão executados os programas clientes.

O desenvolvimento desta abordagem torna mais segura a utilização do sistema Open em um ambiente de ensino e pesquisa, pois o único programa que precisará ser executado no Open LPC passa a ser o *OpenSever* em uma versão estável. Já no dispositivo externo

serão executados os programas experimentais ou em desenvolvimento, os quais podem travar durante a execução, por exemplo, entre outros acontecimentos que possam gerar uma falha de comunicação. No caso em que algo assim vier a acontecer, o *OpenSever* continuará a ser executado no Open LPC e manter-se-a comunicando com a controladora do robô, enviando a última referência armazenada na memória, até que o programa cliente se reconecte ou o OpenServer receba o comando de desligar a comunicação. Dessa forma, o risco de se interromper bruscamente a comunicação com a controladora é evitado.

Além disso, o *OpenSever* expõe apenas um subconjunto das funcionalidades e configurações da eORL, simplificando a utilização e reduzindo a quantidade de erros possíveis.

3.1.1 Funcionamento

Foi implementado um modo *DEBUG*, a ser ativado antes do processo de compilação, que imprime no terminal qual função foi executada, o comando vindo do programa cliente e a resposta enviada, conforme pode ser visto nas Figuras 3.5, 3.6 e 3.7. Quando o modo *DEBUG* não está ativado, apenas a biblioteca eORL imprime textos no terminal, por exemplo, a tela de boas vindas, mensagens de erro, entre outros. Caso seja desejado, também é possível desativar as mensagens emitidas pela eORL desativando a "verbosidade".

Quando o programa é executado, ele tenta se conectar com a controladora usando a biblioteca eORL através da conexão EPL. Quando a conexão com a controladora através da eORL é bem sucedida, a mesma realiza o download do arquivo de configurações do robô na controladora e passa a simular um robô virtual no Open LPC, enviado como referência a posição do robô virtual para a controladora do robô real realizar o seguimento de referência, desta forma sincronizando ambos. Quando o robô virtual simulado se movimenta, a controladora do robô real copia esse movimento. Essa simulação e comunicação são realizadas pela *eORL Motion* e pela *eORL Comunication*, trabalhando em conjunto. Depois que esse processo acaba, o programa entra em modo *Listen*, ou seja, fica esperando um cliente se conectar ao *soket* através da rede TCP/IP via *Ethernet* convencional, conforme pode ser visto na Figura 3.5.

O funcionamento do programa pode ser alterado antes do processo de compilação, apenas alterando uma flag, onde ao invés de sincronizar a movimentação do robô virtual e real, o OpenServer passa a enviar direto para o controlador de posição da controladora

as referências recebidas do programa cliente. Assim, o programa cliente será o responsável por gerar a trajetória. Usar a eORL para gerar a trajetória localmente no OpenServer é a opção mais segura, pois caso contrário, se a comunicação com entre o OpenServer e o programa cliente for interrompida com o robô em movimento, isso geraria uma parada brusca no robô e erro na controladora devido à referência de posição descontínua. A opção de enviar as referências de posição diretamente da controladora é interessante para desenvolver geradores de trajetória usando o OpenServer e sua API para facilitar o desenvolvimento. Para garantir que não haverá interrupção/atraso na comunicação, o gerador de trajetória desenvolvido deverá ser executado no Open LPC juntamente com o OpenServer, com ambos comunicando pela rede interna *localhost*.

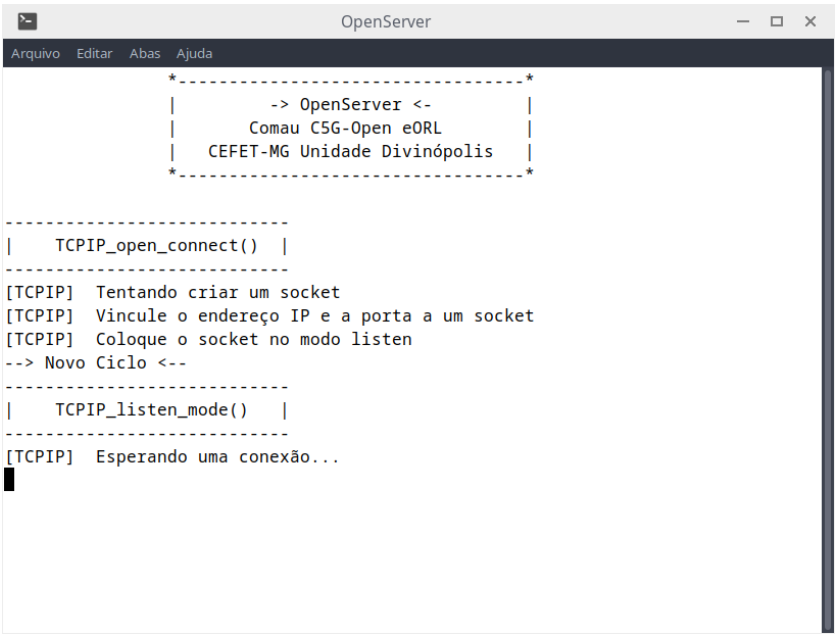


Figura 3.5 – OpenServer esperando a conexão de um programa cliente.

Quando um cliente se conecta ao servidor é realizada uma troca de pacotes para verificar a integridade da conexão. Caso essa verificação seja bem sucedida o servidor fica esperando o recebimento de instruções do cliente, conforme pode ser visto na Figura 3.6. O servidor só atua mediante os comandos do cliente.

O primeiro caractere recebido pelo servidor informa qual procedimento ele deve seguir. No caso, a instrução 'j' está configurada para logo após virem os valores de referência das juntas do robô (truncado em 5 casas decimais, mas, sem uso da vírgula para representar as casas decimais) concatenados na forma de *strings*, como pode ser visto na Figura 3.7. Ao receber essa instrução existem duas ações possíveis a serem realizadas, escolhida na etapa

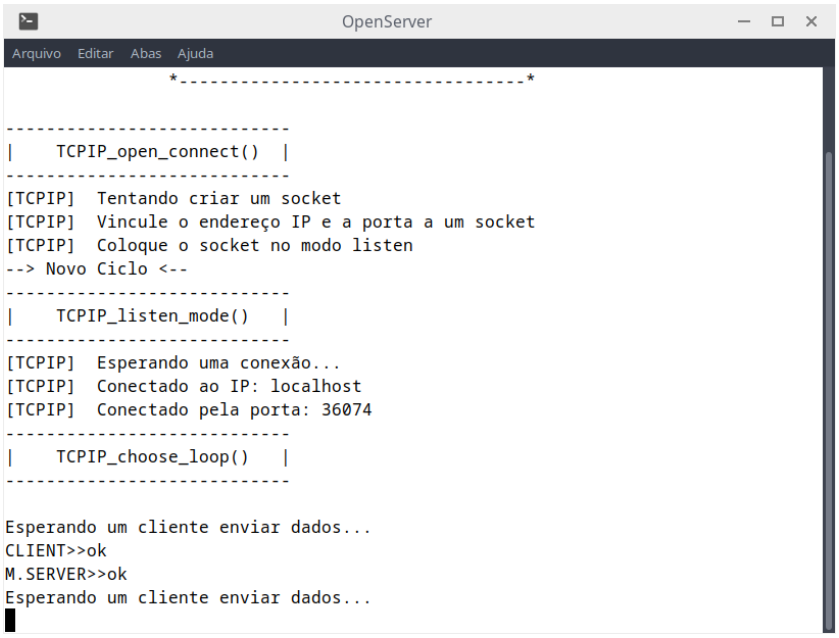


Figura 3.6 – OpenServer esperando a instrução do programa cliente conectado.

de pré-compilação: ou imediatamente o OpenServer envia o comando de movimentação para a *eORL* gerar a tragetória, ou usa a *eORL* para enviar diretamente as referências para a controladora.

Em seguida, também na forma concatenada sem uso de virgula, o servidor envia a resposta ao programa cliente com a leitura mais recente dos sensores das juntas do robô armazenada na memória do servidor, as quais foram recebidas da controladora.

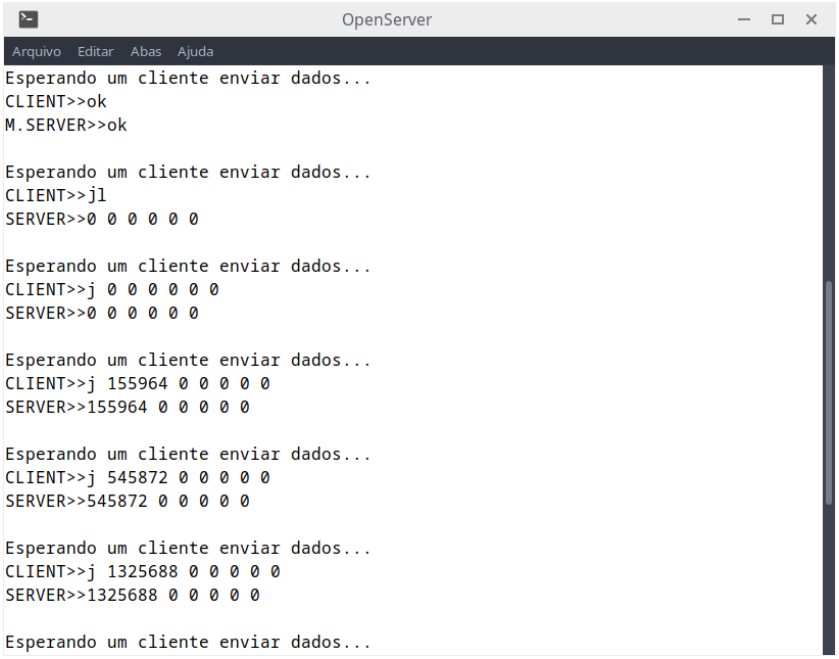


Figura 3.7 – OpenServer executando a instrução do programa cliente e respondendo com a leitura dos sensores.

Esta forma de transmitir os dados (*strings* concatenadas) foi desenvolvida com a intenção de ser uma construção simples de ser implementada em outras linguagens de programação.

Para ilustrar o funcionamento simultâneo do OpenServer, do programa cliente e da controladora, bem como a comunicação entre eles, foi desenvolvido um fluxograma contendo o funcionamento desta estrutura. O mesmo pode ser visualizado na Figura 3.8, onde as setas na cor preta mostram o fluxo de funções do programa; as setas na cor ciano indicam a comunicação *Ethernet* entre o dispositivo do cliente e o Open LPC; e as setas na cor magenta indicam a comunicação do OpenServer com a controladora do robô.

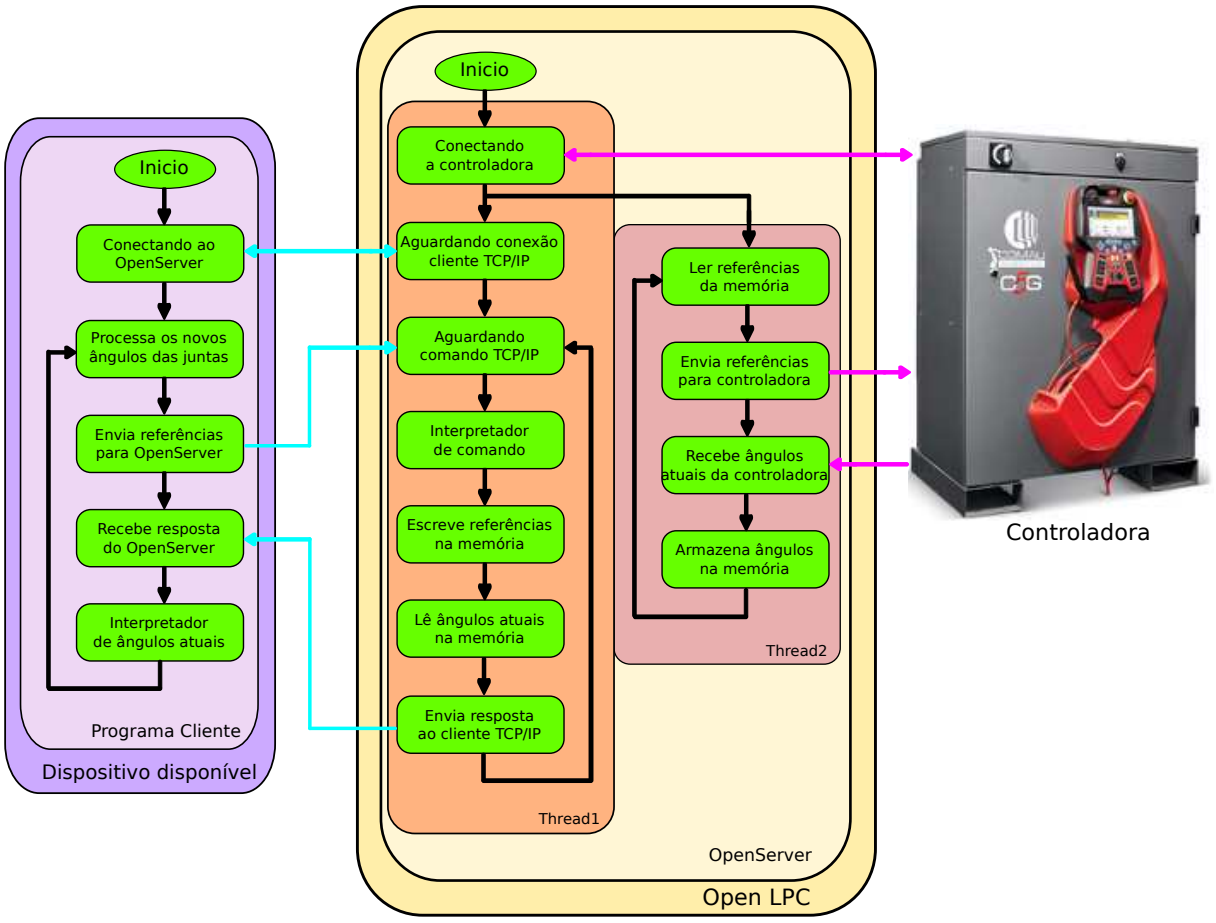


Figura 3.8 – Fluxograma do funcionamento e comunicação do OpenServer, programa cliente e controladora.

Para controlar o robô pelo OpenServer é pre-requisito que a controladora esteja ligada e devidamente conectada ao Open LPC antes de iniciar o programa, pois a primeira ação que o OpenServer faz, quando é iniciado, é se conectar à controladora, como pode ser visto no fluxograma da Figura 3.8. Caso a conexão não seja bem sucedida o programa inicializa o robô virtual e passa a realizar uma simulação, como será detalhada na Subseção 3.1.4.

Caso seja bem sucedida, o OpenServer abre a possibilidade de um programa cliente se conectar a ele. Depois de algum cliente se conectar, o OpenServer fica aguardando comandos do programa cliente, sendo os comandos possíveis detalhados na Subseção 3.1.2. Ao receber um comando ele é enviado ao interpretador de comandos que irá extrair os dados de referências das juntas (ou de coordenadas cartesianas) do robô para gerar o comando de movimentação (ou salvar na memória). Em seguida, o OpenServer irá ler da memória os valores atuais de tais ângulos e enviá-los ao programa cliente.

O funcionamento do programa cliente por sua vez tende a ser bem mais simples. O que ele precisa fazer depois de se conectar ao OpenServer é basicamente realizar em loop: o cálculo da POSE desejada para o robô, enviar para o OpenServer as referências de ângulos de juntas via API desenvolvida, esperar receber a resposta do OpenServer pela mesma API e interpretar resposta para extrair a informação de cada ângulo referente à posição atual das juntas do robô. O funcionamento do programa cliente pode ser adaptado pelo desenvolvedor conforme for atender melhor as suas necessidades.

A controladora aparece no fluxograma como uma caixa preta, ou seja, não se sabe de fato como é implementada a programação da controladora devido ao seu código ser fechado, proprietário da Comau. Algumas informações básicas são disponibilizadas pelo manual e estão fundamentadas no Capítulo 2.

3.1.2 API desenvolvida

Para o programa cliente se comunicar com o OpenServer, é usada a API desenvolvida em cima do protocolo TCP/IP. Ela foi projetada para ser simples, de forma a possibilitar e facilitar o desenvolvimento do programa cliente em qualquer outra linguagem de programação. Portanto, foi elaborada uma forma padronizada de concatenar e desconcatenar strings, onde o programa cliente envia comandos por meio delas para o OpenServer, e recebendo informações da mesma maneira.

Neste protocolo o programa cliente é o mestre da conexão, o OpenServer, como explicando anteriormente, fica aguardando receber a string de comandos do programa cliente. Assim que ele recebe a string, ele analisa o primeiro caractere dela da seguinte forma:

- 1° caractere 'c' = Modo espaço CARTESIANO: Analise o próximo caractere.

  2° caractere 'l' = Modo apenas LEIA: Ignore o restante da string e responda com uma string concatenada contendo a posição atual do robô no espaço de cartesiano.

2° caractere ' ' = Modo padrão (leia e escreva): Desconcatene o restante da string salvando na memória a posição de referência do robô em espaço cartesiano e responda com uma string concatenada contendo a posição atual do robô no espaço de cartesiano.

- 1° caractere 'j' = Modo espaço de JUNTAS. Analise o próximo caractere.

  2° caractere 'l' = Modo apenas LEIA: Ignore o restante da string e responda com uma string concatenada contendo a posição atual do robô no espaço de juntas.

  2° caractere ' ' = Modo padrão (leia e escreva): Desconcatene o restante da string salvando na memória a posição de referência do robô em espaço de juntas e responda com uma string concatenada contendo a posição atual do robô no espaço de juntas.

- 1° caractere 'x': Comando para fechar. Ignore o restante da string, encerre a conexão com a controladora, a conexão com o programa cliente e a execução do OpenServer.

- Qualquer outro 1° caractere: Modo espelho. Apenas responda ao programa cliente com a mesma string recebida.

No modo espaço de juntas e espaço cartesiano, ao desconcatenar uma string e a mesma contiver algum caractere diferente de espaço ou números de 0 a 9, isso gerará um erro, caindo em um tratamento de exceção, resultando na não escrita das referências na memória. Logo a referência na memória permanecerá com seu valor atual.

3.1.3 Solução do Problema da Assincronia das Comunicações

As taxas de transmissão de dados podem não ser necessariamente as mesmas. A taxa de comunicação da eORL com a controladora deve ser configurada para ocorrer a uma taxa fixa de 0,4 ms, 2 ms, 4 ms, 8 ms ou 16 ms. Já a taxa de comunicação entre o programa servidor e o programa cliente não tem taxas fixas definidas, podendo inclusive a comunicação ser realizada de forma não constante, sendo o programa cliente responsável por decidir quando enviar um pacote de dados, onde o OpenServer passivamente irá responder imediatamente com outro pacote de dados e voltar a aguardar o recebimento de um novo pacote. Assim o desenvolvedor do programa cliente não tem a responsabilidade de programar o cliente para garantir uma taxa de amostragem fixa, nem do processamento

do programa cliente demorar o suficiente para ultrapassar alguma taxa de amostragem pré-definida.

No OpenServer a comunicação com a controladora, gerenciada pela eORL através do protocolo EPL, ocorre em uma thread separada da comunicação com o programa cliente, que ocorre através da API desenvolvida em cima do protocolo TCP/IP. Logo, quando uma instrução em uma thread tentar escrever em uma variável, isto pode ocasionar um conflito caso uma instrução em outra thread tente ler a variável ao mesmo tempo. Um algoritmo de comunicação entre as threads do OpenServer foi implementado de forma a superar esse problema ocasionado pela falta de sincronismo das mesmas.

O pacote de dados sensoriais das juntas do robô (oriundos da controladora) é armazenado em um ponteiro inteligente local, sobrescrevendo o anterior a medida que vai sendo recebido, e o mesmo é feito com pacote de dados das referências das juntas do robô (oriundos do cliente), mas em outro ponteiro inteligente local na outra thread. Quando o pacote de dados acaba de ser transferido para o ponteiro local, imediatamente é dada a instrução do objeto do ponteiro inteligente local ser atribuído a um ponteiro inteligente global, já criado no início na execução do programa. Um ponteiro local para a leitura de dados é criado na thread recebendo o objeto do ponteiro global. Desta forma, os ponteiros inteligentes globais atuam como uma ponte de comunicação entre os ponteiros locais, permitindo assim que a thread de comunicação com o cliente esteja parada em modo 'listen' do protocolo TCP/IP, ou seja, sem atualizar/ler os valores dos ponteiros globais, enquanto a thread de comunicação com a controladora, gerenciada pela eORL esteja trabalhando a uma taxa fixa, atualizando e lendo valores nos ponteiros inteligentes globais.

A Figura 3.9 ilustra melhor o funcionamento dos ponteiros inteligentes, onde a letra 'P' se refere a 'ponteiro', 'r' se refere a 'referência' da junta do robô, 's' se refere aos 'sensores' das juntas do robô, '1' e '2' se referem às threads e, por fim, 'g' se refere aos ponteiros 'globais'.

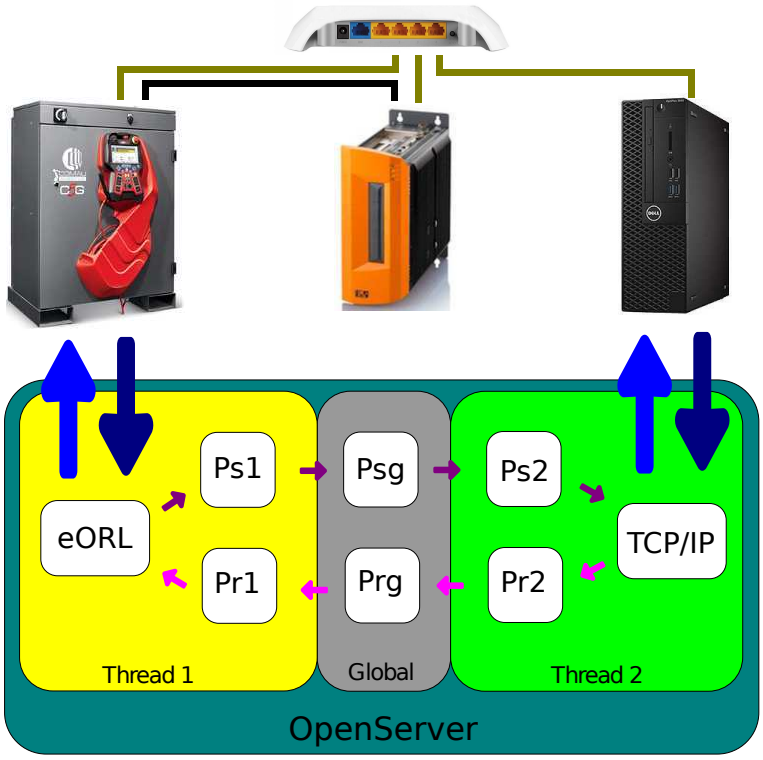


Figura 3.9 – Funcionamento dos ponteiros inteligentes no OpenServer.

Desta forma, enquanto a thread de comunicação com o cliente não atualiza na variável global os valores de referência das juntas do robô, a thread de comunicação com a controladora envia repetidamente a última referência salva. Quando a thread de comunicação com o cliente for ler os dados sensoriais da junta do robô no ponteiro global, ela irá ler o pacote de dados mais recente, pois a thread de comunicação com a controladora está continuamente atualizando o pacote de dados em tal ponteiro.

Essa abordagem soluciona o problema da assincronia de comunicação, pois quando ocorre a tentativa de atribuição de um objeto de um ponteiro inteligente local em um global em uma thread, ao mesmo tempo que acontece a tentativa de atribuição de um ponteiro inteligente global em um local por outra thread, o ponteiro inteligente tem um mecanismo de gerenciamento para tal situação. Ele cria um segundo objeto para ser atribuído enquanto o primeiro é lido. Depois, o primeiro é automaticamente destruído e o segundo objeto passa a ocupar o lugar do primeiro.

3.1.4 OpenServer em Modo de Simulação

As funções da *eORL Motion* podem ser executadas sem as funções da *eORL Comunication* com o objetivo de somente simular a movimentação do robô. Foi implementado

no OpenServer uma modalidade de simulação que é ativada automaticamente quando o OpenServer não consegue conectar a controladora. Ou seja, primeiramente o OpenServer tenta se conectar à controladora para receber o arquivo de configuração do robô real para dar inicio ao robô virtual. Caso essa conexão não seja bem sucedida, o OpenServer usa um arquivo de configuração do robô real salvo no disco rígido do Open LPC para iniciar o robô virtual.

O funcionamento do OpenServer no modo de simulação é exatamente igual ao seu funcionamento no modo padrão (controlar o robô real), com a exceção de que ao invés de enviar as referências geradas pela *eORL Motion* para a controladora *eORL Comunication* ao mesmo tempo que recebe a leitura dos sensores, o que é feito é que as referências *eORL Motion* são enviadas para o programa cliente como se fossem a leitura dos sensores. Na Figura 3.10 é possível ver o fluxograma do OpenServer quando ele está no modo de simulação.

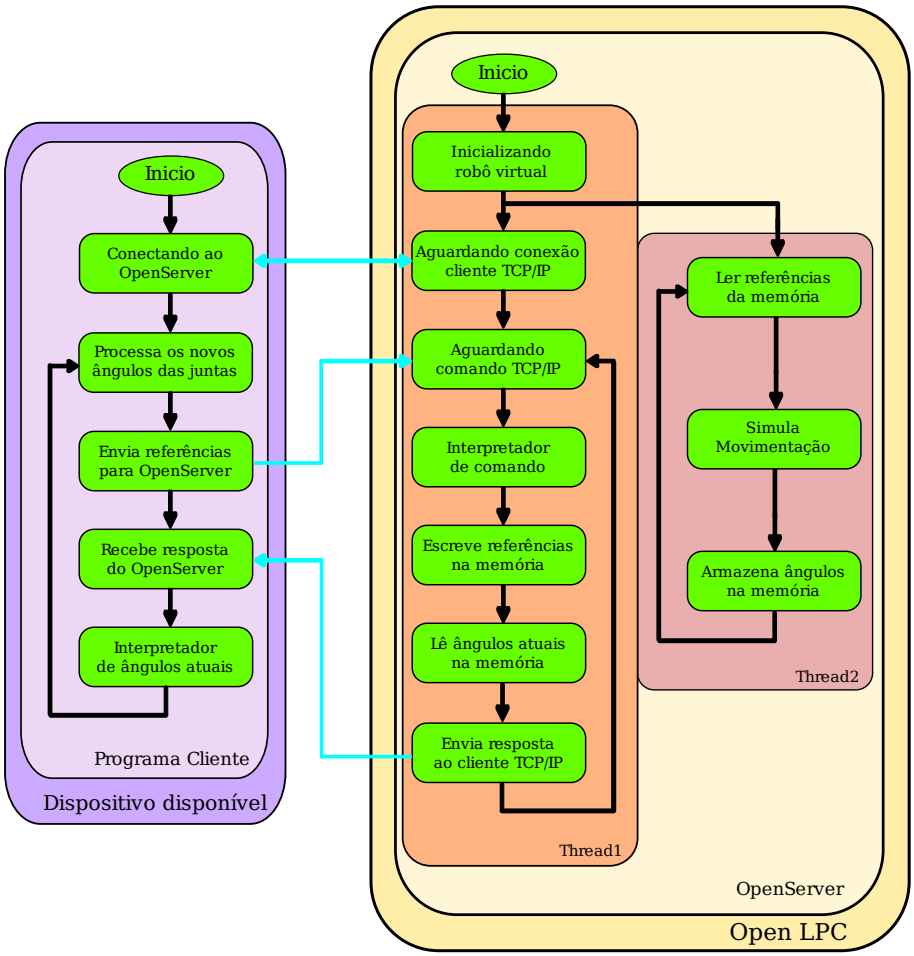


Figura 3.10 – Fluxograma do funcionamento e comunicação do programa cliente com o OpenServer no modo de simulação.

O que a controladora faz quando recebe as referências do controlador de posição é realizar o controle das juntas com seguimento de referência. É partido do pressuposto que o erro durante o seguimento de referência é insignificante, pois o objetivo do controlador é obter erro nulo. Logo, a simulação pelo OpenServer chega a ser próxima o suficiente da realidade, se não por um erro de posição insignificante durante a movimentação.

A fabricante do sistema desenvolveu um modo de simulação através da controladora, como explicado no Capítulo 2. Entretanto, essa solução exige que o OpenServer seja executado no Open LPC conectado à controladora em modo de simulação. O modo de simulação desenvolvido no OpenServer oferece a vantagem do utilizador executar o OpenServer e simular a movimentação do robô em seu computador pessoal sem precisar estar conectado a controladora.

3.2 O OpenClientQt

Para testar o OpenSever foi preciso desenvolver um programa cliente que se comunicasse com ele via a API desenvolvida. Este programa foi chamado de *OpenClientQt*, isto é, um programa cliente para OpenServer que usasse a interface gráfica *Qt*. Ele foi escrito em C++ usando bibliotecas padrão para sistemas operacionais baseados em Linux, além das bibliotecas *Qt*, e permite ao usuário enviar os ângulos de referência de cada junta do robô e visualizar a resposta das juntas em tempo real através da interface gráfica.

Para ilustrar o funcionamento do *OpenClientQt* foi feita uma captura de tela, que pode ser vista na Figura 3.11, durante o funcionamento do mesmo. No caso, a junta 1 estava na posição de 90° e, em determinado instante, sua referência foi alterada para 0°. Logo tal junta começou a variar em função do tempo em direção à posição de referência. A captura de tela foi realizada no instante em que tal junta estava na posição de $30,10983°$, como pode ser verificado na figura.

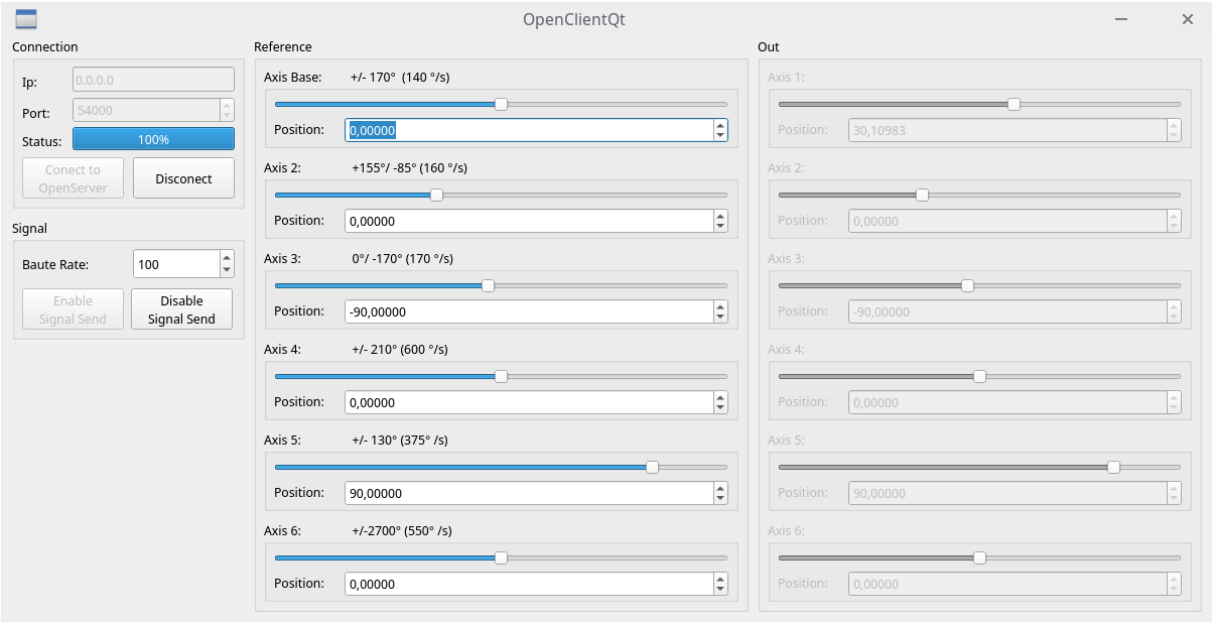


Figura 3.11 – Interface gráfica do *OpenClientQt*.

O funcionamento do programa é simplesmente, depois de iniciar a conexão com o OpenServer, ler os valores dos componentes da interface gráfica e enviar para o OpenServer, ao mesmo tempo que recebe os ângulos das juntas vindos do OpenServer e os atualiza nos componentes da interface gráfica.

Capítulo 4

Resultados e discussões

Antes de coletar os resultado, primeiramente o OpenServer foi compilado e executado no Open LPC conectado à controladora do robô, o qual ficou aguardando conexão via rede TCP/IP, como esperado.

Já o *OpenClientQt* foi compilado e executado em dois dispositivos diferentes, um Computador pessoal, do inglês *Personal computer* (PC) de arquitetura *x86-64* e um Computador de placa única, do inglês *Single board computer* (SBC) de arquitetura *arm64*. Durante a execução do OpenServer a taxa de amostragem foi aferida imprimindo no terminal o tempo que cada pacote foi enviado e depois realizando a subtração do tempo atual pelo tempo do último pacote enviado. Os dados foram salvos em uma tabela e convertidos em gráficos.

4.1 Teste com PC *x68-64*

O computador pessoal estava executando o sistema operacional GNU com Kernel Linux Genérico para arquitetura *x68-64*, com o ambiente de desktop LXDE e a distribuição era o Debian 11. É importante ressaltar o ambiente de desktop pois ele pode ser um dos principais responsáveis pelo consumo de processamento do computador, influenciando diretamente nos resultados dos testes. Por este motivo foi escolhido o LXDE, que é um dos ambientes de desktop com menos consumo de processamento que existe.

As informações de hardware possivelmente mais impactantes nos resultados coletados estão listadas abaixo.

1. CPU: Intel Core i7 8550U

2. GPU: Intel UHD Graphics 620
3. Memória RAM: DDR4 - 16GB - Single Channel - 2400MHz
4. Memória ROM: SSD NVME - 512GB - 1500MB/s
5. Porta ethernet: Gigabit

Apesar da porta de ethernet ser gigabit, o roteador utilizado só tem capacidade para conexões de 100 megabits.

Os resultados dos testes de taxa de amostragem do PC podem ser vistos na Figura 4.1, onde o eixo horizontal representa cada amostra coletada e o eixo vertical o tempo em microssegundos.

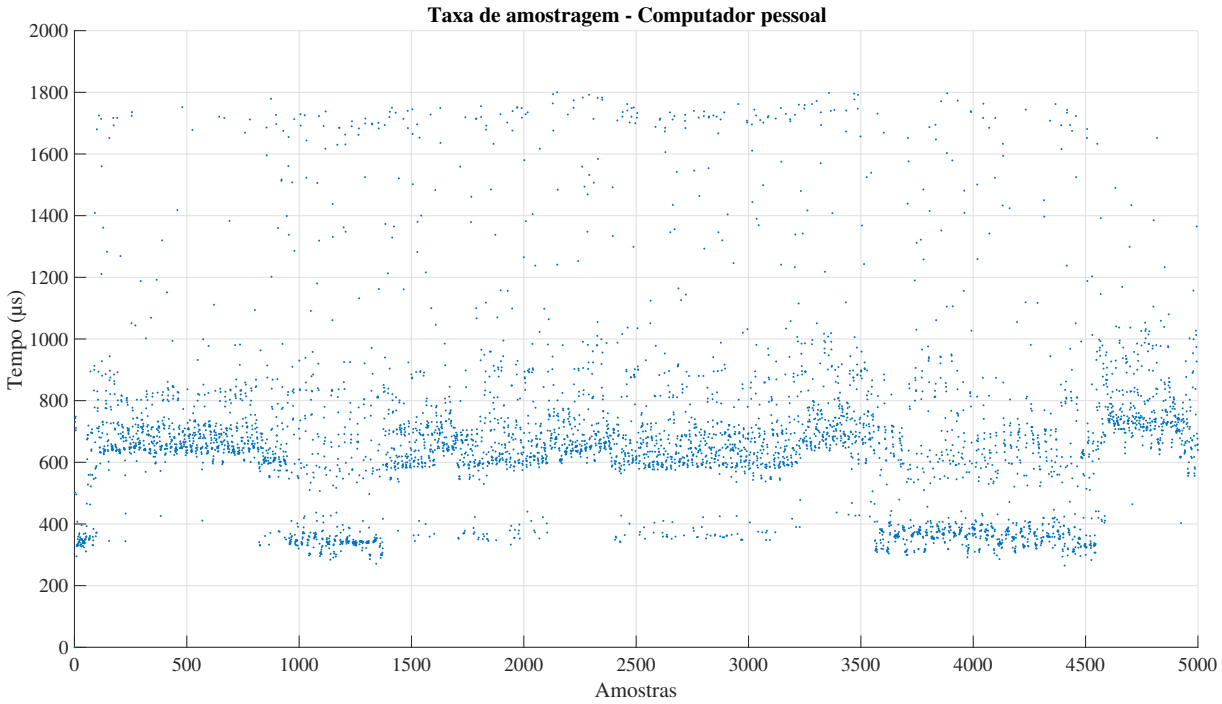


Figura 4.1 – Resultado com PC *x68-64*.

4.2 Teste com SBC *arm64*

O SBC utilizado nos testes foi o Raspberry Pi 4 Model B executando também o sistema operacional GNU com Kernel Linux Genérico, mas compilado para a arquitetura *arm64*, a distribuição era o Raspberry Pi OS Buster, que na verdade é a distribuição Debian 10 com algumas modificações desenvolvidas pela empresa Raspberry. O ambiente desktop é o LXDE-Pi, uma versão modificada do LXDE desenvolvida pela mesma empresa.

As informações de hardware possivelmente mais impactantes nos resultados coletados estão listadas abaixo.

1. CPU: ARM Cortex A72 QuadCore - 1.5GHz

2. GPU: Broadcom VideoCore VI - 500MHz

3. Memória RAM: LPDDR4 - 8GB - Single Channel - 3200MHz

4. Memória ROM: Cartão SD - 170MB/s

5. Porta ethernet: Gigabit

Apesar da porta de ethernet ser gigabit, foi utilizado o mesmo roteador do teste anterior que apenas tem a capacidade para conexões de 100 megabits.

Os resultados dos testes de taxa de amostragem do SBC podem ver vistos na Figura 4.2, onde o eixo horizontal representa cada amostra coletada e o eixo vertical o tempo em micro segundos.

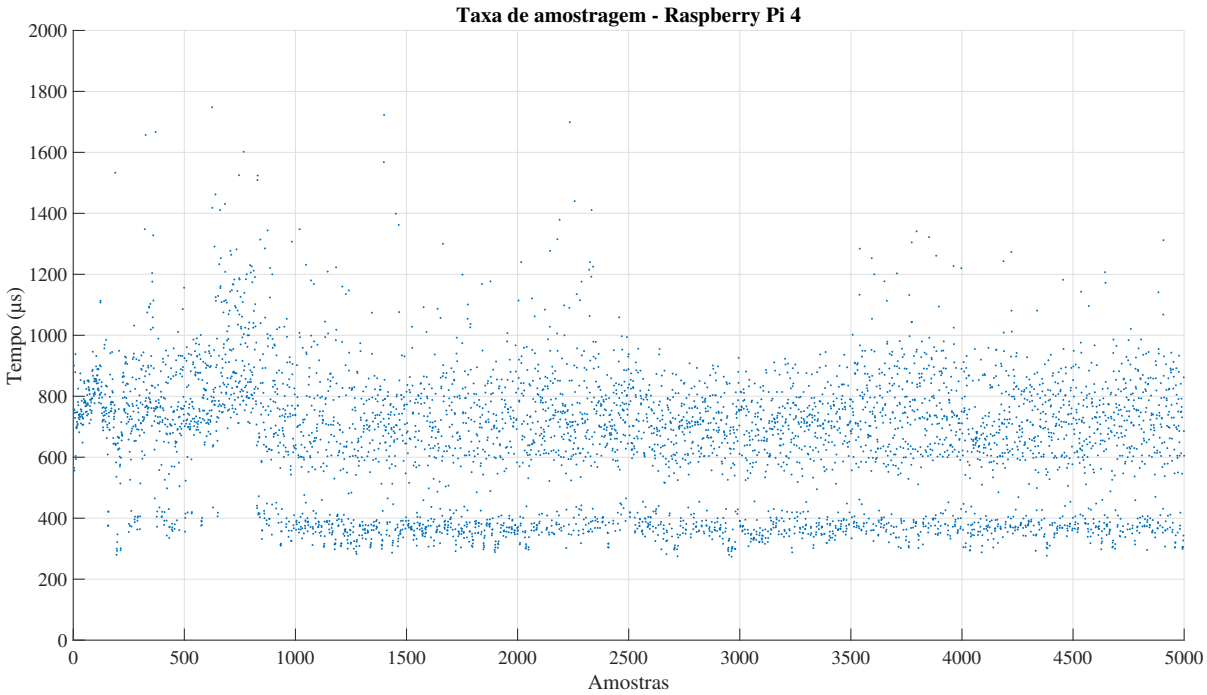


Figura 4.2 – Resultado com SBC *arm64*.

4.3 Teste de conexão com outras linguagens, dispositivos e sistemas operacionais

A fim de verificar se é possível estabelecer a conexão com o OpenServer utilizando outras linguagens de programação e dispositivos foram testadas a linguagem Python e MatLab. A conexão foi estabelecida com sucesso usando a biblioteca padrão TCP/IP dos mesmos, demonstrando que é possível desenvolver programas clientes em linguagens de programação diferentes da linguagem da biblioteca eORL. Como ambos testes foram realizados utilizando o Sistema Operacional *Windows*, foi demonstrando também que podem ser desenvolvidos softwares para outros sistemas operacionais.

Foi testada a comunicação com um dispositivo móvel rodando o sistema operacional *Android* com Kernel Linux através de um aplicativo genérico de conexão TCP/IP. O dispositivo móvel estava conectado à rede interna mas através de conexão sem fio. A comunicação foi bem sucedida, demonstrando que podem ser desenvolvidos softwares clientes para *smartphones* controlarem o braço robótico através do OpenServer e demonstrando também que a rede sem fio pode ser utilizada para conexão com o mesmo.

4.4 Discussões

Antes de tudo é importante salientar que os testes de taxa de comunicação foram realizados entre dispositivos clientes executando o sistema operacional GNU com Kernel Linux genérico, enquanto o Kernel do sistema operacional do Open LPC é o LinuxRT (Linux *Real-Time*), ou seja, enquanto o LinuxRT garante a taxa de execução da thread de comunicação entre Open LPC e controladora a cada 400 µs, a comunicação entre os dispositivos clientes e o Open LPC variou de uma taxa de 350 µs a 1800 µs. O que já demonstra que o OpenServer cumpriu o papel de realizar a comunicação a taxas diferentes da controladora e a taxas não constantes.

A taxa de comunicação entre o dispositivo cliente e o Open LPC pode ser melhorada através da utilização também de um Kernel LinuxRT no dispositivo cliente, de forma a garantir a execução da thread de comunicação a um determinado período de tempo, se isso for importante para o utilizador. Caso seja utilizado no dispositivo um Kernel que não garante a taxa de amostragem, ela vai variar conforme as características do hardware

utilizado e dos softwares em execução no mesmo tempo que o programa cliente, por exemplo, o restante do sistema operacional, interface gráfica, etc.

No caso dos testes realizados para aferir a taxa de comunicação, é possível comparar os resultados dado que eles foram coletados em hardwares diferentes, fazendo uso do mesmo sistema operacional, mesmo tipo de kernel, mesma distribuição e mesma interface gráfica. Analisando ambos gráficos, é possível perceber que no computador pessoal ocorreu uma quantidade maior de amostras acima de 1000 µs. Já no gráfico referente ao Raspberry Pi 4, as amostras ficaram mais concentradas no intervalo de 350 µs a 1000 µs. Logo, é possível afirmar que a taxa de amostragem do Raspberry Pi 4 foi mais estável. Provavelmente isso se deu devido à frequência da memória RAM do Raspberry Pi 4 utilizado ser maior, cerca de 30%. O que demonstra que taxa de comunicação varia de acordo com o hardware, quando o Kernel não garante a taxa de amostragem.

Em ambos os testes nenhum pacote foi enviado a uma taxa maior que 1800 µs. Logo, é possível estabelecer que 2 ms é a menor taxa estável, considerando uma margem de segurança, para os hardwares testados. Essa taxa provavelmente será suficiente para atender a maioria das aplicações.

O *OpenClientQt* foi um programa desenvolvido com o objetivo de testar o interpretador de comandos do OpenServer e poder movimentar o robô (real ou simulado) através do mesmo. Para isso, ele possui uma interface gráfica onde o usuário pode alterar as referências de cada junta, assim testando se o OpenServer está interpretando corretamente as referências enviadas pela API. Em uma thread separada, os dados da interface gráfica são enviados pela API e o tempo de envio é impresso no terminal.

Talvez a abordagem de usar o *OpenClientQt* para executar os testes e coletar os resultados de taxa de amostragem não tenha sido favorável. O programa teve que ser dividido em threads, a qual a thread da interface gráfica pode ter consumindo processamento, que não agregou nada ao teste. Além disso, a abordagem de executar uma thread a cada período de tempo determinado provavelmente não tenha sido tão precisa quanto se o programa tivesse apenas uma única thread que tentasse apenas enviar e receber os pacotes o mais rápido possível. Para verificar isso seria necessário alterar o código a fim de retirar a interface gráfica, dispensando a biblioteca que executa a thread com base no relógio do sistema, realizando novos testes com o novo programa e comparando os resultados com os anteriores.

Com base em todas as discussões acima é possível afirmar que a taxa máxima tem que ser analisada para cada software e hardware em que vai ser executado o programa cliente, caso seja importante que a comunicação seja realizada próximo da taxa máxima. E que o OpenServer cumpriu seu papel de permitir realizar uma taxa de comunicação diferente da controladora e inclusive taxas de comunicação variáveis.

Capítulo 5

Conclusão

Este trabalho de pesquisa foi iniciado diante da necessidade de se investigar se seria possível dar mais liberdade no desenvolvimento de aplicações para um robô industrial com controladora aberta, atendendo inclusive a demanda do Laboratório de Robótica do CEFET-MG.

Diante disso, o objetivo geral do trabalho foi desenvolver um programa com um arranjo cliente servidor que estendesse a liberdade que a solução de controladora aberta da fabricante proporciona aos programadores de robôs industriais. Tal arranjo foi desenvolvido e o objetivo geral foi atingido.

A API para enviar e receber variáveis referentes aos ângulos de juntas do robô foi criada usando o protocolo de comunicação TCP/IP, permitindo que programas clientes programados nas mais diversas linguagens de programação e sistemas operacionais se comuniquem fazendo uso dele, cumprindo assim o primeiro objetivo específico.

A função que comunica com a controladora do robô, salvando e enviando informações da memória, foi implementada, bem como a função que se comunica com o programa cliente, salvando e enviando informações da memória, atingindo assim o segundo e o terceiro objetivos específicos.

E, por fim, o problema gerado pelo assincronismo da comunicação do OpenServer com a controladora e o programa cliente foi solucionado através do uso de ponteiros inteligentes para salvar as informações na memória, garantindo assim que não ocorra problemas na atribuição de valores na memória no momento da leitura dos mesmos. Assim, o último objetivo específico foi atingido.

Foi descoberto que é possível simular a movimentação dos robôs da fabricante fazendo

uso da eORL. Logo, foi implementado um modo de simulação do robô do laboratório no OpenServer e estabelecido que quando ele não conseguir estabelecer conexão com a controladora, ele entre automaticamente neste modo.

Partiu-se da hipótese que seria possível desenvolver um programa que se comunicasse ao mesmo tempo com a controladora do robô e com um programa cliente, porém em taxas diferentes, sendo este dispositivo de outra arquitetura, o programa programado em uma linguagem diferente e rodando em um sistema operacional diferente. A hipótese foi confirmada durante o desenvolvimento de tal programa.

Uma limitação do presente trabalho é obter uma taxa estável de comunicação quando a taxa desejada se torna menor que 2 ms, bem como diagnosticar a real fonte dessa instabilidade.

Uma forma de dar continuidade ao diagnostico da instabilidade seria desenvolvendo um programa cliente mais simples, sem interface gráfica e sem divisão de threads, específico para coleta de dados de taxa de amostragem a fim de verificar se a instabilidade não foi causada pelo programa cliente, devido a abordagem utilizada no desenvolvimento do mesmo.

5.1 Perspectivas futuras

Com objetivo de atingir taxas de comunicação maiores, está em fase de estudos uma alteração da API desenvolvida para, ao invés de utilizar o protocolo TCP/IP, fazer uso protocolos de comunicação de baixa latência.

Está sendo elaborado o desenvolvimento um programa cliente do OpenSever que adiciona ao sistema robótico uma malha de controle baseada em visão computacional. Nesse desenvolvimento, uma câmera afixada no efetuador do robô filma uma figura de referência, o programa cliente processa a imagem e gera as referências que são enviados ao OpenServer, de forma a manter constante a pose (posição e orientação) do manipulador robótico em relação à pose da figura de referência. Dessa maneira, é adicionada uma câmera à estrutura desenvolvida, conforme é representado na Figura 5.1, onde (1) representa a câmera e (2) a figura de referência.

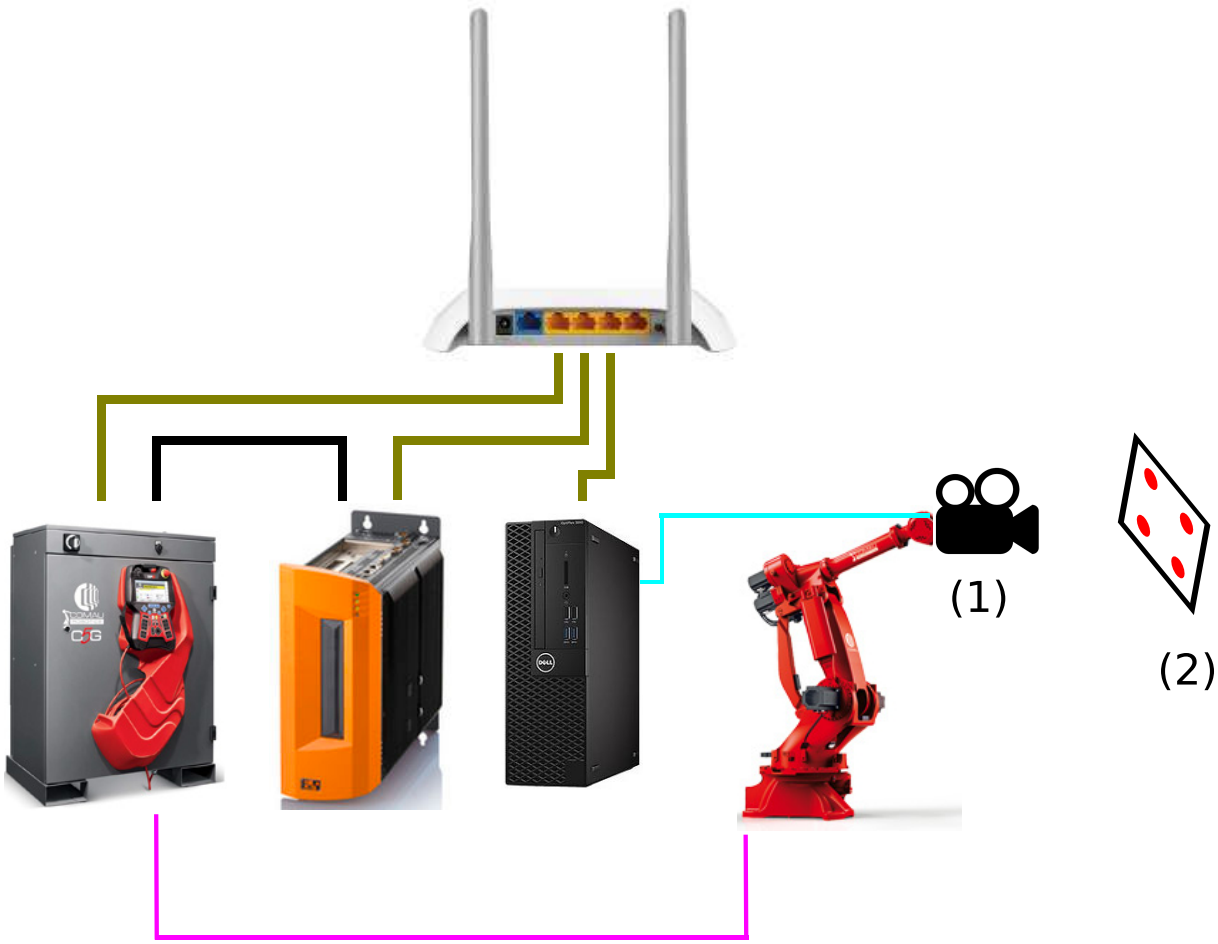


Figura 5.1 – Estrutura proposta para utilização do OpenServer.

Referências

ANTONELLI, G. et al. A Modular and task-oriented architecture for open control system: the evolution of C5G Open towards high level programming. **Workshop on the IEEE International Congress on Robotics and Automation, Lund University, Sweden.**, 2010.

BISSON, A. **Development of an interface for intuitive teleoperation of Comau manipulator robots using RGB-D sensors**. 2014. Diss. (Mestrado) – Università degli Studi di Padova.

COMAU_ROBOTICS. **C5G - Comau Open Controller - Instruction Handbook**. [S.l.], abr. 2019.

_____. **Control Unit C5G - Standard Version - Technical Specification**. [S.l.], out. 2010.

CPPREFERENCE. **Shared pointer - Cpp Reference**. [S.l.: s.n.], 2015. `https://www.cplusplus.com/reference/memory/shared_ptr`. Último acesso 2015-06-30.

FERRARA, V. **C5GOpen: The latest generation of the Industrial Robots Open Control System for University and SMEs**. 2013. Diss. (Mestrado) – Politecnico di Torino.

KUBUS D. NILSSON, K.; JOHANSSON, S. Innovative robot control architectures for demanding (research) applications. **Workshop on the IEEE International Congress on Robotics and Automation, Lund University, Sweden.**, 2010.

MEYERS, Scott. **Effective modern C++: 42 specific ways to improve your use of C++11 and C++14**. [S.l.]: O'Reilly, 2015. ISBN 9781491903995.

ROBERTI, F. et al. Open Software Structure for Controlling Industrial Robot Manipulators. **In book: Robot Manipulators Trends and Development.**, 2010.

SPONG, M.W.; HUTCHINSON, S.; VIDYASAGAR, M. **Robot Modeling and Control**. [S.l.]: Wiley, 2005. ISBN 9780471649908.