UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

CARLOS EDUARDO PEDROSO DE OLIVEIRA

CONTROLE DE UM MANIPULADOR ROBÓTICO SOB EFEITO DE DINÂMICAS NÃO MODELADAS NA BASE

Porto Alegre
2019

CARLOS EDUARDO PEDROSO DE OLIVEIRA

CONTROLE DE UM MANIPULADOR ROBÓTICO SOB EFEITO DE DINÂMICAS NÃO MODELADAS NA BASE

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

Orientador:
Prof. Dr. Renato Ventura Bayan Henriques

Porto Alegre
2019

CARLOS EDUARDO PEDROSO DE OLIVEIRA

CONTROLE DE UM MANIPULADOR ROBÓTICO SOB EFEITO DE DINÂMICAS NÃO MODELADAS NA BASE

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

Banca Examinadora:

Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

Prof. Dr. Walter Fetter Lages, UFRGS
Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Prof. Dr. Rafael Antonio Comparsi Laranja, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

_____
Marcelo Götz
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, dezembro de 2019.

AGRADECIMENTOS

Agradeço aos meus pais, Anna Paula e Carlos César, pelo suporte e carinho ao longo dos anos.

À minha irmã pela ajuda prestada e pelo currículo com foto redonda.

A toda minha família pelo carinho e bons momentos.

Ao meu orientador Renato, por acreditar no meu trabalho, esclarecer dúvidas e fornecer oportunidades ao longo destes anos de graduação.

Ao professor Walter, por todas as dúvidas esclarecidas e suporte prestado.

Aos amigos do `dssssasdf` pelas conversas sobre muitos temas.

Aos demais amigos e colegas que cruzaram pelo meu caminho e o tornaram mais agradável.

RESUMO

Neste trabalho, foi considerado o controle de um manipulador robótico montado sobre uma base não inercial. Foi utilizada a hipótese de que o manipulador não afeta os movimentos da plataforma. Controladores foram propostos para controlar o manipulador rejeitando as perturbações impostas pelo movimento da base. Foi considerado que os movimentos da plataforma são desconhecidos pelo controlador do manipulador, por isso utilizou-se uma unidade de medição inercial (IMU) com nove eixos para obter as orientações, velocidades e acelerações das juntas da plataforma. Uma plataforma de dois graus de liberdade foi descrita para representar a base não inercial e os controladores baseados no controle por torque calculado foram obtidos considerando a dinâmica da plataforma e implementados no *Robot Operating System* (ROS) utilizando o *framework* do `ros_control`. Os controladores propostos foram simulados utilizando o simulador Gazebo para uma referência de posição de junta e os resultados foram comparados utilizando o erro RMS como medida de desempenho. Os controladores que consideram a dinâmica da plataforma apresentaram significativa melhora no desempenho com relação a um controlador de torque calculado sem compensação para o movimento da plataforma.

Palavras-chave: Robótica, sistemas de controle, sistema de controle de movimento, controle de vibrações, controle MIMO.

ABSTRACT

In this paper, the control of a robotic manipulator mounted on a non-inertial base was considered. The assumption was made that the manipulator does not affect the movements of the platform. Controllers were proposed to control the manipulator by rejecting the disturbances imposed by the movement of the base. It was considered that the movements of the platform are unknown by the manipulator's controller, so an inertial measurement unit (IMU) with 9-axis was used to obtain the orientations, speeds and accelerations of the platform joints. A 2 degree of freedom platform was described to represent the non-inertial base and the controllers based on computed torque control law were obtained considering the platform dynamics and implemented in the *Robot Operating System*. (ROS) using the *framework* of `ros_control`. The proposed controllers were simulated using the Gazebo simulator for a joint position reference and the results were compared with the RMS error as the performance measure. The controllers considering the platform dynamics showed a significant performance improvement over a computed torque controller without compensation for platform movement.

Keywords: Robotics, control systems, motion control systems, MIMO control.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CTCPD	Controle de Torque Calculado Compensado para Movimentos da Plataforma e PD
CTCPID	Controle de Torque Calculado Compensado para Movimentos da Plataforma e PID
CTCNC	Controle de Torque Calculado Não Compensado para Movimentos da Plataforma
ENU	East North Up (Leste Norte para cima)
GDL	Graus de Liberdade
IMU	*Inertial Measurement Unit* (Unidade de Medida Inercial)
MIMO	*Multiple Input-Multiple Output* (Multiplas Entradas e Multiplas Saídas)
NED	North East Down (Norte Leste para baixo)
PD	Proporcional Derivativo
PID	Proporcional Integral Derivativo
ROS	*Robot Operating System* (Sistema Operacional de Robôs)
RMS	*Root Mean Square* (Raíz do Valor Quadrático Médio)
RMSE	*Root Mean Square Error* (Raiz do Erro Quadrático Médio)
SISO	*Single Input Single Output* (Entrada Única e Saída Única)
URDF	*Unified Robot Description Format* (Formato de Descrição Universal de Robôs)

1 INTRODUÇÃO

A utilização de manipuladores robóticos em plataformas não-inerciais como navios, plataformas de petróleo, ou veículos movendo-se sobre terreno irregular apresenta interações entre a dinâmica do manipulador e da base (SADRAEI; MOGHADDAM, 2015). Os esforços gerados no manipulador pelo movimento da plataforma podem comprometer o desempenho do sistema tornando-o impreciso, exigente em termos energéticos, ou até mesmo inoperável devido aos limites de torque dos atuadores.

Duas classes de problemas com respeito a relação entre as massas do manipulador e da base podem ser consideradas: a primeira é quando o movimento do robô afeta os movimentos da base e ocorre quando ambos têm massas comparáveis; a segunda refere-se ao cenário em que o robô não afeta a dinâmica da base, como no caso de um manipulador leve sobre uma base pesada (WRONKA, 2010; WRONKA; DUNNIGAN, 2011). Além disso, ainda é possível classificar o problema em outras duas classes, conforme a especificação do objetivo de trabalho: na primeira, o objetivo da tarefa é conhecido como um ponto ou trajetória com relação à base do robô. Esse é o caso de um robô montado em um navio e operando em uma referência dentro dele, por exemplo. A outra classe diz respeito ao problema em que as referências são conhecidas em relação a outro sistema de coordenadas, como o sistema de coordenadas inercial. Nesse caso, pode-se citar como exemplo um robô realizando uma tarefa em um navio, posicionando objetos em um porto.

Para controlar um manipulador montado sobre uma base móvel, o controlador deve considerar o movimento da base e calcular uma ação de controle que compense as perturbações induzidas no manipulador pela base. Assumindo que a informação sobre o movimento da plataforma não está disponível, um sensor pode ser utilizado para estimar o estado da plataforma. Dunnigan e Wronka (2011) apresentam uma comparação de técnicas de modelagem e controle, utilizando um acelerômetro para compensar parcialmente o movimento da base. Sabendo que uma unidade de medida inercial (IMU) é capaz de fornecer velocidades angulares, acelerações e através de procedimentos de fusão de dados (LAGES; HENRIQUES, 2019), a orientação de um corpo, neste trabalho os dados de uma IMU serão utilizados pelo controlador para compensar toda a dinâmica modelada da plataforma.

Esse trabalho se propõe a realizar o controle de um manipulador robótico sob efeito de dinâmicas não modeladas na base. O robô foi montado sobre uma base com dois graus de liberdade (*roll* e *pitch*). Foi feita a suposição de que os movimentos do braço robótico não afetam os movimentos da plataforma. Determinou-se como objetivo estabilizar o robô enquanto seu efetuador é movido para uma pose desejada, gerada com relação a sua base. O controlador foi implementado no *Robot Operating System* (ROS) (ROS.ORG, 2010) e os resultados obtidos através do simulador Gazebo (KOENIG; HOWARD, 2004). O robô considerado foi o Barrett WAM (BARRETT, 2011), com sete graus de liberdade e o sensor

IMU foi simulado através do plugin `GazeboRosImuSensor`.

Este documento está dividido como segue: no Capítulo 2 é apresentada a revisão bibliográfica, onde são descritos os modelos dinâmicos para manipuladores robóticos, a lei de controle de torque calculado para um robô sobre base fixa, o ROS e pacotes relacionados a esse; no Capítulo 3 o procedimento de obtenção do controlador para o robô montado sobre uma plataforma não-inercial é descrito, são apresentados os detalhes da implementação desse controlador e de pacotes ROS necessários para simulação do sistema. O experimento de simulação para validação dos controladores é descrito. Os resultados do experimento são apresentados no Capítulo 4. Por fim, no Capítulo 5 são apresentadas as conclusões.

2 REVISÃO BIBLIOGRÁFICA

2.1 Cinemática direta de um Manipulador Robótico

Um manipulador robótico pode ser modelado como uma cadeia cinemática articulada com diversos elos conectados através de juntas prismáticas ou rotacionais acionadas por atuadores (FU; GONZALES; LEE, 1987). O modelo cinemático de um robô obtém uma relação instantânea entre a configuração de juntas do manipulador e qualquer ponto associado a esse. É possível distinguir os modelos cinemáticos entre diretos e inversos. O modelo cinemático direto é capaz de informar a posição de qualquer ponto associado ao manipulador tendo como entrada a configuração de juntas, já o modelo cinemático inverso determina a configuração de juntas através de um ponto (WRONKA, 2010).

Atribuindo-se um sistema de coordenadas a cada elo do robô é possível relacionar um sistema de coordenadas ligado ao corpo com o sistema de referência (FU; GONZALES; LEE, 1987). Uma matriz de transformação homogênea composta pela matriz de rotação e o vetor de posição pode ser usada para expressar a posição e orientação de um ponto ligado ao corpo no sistema de referência (WRONKA, 2010).

Uma forma de sistematizar a obtenção do modelo de cinemática direta do manipulador é através da utilização da notação de Denavit Hartenberg. Essa notação utiliza dois ângulos e duas translações para descrever a relação entre dois sistemas de coordenadas. A representação de um ponto P_i expresso no sistema de coordenadas i em um sistema de coordenadas $i-1$ é realizada através da matriz de transformação homogênea:

$$ {}^{i-1}T_i = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\,\mathrm{sen}(\theta_i) & \mathrm{sen}(\alpha_i)\,\mathrm{sen}(\theta_i) & a_i\cos(\theta_i) \\ \mathrm{sen}(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\mathrm{sen}(\alpha_i)\cos(\theta_i) & a_i\,\mathrm{sen}(\theta_i) \\ & -\mathrm{sen}(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

onde α_i, θ_i, a_i e d_i são os parâmetros de Denavit Hartenberg para o elo i. Em Fu, Gonzales e Lee (1987) são apresentadas as regras para obtenção de tais parâmetros. A submatriz que compreende as três primeiras linhas e colunas da Equação (1) representa a matriz de rotação enquanto que o vetor $\begin{bmatrix} a_i\cos(\theta_i) & a_i\,\mathrm{sen}(\theta_i) & d_i \end{bmatrix}^T$ é o vetor de translação. Através da multiplicação de matrizes de transformação homogênea para sistemas de coordenadas sucessivos é possível obter a transformação de um sistema de coordenadas qualquer para o sistema de coordenadas da base:

$$ {}^0T_i = {}^0T_1\,{}^1T_2 \cdots {}^{i-1}T_i = \prod_{j=1}^{i} {}^{j-1}T_j \tag{2} $$

2.2 Modelo Dinâmico de um Manipulador Robótico Sobre Base Fixa

O modelo dinâmico de um manipulador robótico fornece a relação entre os esforços aplicados nas juntas do robô com as acelerações, velocidades e posições resultantes (SICILIANO; KHATIB, 2007). Em especial, a dinâmica inversa calcula os torques nas juntas com base em suas posições, velocidades e acelerações. Na literatura, são duas as formulações mais difundidas para a obtenção do modelo dinâmico: a formulação de Lagrange-Euler e a formulação de Newton-Euler. A última é mais eficiente em termos computacionais, enquanto que a primeira é mais simples e obtém um modelo explícito (WIT; BASTIN; SICILIANO, 1996).

Para um manipulador robótico com n graus de liberdade (GDL) e juntas não flexíveis, o modelo dinâmico pode ser expresso através do formalismo de Lagrange-Euler conforme Fu, Gonzales e Lee (1987):

$$\tau = M(q)\ddot{q} + V(q,\dot{q}) + G(q) = f(q,\dot{q},\ddot{q}) \tag{3}$$

onde τ é o vetor $n \times 1$ de torque nas juntas, $q, \dot{q}, \ddot{q}$ são os vetores $n \times 1$ de posição, velocidade e aceleração das juntas respectivamente, $M(q)$ é a matriz $n \times n$ de inércia, $V(q,\dot{q})$ é o vetor $n \times 1$ de forças centrífuga e de Coriólis e $G(q)$ é o vetor $n \times 1$ de forças devido à gravidade.

2.3 Modelo de um Manipulador Montado em uma Base Não Inercial

O modelo apresentado na Equação (3) descreve os torques nas juntas do robô para o caso de uma base fixa. Essa expressão não se mantém correta quando o manipulador está sobre uma base não inercial, pois essa produz novos esforços em suas juntas. Para um manipulador sobre uma plataforma móvel, Wronka (2010) obteve o modelo dinâmico através da formulação de Lagrange-Euler assumindo que o movimento do manipulador não apresenta influência no movimento da plataforma. Esse modelo pode ser expressado como:

$$\tau = M(q)\ddot{q} + V(q,\dot{q}) + G_p(q,q_p) + F_p(q_p,\dot{q}_p,\ddot{q}_p) + V_p(q_p,\dot{q}_p)\dot{q} \tag{4}$$

onde q_p é o vetor de coordenadas generalizadas da plataforma, $G_p(q,q_p)$ é o vetor de forças gravitacionais com influência da plataforma, $F_p(q_p,\dot{q}_p,\ddot{q}_p)$ são as forças inerciais, de Coriolis e centrífugas induzidas no manipulador devido ao movimento da plataforma e $V_p(q_p,\dot{q}_p)$ sãp as forças de Coriolis e centrífugas geradas pelo movimento tanto da plataforma quanto do manipulador.

Note que $M(q)$ e $V(q,\dot{q})$ dependem apenas do movimento do robô e não dependem da movimentação da plataforma.

2.4 Controle de Movimento de Manipuladores Robóticos

O controle de movimento de manipuladores robóticos pode ser classificado em duas categorias no que diz respeito à referência: o controle no espaço das juntas e o controle no espaço operacional. Essa distinção refere-se ao espaço em que a referência para o controlador é passada. No primeiro o controlador recebe como referência coordenadas de junta desejadas, enquanto que no último a referência é uma pose desejada. Neste trabalho é considerado o controle no espaço das juntas para o manipulador.

É possível realizar o controle do manipulador através de um controle independente de

cada uma das juntas do robô[1], para isto se considera cada junta um sistema *single input single output* (SISO) desacoplado das demais. Entretanto, equações (3) e (4), mostram que os manipuladores robóticos são sistemas não lineares de múltiplas entradas-múltiplas saídas (MIMO). Os termos de inércia e forças centrífugas e de Coriolis geralmente criam acoplamento entre todas as juntas dos robôs (LAGES, 2016). Dessa forma o controle independente por junta pode não ser suficiente para o seguimento de trajetórias. Neste trabalho, considera-se o controle por torque calculado do robô, um controlador MIMO não linear.

2.4.1 Controle por Torque Calculado de um Robô sobre uma Base Inercial

O controle por torque calculado considera a dinâmica do manipulador, descrita pela Equação (3) para aplicar uma linearização por realimentação. O princípio básico da linearização por realimentação é o de transformar um sistema não linear em um sistema linear, ou parcialmente linear, para então se aplicar uma lei de controle linear sobre o sistema resultante (SLOTINE; LI, 1991). Essa linearização ocorre pelo cancelamento das não linearidades do sistema. No caso do manipulador robótico, obtém-se a seguinte lei de controle:

$$\tau = M_n(q)\nu + V_n(q,\dot{q}) + G_n(q) \tag{5}$$

onde ν representa a nova entrada do sistema, $M_n(q)$ é a matriz de inércia nominal, $V_n(q,\dot{q})$ é o vetor para forças centrífugas e de Coriolis nominais e $G_n(q)$ é o vetor de forças gravitacionais nominais.

Dessa forma, ao substituir-se a Equação (5) em (3) e assumindo que os parâmetros do modelo do robô correspondem aos parâmetros do robô real, o sistema resultante é dado por:

$$\ddot{q} = \nu \tag{6}$$

que é um sistema linear e totalmente desacoplado.

Definindo o erro $e = q_r - q$, a Equação (6) pode ser expressa no espaço de estados como:

$$\begin{bmatrix} \dot{e} \\ \ddot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{7}$$

com $u = \ddot{q}_r - \nu$.

Pode-se fazer o sistema da Equação (7) convergir para sua origem através de uma realimentação estática de estados. Além disto, como na Equação (6) as dinâmicas das juntas estão desacopladas uma das outras, pode-se escolher o ganho como uma matriz diagonal. Dessa forma a realimentação de estado é o equivalente a um controlador proporcional derivativo (PD):

$$u = K \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = K_p e + K_d \dot{e} \tag{8}$$

levando a uma dinâmica de malha-fechada descrita por:

$$\ddot{e} + K_d \dot{e} + K_p e = 0 \tag{9}$$

A entrada do sistema linear ν pode ser calculada como:

$$\nu = \ddot{q}_r + K_p(q_r - q) + K_d(\dot{q}_r - \dot{q}) \tag{10}$$

[1] Conhecido como Controle Independente por Junta

e quando substituída na Equação (5), resulta na lei de controle completa do torque calculado, dada por:

$$\tau = M_n(q)\left(\ddot{q}_r + K_d(\dot{q}_r - \dot{q}) + K_p(q_r - q)\right) + V_n(q,\dot{q}) + G_n(q) \tag{11}$$

É possível notar que a Equação (5) possui a mesma estrutura de (3), com $\ddot{q}$ substituído por ν. Apesar do modelo na Equação (3) ter sido apresentado através da formulação de Lagrange-Euler, $\tau = f(q,\dot{q},\ddot{q})$ pode ser calculado através da formulação de Newton-Euler (FU; GONZALES; LEE, 1987), que é mais eficiente em termos computacionais. Dessa forma, a Equação (11) pode ser calculada por:

$$\tau = f(q,\dot{q},\nu) \tag{12}$$

onde $f(\cdot,\cdot,\cdot)$ é calculado através do procedimento iterativo de Newton-Euler.

2.5 Sistema Operacional de Robôs (Robot Operating System)

O ROS é uma coleção de *frameworks* para o desenvolvimento de aplicações robóticas, disponibilzando bibliotecas e ferramentas que permitem abstração de *hardware*, transmissão de mensagens, gerenciamento de pacotes etc (ROS.ORG, 2019a). Dentre as vantagens e princípios que regem o ROS está a reutilização de código, processamento distribuído e *software* de código aberto (QUIGLEY et al., 2009).

São quatro os conceitos fundamentais da implementação do ROS: nodos, mensagens, tópicos e serviços. Nodos são processos do ROS (ROS.ORG, 2019c). Mensagens são estruturas de dados que carregam informação relevante para alguma parte do sistema (ROS.ORG, 2019b). Tópicos são canais nomeados através dos quais nodos podem trocar mensagens utilizando um mecanismo *publish/subscribe* (ROS.ORG, 2019d). Serviços são um mecanismo de comunicação do tipo *request/reply*, que permitem troca de informações de forma síncrona (QUIGLEY et al., 2009).

A organização do *software* desenvolvido em ROS se dá através de pacotes. Os pacotes devem implementar funcionalidades específicas, podem conter nodos, arquivos de configurações, *plugins* externos etc. Com essa estrutura em pacotes busca-se permitir a fácil reutilização do código. Por ser um sistema operacional organizado em pacotes pode-se criar dependência entre arquivos de maneira simples (BARROS; LAGES, 2014).

A descrição de um robô no ROS pode ser realizada através de um arquivo *Unified Robot Description Format* (URDF). Esse é um arquivo no formato XML utilizado para descrever as propriedades cinemáticas e dinâmicas do robô. É nesse arquivo em que se descreve a forma como os elos e juntas de um robô estão conectados e também a geometria dos corpos e os parâmetros de inércia (BARROS; LAGES, 2014).

2.5.1 ros_control

O `ros_control` é um conjunto de pacotes desenvolvidos para permitir a implementação e gerenciamento de controladores genéricos no ROS. Esses pacotes foram desenvolvidos de forma a permitir execução em tempo real dos controladores, algo não alcançado com nodos e tópicos do ROS, pois esses utilizam-se de comunicação assíncrona e não operam em tempo real (CHITTA et al., 2017a). Para o controle de juntas, os ROS *controllers* permitem comunicação baseada em chamada síncrona de funções e computação em tempo real, desde que se tenha no sistema operacional no qual o ROS executa, uma política de escalonamento baseada em prioridade estritamente preemptiva (LAGES, 2016).

O elemento central desse *framework* é o nível de abstração de *hardware*, que fornece uma interface entre os ROS *controllers* e o *hardware* de diferentes robôs reais ou simulados. A classe `hardware_interface::RobotHW` fornece essa abstração. Velocidades, posições e esforços de junta são abstraídos através de interfaces de escrita e leitura, ou apenas leitura (CHITTA et al., 2017a). Essa classe contém interfaces para outros elementos além das juntas, como sensores de força, IMUs, etc. Isto garante que a implementação dos controladores não dependa do robô, ou do *hardware* específico do sensor, permitindo utilizar a mesma implementação para diferentes sistemas. O `controller_manager` é responsável por gerenciar o ciclo de vida dos controladores. É importante notar que no ROS controlador é qualquer *plugin* carregado pelo `controller_manager`, não necessariamente atuando como um controlador do ponto de vista da Engenharia de Controle.

No caso de uma simulação no Gazebo, a classe que implementa o acesso às variáveis de junta do robô é derivada da classe `RobotSim`, a classe padrão é chamada `DefaultRobotHWSim`. Além disso a sincronização de tempo entre o simulador e o ROS é feita com base no tempo simulado, enquanto que para um *hardware* real uma classe deve ser implementada para sincronizar a amostragem dos sensores, execução do controlador e escrita da ação de controle no atuador (LAGES, 2016).

Durante a execução do laço de tempo real do ROS o `ControllerManager` executa os controladores ativos um a um a cada período de amostragem, através de uma chamada da função `update()` de cada um. O controlador do robô recebe as referências através do tópico `/controller/command` o qual pode ter valores escritos por um planejador de trajetórias. O acesso às posições, velocidades e esforços de junta do robô é realizado através de funções implementadas pela classe `JointHandle`. Essas funções utilizam ponteiros para acessar as variáveis de junta no objeto da classe `RobotHW`, que implementa o acesso real ao *hardware* do robô (LAGES, 2016). A Figura 1 apresenta a arquitetura de implementação de controladores no ROS.

Na Figura 2 é apresentado o laço de tempo real do ROS. Os elementos retangulares são usados para representar as classes utilizadas e as setas representam chamada de funções, ou acesso a variáveis através de ponteiros. As setas apontam para a classe que implementa a função, ou contém a variável acessada através de ponteiros. As formas de elipse representam tópicos usados para comunicar com nodos fora do laço de tempo real (MACIEL, 2014).

A classe `JointStateController` acessa as variáveis de junta do robô através da classe `JointHandle` e publica seus valores no tópico `/joint_states`. Apesar do nome, essa classe não implementa nenhuma lei de controle. A classe `Controller` representa uma classe em que um controlador propriamente dito é implementado para o controle das juntas do robô. Para tal, método `setCommand()` é usado para escrever torque nas juntas do robô.

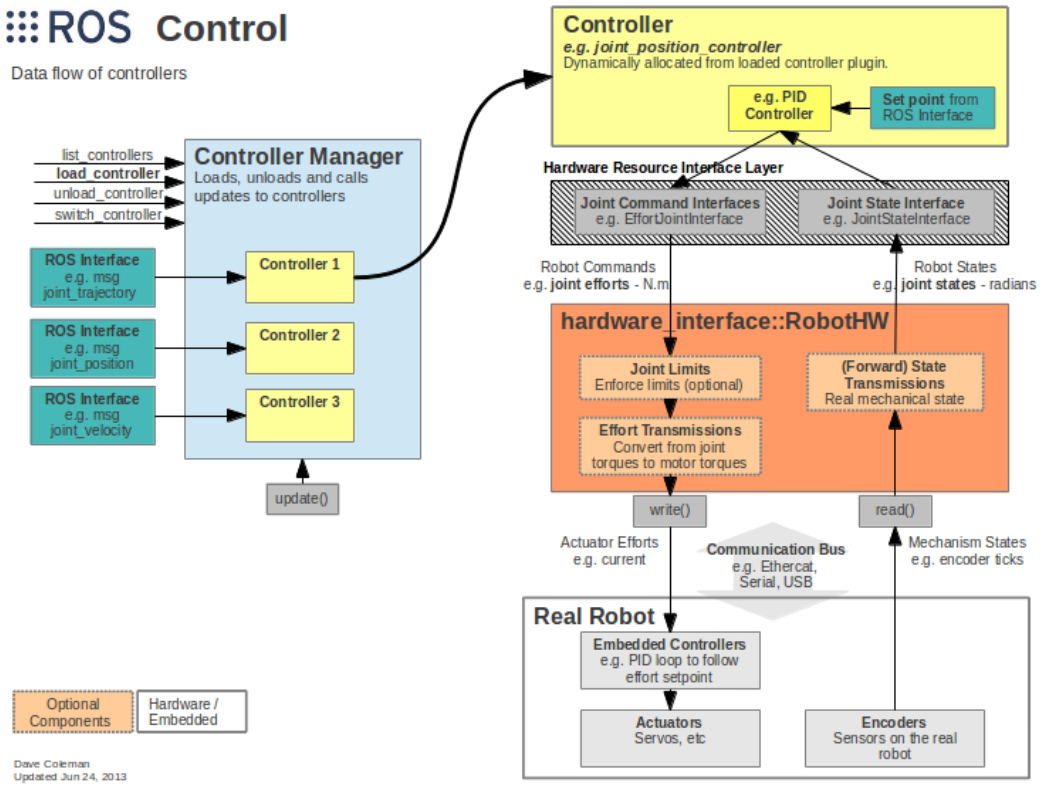
Conforme Maciel (2014), a operação do laço em tempo real segue a seguinte sequência:

1. A cada período de amostragem a simulação do Gazebo realiza a chamada da função `update()` da classe `GazeboROSControlPlugin`.

2. A classe `GazeboROSControlPlugin` chama então a função `readSim()` da classe que implementa a interface de *hardware* do robô. No caso da simulação trata-se da classe `DefaultRobotSim`, que estende `RobotHW`.

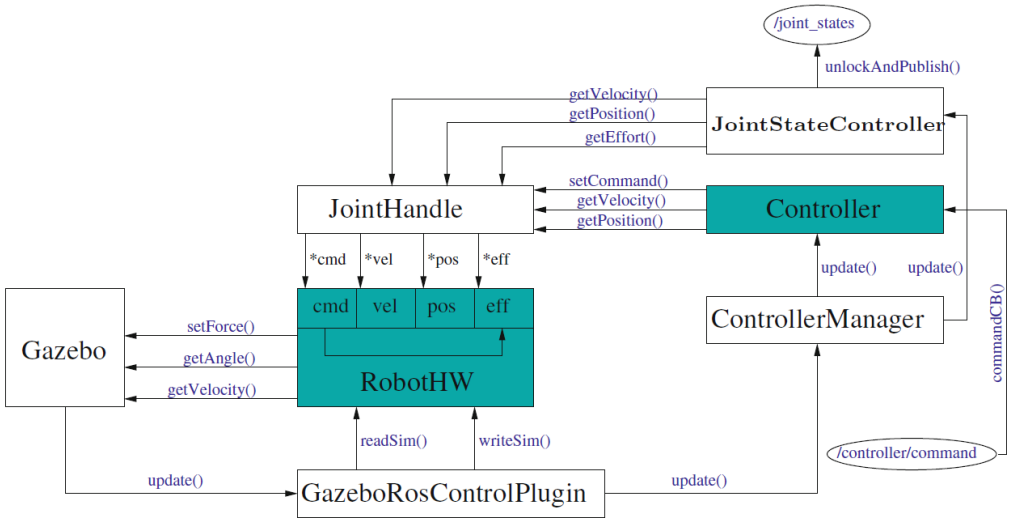
3. A classe `RobotSim` utiliza as funções `getAngle()` e `getVelocity()` para armazenar o estado do robô em variáveis internas.

Figura 1: Arquitetura para controladores no ROS.



Fonte: (CHITTA et al., 2017b)

Figura 2: Laço de tempo real do ROS.



Fonte: (LAGES, 2016)

4. A função `update()` do `ControllerManager` é chamada e esse chama as funções de `update()` de cada um dos controladores ativos.

5. Em controladores de junta essa função contém a implementação da lei de controle. As funções `getVelocity()` e `getPosition()` são utilizadas para a fechar a malha de realimentação com posições e velocidades de junta. A chamada da função `setCommand()` da classe `JointHandle` escreve a ação de controle em uma variável de `RobotSim`.

6. Uma chamada da função `update()` da classe `GazeboROSControlPlugin` faz com que essa invoque `writeSim()` de `RobotSim`, que por sua vez chamará a função `setForce()` do Gazebo, para escrever os torques nas juntas do robô.

2.5.2 Pacote ufrgs_wam

O metapacote `ufrgs_wam` (LAGES, 2016) inclui os pacotes `wam_description` e `wam_bringup`. O primeiro implementa a descrição do robô Barret WAM de 7 GDL em URDF, enquanto o segundo possui arquivos de configuração de diferentes controladores para serem utilizados com o robô, além de alguns scripts úteis. As dimensões e sistemas de coordenadas do robô Barrett WAM estão disponíveis no Anexo A.

No diretório xacro do pacote `wam_description` ficam os arquivos que descrevem sua geometria e parâmetros de inércia, os arquivos stl que são utilizados para a visualização e colisão estão na diretório meshes e também há um diretório com alguns scripts para movimentação do robô (BARROS; LAGES, 2014). O arquivo `wam.launch` carrega os parâmetros do manipulador no servidor de parâmetros, enquanto que o `gazebo.launch` carrega o `wam.launch` e inicia uma simulação no gazebo com o robô. O arquivo `display.launch` carrega a visualização do robô no RVIZ. Em todos esses launches existem parâmetros que podem ser utilizados para alterar a configuração da cena carregada.

No diretório config do pacote `wam_bringup` arquivos de configuração são salvos, contendo informações sobre o nome e tipo de controladores, ganhos e outros parâmetros relevantes a serem carregados no servidor de parâmetros do ROS. O arquivo de launch do gazebo carrega os conteúdos arquivo yaml do controlador utilizado no servidor de parâmetros, faz *spawn* de controladores e carrega o arquivo `gazebo.launch` do pacote `wam_description`.

3 METODOLOGIA

3.1 Controle por Torque Calculado de um Robô sobre uma Base não Inercial

Ao aplicar a Equação (5) ao manipulador sobre a plataforma, os termos não lineares de (4) não são cancelados por completo, o que prejudica a resposta do sistema. Para aplicar a lei de controle de torque calculado ao robô sobre a plataforma o modelo dado pela Equação (4) deve ser considerado, pois esse apresenta o movimento do robô devido aos torques aplicados em suas juntas e devido às forças produzidas pelo movimento da plataforma. Oliveira, Lages e Henriques (2019) apresentam a formulação de um controlador por torque calculado com a compensação do movimento da plataforma.

Dessa forma, considerando o modelo descrito pela Equação (4) uma versão do controle por torque calculado pode ser obtida para o caso de um manipulador montado sobre uma plataforma não inercial. A linearização por realimentação se torna (OLIVEIRA; LAGES; HENRIQUES, 2019):

$$\tau = M_n(q)\nu + V_n(q,\dot{q}) + G_{pn}(q,q_p) + F_{pn}(q_p,\dot{q}_p,\ddot{q}_p) + V_{pn}(q_p,\dot{q}_p)\dot{q} \tag{13}$$

onde ν pode ser calculado como no caso para base fixa, através da Equação (10). Além disso, esse procedimento produz uma dinâmica do erro na mesma forma da Equação (9). É importante notar que para calcular a Equação (13) informações sobre o movimento da plataforma são necessárias. Supõem-se que esses dados não estão disponíveis, dessa forma é preciso utilizar de algum sensor para medi-los. Para tal, uma IMU pode ser utilizada, sendo montada sobre o topo da plataforma

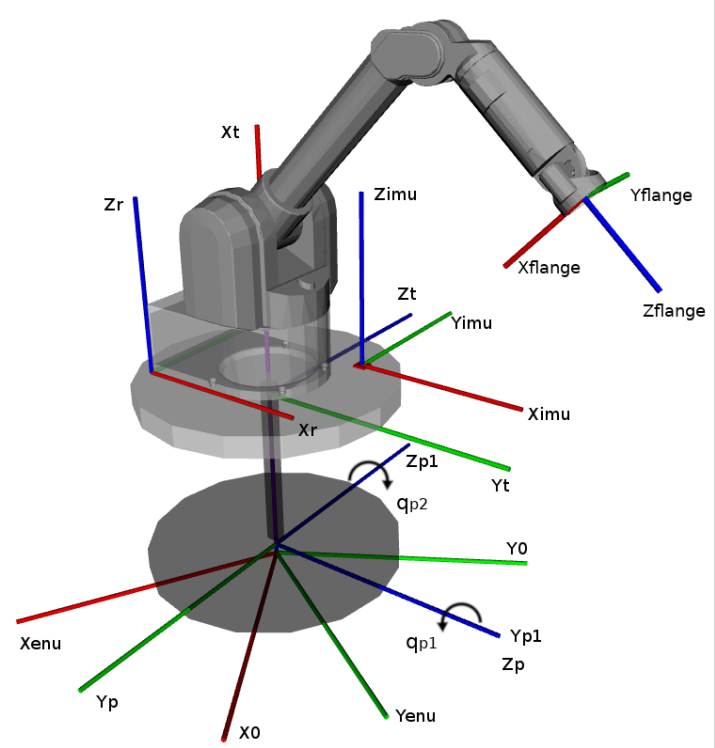
Uma IMU de 9 eixos é considerada, pois a Equação (13) assume o conhecimento de $q_p, \dot{q}_p, \ddot{q}_p$ e como esses dispositivos possuem um acelerômetro, giroscópio e magnetômetro é possível obter esses dados. Em algumas IMUs é incluído um microprocessador no qual um procedimento de fusão sensorial é executado para estimar a orientação do sensor. Geralmente um filtro de Kalman estendido (GOODWIN; SIN, 1984) é utilizado para tais estimativas. Em Lages e Henriques (2019) são apresentados detalhes de como utilizar o filtro de Kalman estendido para estimar a orientação de IMUs que não possuem fusão sensorial interna.

As medidas de velocidade angular e aceleração linear de uma IMU são tomadas com relação ao sistema de coordenadas inercial e representadas no sistema de coordenadas do corpo do sensor (KOK; HOL; SCHÖN, 2017). Já a estimativa de orientação é medida com relação a um sistema de coordenadas com eixos alinhados com o norte magnético e a vertical e também representada nesse sistema de coordenadas. A convenção *earth, north, up* (ENU) é adotada neste trabalho, pois é a adotada pelo ROS (FOOTE; PURVIS, 2010).

Dessa forma, é definido que o eixo X aponta para o leste, o eixo Y aponta para o norte magnético e o eixo Z aponta para cima. Para o caso em que o *hardware* da IMU segue a convenção *north, east, down* (NED), o *driver* desse dispositivo deve converter os dados para a convenção ENU.

A plataforma possui duas juntas rotacionais coincidentes, isto é $q_p = \begin{bmatrix} q_{p1} & q_{p2} \end{bmatrix}^T$, que produzem o movimento de *roll* e *pitch*. A Figura 3 apresenta os sistemas de coordenadas da IMU, do topo e base da plataforma, da base e flange do robô, ENU, o *frame* inercial e das juntas da plataforma. O ângulo entre X_{p1} e X_p em torno de Z_p é q_{p1} e o ângulo entre X_{p2} e X_{p1} em torno de Z_{p1} é q_{p2}.

Figura 3: Sistemas de coordenadas da plataforma.



Fonte: Adaptada de (OLIVEIRA; LAGES; HENRIQUES, 2019)

Para se obter q_p através da estimação de orientação da IMU fixa ao topo da plataforma, é necessário considerar a orientação da base da plataforma com relação ao sistema de coordenadas inercial ${}^0R_p$, a orientação do topo da plataforma com respeito à sua base ${}^pR_t(q_p)$, a orientação da IMU em relação ao topo da plataforma ${}^tR_{imu}$, a orientação da IMU com relação ao sistema de coordenadas ENU ${}^{enu}R_{imu}$ e a orientação desse último em relação ao *frame* inercial.

A orientação do sistema de coordenadas do topo da plataforma expressa na referência inercial pode ser obtida em função das varáveis de junta da plataforma através de:

$$ {}^0R_t = {}^0R_p {}^pR_t(q_p) $$

Porém as variáveis de junta da plataforma são desconhecidas. Como a orientação do sistema de coordenadas da IMU com respeito ao sistema de coordenadas inercial pode ser calculada como:

$$ {}^0R_{imu} = {}^0R_{enu} {}^{enu}R_{imu} $$

e além disso, a orientação do frame da IMU com relação ao topo da plataforma pode ser obtida por:

$$ {}^{t}R_{imu} = {}^{0}R_{t}^{T}\ {}^{0}R_{imu} = \left({}^{0}R_{p}\,{}^{p}R_{t}(q_p)\right)^{T}\,{}^{0}R_{enu}\,{}^{enu}R_{imu} $$

multiplicando essa expressão por ${}^{t}R_{imu}^{T}$ à direita e ${}^{p}R_{t}(q_p)$ à esquerda, obtém-se:

$$ {}^{p}R_{t}(q_p) = {}^{0}R_{p}^{T}\ {}^{0}R_{enu}\ {}^{enu}R_{imu}\ {}^{t}R_{imu}^{T} \tag{14} $$

que pode ser simplficada para:

$$ {}^{p}R_{t}(q_p) = {}^{enu}R_{p}^{T}\ {}^{enu}R_{imu}\ {}^{t}R_{imu}^{T} \tag{15} $$

Observa-se que ${}^{enu}R_{p}$ e ${}^{t}R_{imu}$ são constantes e que ${}^{enu}R_{imu}$ a estimação da orientação obtida pela IMU, geralmente é expressada como um quaternion unitário ${}^{enu}Q_{imu} = w + ix + jy + kz$. Esse quaternion pode ser representado como uma matriz de rotação através da seguinte expressão:

$$ {}^{enu}R_{imu} = \begin{bmatrix} w^2+x^2-y^2-z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & w^2-x2+y^2-z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & w^2-x^2-y^2+z^2 \end{bmatrix} \tag{16} $$

A matriz de rotação ${}^{p}R_{t}(q_p)$ também pode ser obtida em função dos parâmetros de Denavit-Hartenberg da plataforma. Com base no sistema de coordenadas proposto para as juntas, obtém-se os seguintes parâmetros referentes apenas à parte rotacional:

Tabela 1: Parâmetros de Denavit-Hartenberg referentes à rotação.

Elo	θ	α
p_1	q_{p1}	$\pi/2$
p_2	q_{p2}	0

Tomando da matriz de transformação homogênea, descrita pela Equação (2), apenas a matriz de rotação (matriz formada pelas três primeiras linhas e colunas), pode-se obter a orientação do topo da plataforma no sistema de sua base através de:

$$ \begin{aligned} {}^{p}R_{t}(q_p) &= {}^{p}R_{p1}\ {}^{p_1}R_{t} \\ &= \begin{bmatrix} \cos q_{p1} & 0 & \operatorname{sen} q_{p1} \\ \operatorname{sen} q_{p1} & 0 & -\cos q_{p1} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \cos q_{p2} & -\operatorname{sen} q_{p1} & 0 \\ \operatorname{sen} q_{p2} & \cos q_{p1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos q_{p1}\cos q_{p2} & -\cos q_{p1}\operatorname{sen} q_{p2} & \operatorname{sen} q_{p1} \\ \operatorname{sen} q_{p1}\cos q_{p2} & -\operatorname{sen} q_{p1}\operatorname{sen} q_{p2} & -\cos q_{p1} \\ \operatorname{sen} q_{p2} & \cos q_{p2} & 0 \end{bmatrix} \end{aligned} \tag{17} $$

Dessa forma, após o cálculo de ${}^{p}R_{t}(q_p)$ a partir da Equação (15), os ângulos de junta da plataforma podem ser extraídos da Equação (17) através de:

$$ q_p = \begin{bmatrix} \operatorname{atan2}(r_{13}, -r_{23}) \\ \operatorname{atan2}(r_{31}, r_{32}) \end{bmatrix} \tag{18} $$

onde r_{ij} é o elemento da linha i e coluna j de ${}^{p}R_{t}$.

As velocidades das juntas para a plataforma podem ser obtidas através das velocidades angulares medidas pela IMU, ${}^{imu}\omega_{imu}$. Entretanto, as medidas da IMU são representadas em seu próprio *frame* (KOK; HOL; SCHÖN, 2017) e podem ser transformadas para as coordenadas da plataforma através de:

$$ {}^{p}\omega_t = {}^{p}R_t(q_p)\ {}^{t}R_{imu}\ {}^{imu}\omega_{imu} $$

onde ${}^{p}\omega_t$ é a velocidade angular do topo da plataforma representada na base da plataforma.

Por outro lado, a velocidade angular do topo da plataforma pode ser calculada a partir das velocidades das juntas da plataforma por:

$$ {}^{p}\omega_t = J_\omega(q_p)\dot{q}_p $$

onde $J_\omega(q_p)$ é o Jacobiano relacionando as velocidades de juntas da plataforma com a velocidade angular do topo da plataforma. O cálculo da coluna i de $J_\omega(q_p)$ é dado por $J_{\omega i}(q_p) = {}^{p}R_{i-1}\ {}^{i-1}\hat{Z}_{i-1}$ (WHITNEY, 1972; FU; GONZALES; LEE, 1987). Assim:

$$ J_\omega(q_p) = \begin{bmatrix} 0 & \operatorname{sen} q_{p1} \\ 0 & -\cos q_{p1} \\ 1 & 0 \end{bmatrix} $$

Pode-se então usar a pseudo inversa de $J_\omega(q_p)$ para calcular:

$$ \dot{q}_p = J_\omega^{\dagger}(q_p)\ {}^{p}\omega_t $$

com

$$ \begin{aligned} J_\omega^{\dagger}(q_p) &= \left(J_\omega^T(q_p)J_\omega(q_p)\right)^{-1} J_\omega^T(q_p) = \begin{bmatrix} 0 & 0 & 1 \\ \operatorname{sen} q_{p1} & -\cos q_{p1} & 0 \end{bmatrix} \\ &= J_\omega^T(q_p) \end{aligned} $$

Finalmente, as velocidades de junta da plataforma podem ser calculadas a partir da velocidade angular medida pela IMU como:

$$ \begin{aligned} \dot{q}_p &= J_\omega^{\dagger}(q_p)\ {}^{p}R_t(q_p)\ {}^{t}R_{imu}\ {}^{imu}\omega_{imu} \\ &= \begin{bmatrix} \cos q_{p2} & 0 & \operatorname{sen} q_{p2} \\ 0 & 1 & 0 \end{bmatrix}\ {}^{imu}\omega_{imu} \end{aligned} \tag{19} $$

Através de procedimento similar, as acelerações de junta da plataforma podem ser calculadas através da aceleração medida pela IMU, ${}^{imu}a_{imu}$. Deve-se observar que a aceleração medida pela IMU inclui a gravidade, que deve ser removida para calcular a aceleração linear do topo da plataforma:

$$ {}^{p}a_{imu} = {}^{p}R_t(q_p)\ {}^{t}R_{imu}\left({}^{imu}a_{imu} - {}^{0}R_{imu}^{T}g\right) \tag{20} $$

onde ${}^{imu}a_{imu}$ é a aceleração do ponto em que a IMU está fixada sobre o topo da plataforma, representado no *frame* da base da plataforma e $g = \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^T$ é o vetor de gravidade.

Pode-se utilizar o Jacobiano para obter a velocidade linear do ponto de fixação da IMU sobre o topo da plataforma através de:

$$ {}^{p}v_{imu} = J_v(q_p)\dot{q}_p \tag{21} $$

onde $J_v(q_p)$ é o Jacobiano relacionando as velocidades das juntas da plataforma com a velocidade linear do ponto de montagem da IMU. O cálculo da coluna i de $J_v(q_p)$ é dado por $J_{vi}(q_p) = {}^{p}R_{i-1}\left({}^{i-1}\hat{Z}_{i-1} \times {}^{i-1}P_{imu}\right)$ (WHITNEY, 1972; FU; GONZALES; LEE, 1987). Portanto:

$$J_{v1}(q_p) = {}^{p}\hat{Z}_p \times \left({}^{p}R_1\left({}^{1}P_t + {}^{1}R_t\,{}^{t}P_{imu}\right)\right)$$
$$J_{v2}(q_p) = {}^{p}R_1\left({}^{1}\hat{Z}_1 \times \left({}^{1}P_t + {}^{1}R_t\,{}^{t}P_{imu}\right)\right)$$

onde a origem do sistema de coordenadas no topo da plataforma, relativa ao *frame* da base da plataforma é dada por ${}^{p}P_t$ e a origem do sistema de coordenadas da IMU com relação ao *frame* de topo da plataforma é dada por ${}^{t}P_{imu}$.

Derivando a Equação (21) com relação ao tempo, a aceleração linear do ponto de fixação da IMU é:

$${}^{p}a_{imu} = \frac{d}{dt}\left({}^{p}v_{imu}\right) = \dot{J}_v(q_p,\dot{q}_p)\dot{q}_p + J_v(q_p)\ddot{q}_p \tag{22}$$

com $\dot{J}_v(q_p,\dot{q}_p)$ cuja coluna i é dada por $\dot{J}_{vi}(q_p) = {}^{p}R_{i-1}\left({}^{i-1}\hat{Z}_{i-1} \times \left({}^{i-1}\hat{Z}_{i-1} \times {}^{i-1}P_{imu}\right)\right)\dot{q}_{pi}$ (FU; GONZALES; LEE, 1987). Dessa forma:

$$\dot{J}_{v1}(q_p) = \left({}^{p}\hat{Z}_p \times \left({}^{p}\hat{Z}_p \times \left({}^{p}R_1\left({}^{1}P_t + {}^{1}R_t\,{}^{t}P_{imu}\right)\right)\right)\right)\dot{q}_{p1}$$
$$\dot{J}_{v2}(q_p) = {}^{p}R_1\left({}^{1}\hat{Z}_1 \times \left({}^{1}\hat{Z}_1 \times \left({}^{1}P_t + {}^{1}R_t\,{}^{t}P_{imu}\right)\right)\right)\dot{q}_{p2}$$

Assim, a partir das equações (20) e (22) é possível calcular a aceleração de junta da plataforma através das medidas do acelerômetro da IMU como:

$$\ddot{q}_p = J_v^{\dagger}(q_p)({}^{p}R_t(q_p)\,{}^{t}R_{imu}({}^{imu}a_{imu} - {}^{0}R_{imu}^{T}g) - \dot{J}_v(q_p,\dot{q}_p)\dot{q}_p) \tag{23}$$

Portanto, mostra-se que é possível calcular as leis de controle definidas pelas equações (13) e (10) utilizando os sensores das juntas do manipulador e as posições, velocidades e aceleração de junta da plataforma obtidas através das equações (18), (19) e (23). Aqui também pode-se usar a formulação de Newton Euler para expressar a lei de controle através de:

$$\tau = f(q_e,\dot{q}_e,\nu_e) \tag{24}$$

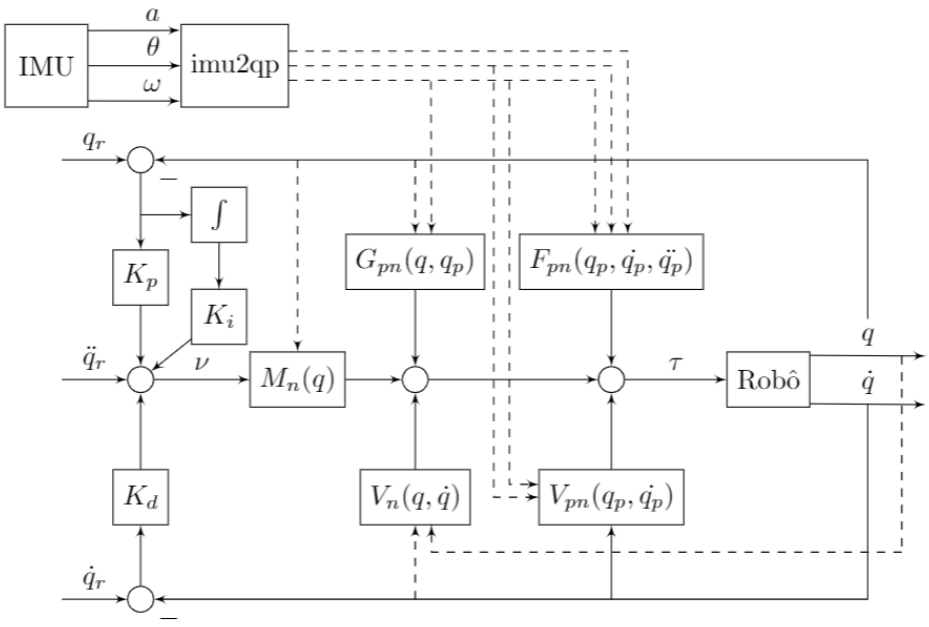
onde $q_e = \begin{bmatrix} q_p & q \end{bmatrix}^T$, $\dot{q}_e = \begin{bmatrix} \dot{q}_p & \dot{q} \end{bmatrix}^T$ e $\nu_e = \begin{bmatrix} \ddot{q}_p & \nu \end{bmatrix}^T$.

A Figura 4 apresenta o diagrama de blocos do controlador.

3.2 Representação da Plataforma

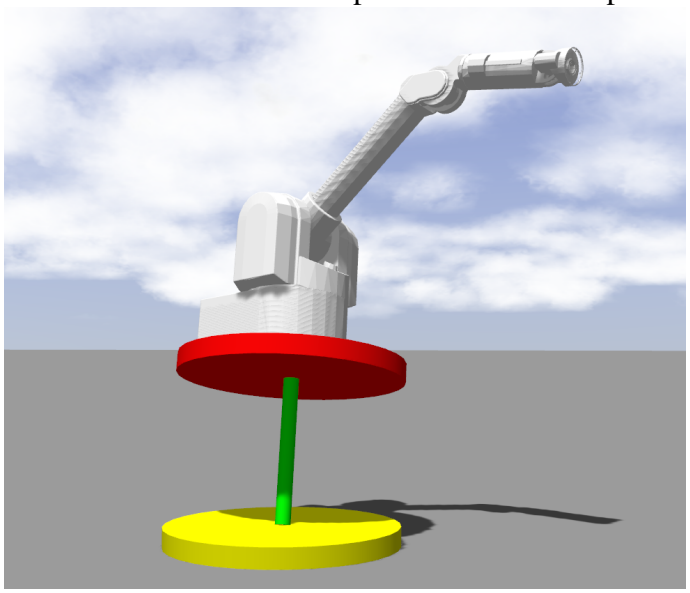
Para se obter uma representação do robô sobre uma base oscilatória, foi desenvolvido um pacote no ROS com a descrição de uma plataforma de dois graus de liberdade (*roll* e *pitch*). Esse pacote é denominado `platform_description` e integra o meta pacote `platform` (OLIVEIRA, 2019a), junto com os pacotes `platform_bringup` e `platform_pose_pub`. A visualização da plataforma desenvolvida no ambiente de simulação, com o robô Barrett WAM montado é apresentada na Figura 5.

Figura 4: Diagrama de Blocos do Controlador de Torque Calculado com Compensação de Movimentos da Plataforma



Fonte: Elaborada pelo autor.

Figura 5: Robô Barrett WAM sobre a plataforma descrita para a simulação

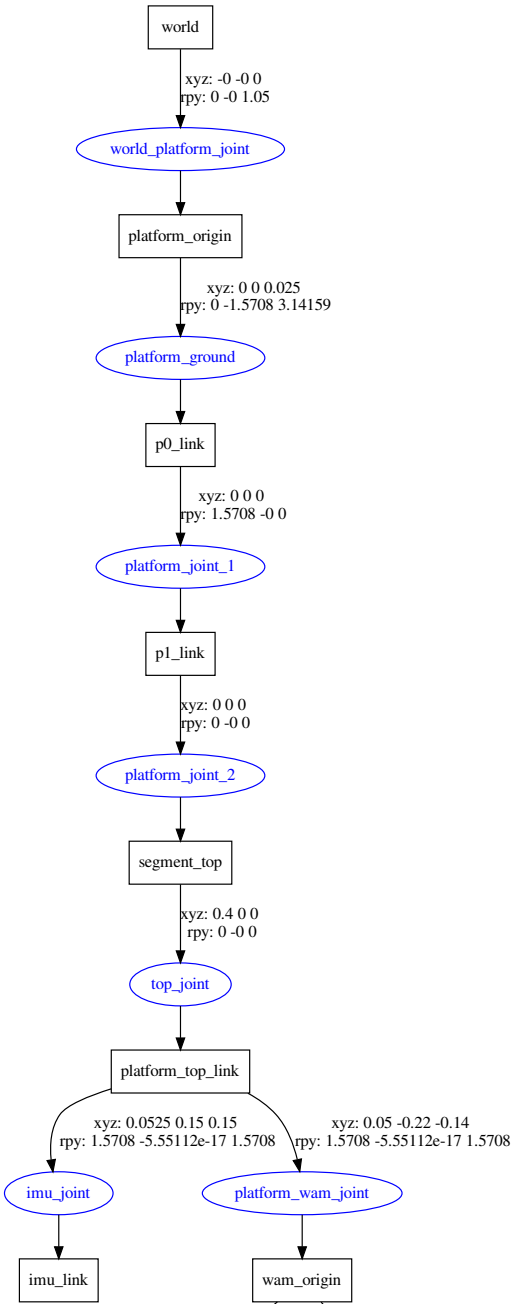


Fonte: Elaborada pelo autor.

O pacote `platform_description` possui os arquivos `xacro` para descrição da cadeia cinemática da plataforma. O arquivo principal desenvolvido para essa tarefa é o `platform.urdf.xacro`. Nesse arquivo a plataforma é formada por cinco elos e cinco juntas, duas das quais são rotacionais com a origem coincidente. O sistema de coordenadas adotado foi implementado para garantir a compatibilidade com as regras de Denavit–Hartenberg para designação de eixos, pois esse é relevante para o equacionamento das velocidades de junta da plataforma em função das leituras da IMU. A Figura 6 apresenta a

árvore cinemática da plataforma com a IMU fixada sobre o elo superior, o mesmo no qual o robô Barrett WAM está montado. Os parâmetros de inércia dos elos da plataforma foram calculados para garantir a essa massa e inércia maiores que os do robô, considerando a massa do WAM em torno de 27 kg (BARRETT, 2011).

Figura 6: Cadeia cinemática da plataforma



Fonte: Elaborada pelo autor.

Para garantir o funcionamento da plataforma com o `ros_control`, foi adicionado um elemento *transmission* referente às duas juntas rotacionais. Esse elemento de transmissão descreve a relação entre o atuador e a junta. Para a interface de *hardware* da junta foi escolhida interface do tipo PositionJointInterface, pois ao ser utilizada com um controlador do tipo position_controllers/JointPositionController é possível comandar diretamente os

valores dos ângulos das juntas. Dessa forma pode-se arbitrar diretamente a pose da plataforma. O sensor IMU utilizado para a simulação do sistema é o `GazeboRosImuSensor`. O parâmetro `rpyOffset` configura a orientação do sistema de coordenadas ENU com relção ao sistema de coordenadas inercial. Para este trabalho foi configurada um rotação em torno do eixo z do sistema inercial de 1.05 rad. O elo associado à IMU é o `imu_link`.

A conexão entre a plataforma e o sistema de coordenadas inercial, bem como com o robô é realizada no arquivo `wam_platform.urdf.xacro`, onde uma junta fixa entre a plataforma e o *world* do Gazebo foi estabelecida, bem como uma junta fixa para a plataforma e a origem do robô. Essa conexão é realizada em um arquivo separado ao de descrição da plataforma, para permitir o uso da mesma com outros manipuladores sem que haja alteração do seu arquivo de descrição. Nesse caso, bastaria criar um URDF novo para a conexão do robô específico.

Além da descrição da plataforma, o pacote `platform_description` possui os arquivos de *launch* para carregar no servidor de parâmetros a descrição do robô. O parâmetro `robot_description` é depois utilizado pelo controlador para obter a cadeia cinemática do robô. Há também um arquivo de configuração da visualização do robô sobre a plataforma no ambiente de visualização Rviz. Esse arquivo não tem utilidade do ponto de vista do controle do robô, ou da descrição da plataforma, mas serve como uma ferramenta que possibilita visualizar os sistemas de coordenadas e até movimentar as juntas do sistema. Para este trabalho essa visualização serviu de ferramenta para conferir o posicionamento e orientação dos eixos dos sistemas de coordenadas da plataforma.

O pacote `platform_bringup` possui o arquivo de *launch* da simulação, o `gazebo.launch`. Esse arquivo carrega o *launch* da plataforma. Um de seus parâmetros é o nome do controlador que será usado. Esse nome deve corresponder ao nome de um arquivo de configuração no formato yaml, armazenado no diretório `config` do pacote. Esses arquivos apresentam as configurações necessárias para o controlador funcionar. No caso dos controladores das juntas da plataforma, o arquivo `platform.yaml` foi adicionado à pasta. Nesse arquivo um controlador do tipo `JointPositionController` é especificado para cada uma das juntas da plataforma. Observe que os controladores da plataforma são usados para gerar o distúrbio na base do robô na simulação e em nenhum momento a ação de controle do controlador do robô atua sobre esses controladores.

O pacote `platform_pose_pub` implementa um *publisher* para a referência do controlador da plataforma. O nodo responsável por essa tarefa publica as posições de junta desejadas nos tópicos `roll_controller/command` e `pitch_controller/command`. O sinal enviado a cada uma das juntas é um seno com amplitude e fase descritas na seção 3.4.

A representação da plataforma é utilizada tanto para gerar os distúrbios na base do manipulador no ambiente de simulação, quanto fornecer ao controlador uma representação de uma cadeia cinemática com dois graus de liberdade rotacionais em *roll* e *pitch*. Essa representação é utilizada para a construção do *solver* da dinâmica inversa. Para a implementação completa do pacote `platform` consultar Oliveira (2019a).

3.3 Implementação dos Controladores

Os controladores propostos na seção 3.1 foram implementados no ROS através do pacote `platform_computed_torque_controller` (OLIVEIRA, 2019b). Como os nodos do ROS não têm capacidade de processamento em tempo real e seus tópicos fornecem um método de comunicação assíncrono, esses não são recomendados para o controle de junta

a baixo nível (LAGES, 2016). Para essa tarefa os pacotes do `ros_control` fornecem a capacidade de implementar controladores em tempo real que não dependem de *hawdware* específico de robôs (CHITTA et al., 2017a).

Em Lages (2016) a lei de controle por torque calculado é implementada no ROS com capacidade de ser executado em tempo real. Para tal, um controlador do ROS foi desenvolvido e a lei de controle foi calculada usando a *Orocos Kinematics and Dynamics Library* (KDL) (BRUYNINCKX, 2001), em particular seu *solver* recursivo de Newton-Euler. Essa arquitetura é utilizada como uma base para a implementação neste trabalho.

A classe `PlatformComputedTorqueController` foi criada para implementar o controlador e é derivada da classe `MultiInterfaceController` que estende `Controller-Base`. Essa classe permite que o controlador utilize mais de uma única interface de *hardware*, assumindo-se que essas interfaces sejam expostas pelo *hardware* do robô. Como a lei de controle utilizada calcula um comando de torque, o controlador utiliza uma `EffortJointInterface`, que garante que os *handles* associados com a escrita de torque nas juntas estarão disponíveis. Uma interface do tipo `ImuSensorInterface` também é definida na declaração da classe do controlador para permitir acesso em tempo real ao sensor IMU. Para a simulação desenvolvida foi utilizada a classe padrão do Gazebo para abstração de *hardware* do robô, a classe `DefaultRobotHWSim`. Como essa classe não implementa interface para o sensor IMU, o controlador acessa as leituras do sensor através do tópico `/imu/data`. Para o caso específico da simulação em que a sincronização entre o tempo do simulador e o ROS é feita com base no tempo simulado, isto não apresenta maiores problemas, porém para garantir uma implementação em tempo real em um sistema físico é necessário implementar uma interface que exponha os *handles* para a IMU.

Os três métodos principais implementados pela classe do controlador são os métodos `init()`, `starting()` e `update()`. Esses métodos sobrescrevem métodos da classe `MultiInterfaceController`.

A função `init()` é chamada pelo gerenciador de controladores do ROS ao carregar o controlador. Nessa função é verificado se o robô tem uma `EffortJointInterface`, sem a qual o controlador não pode funcionar. Essa é a interface que permite o envio de comandos de torque para as juntas do robô. O nome das juntas nas quais o controlador atua é obtido através do servidor de parâmetros do ROS, no qual os parâmetros utilizados pelo controlador são carregados do arquivo de configuração do controlador pelo arquivo de `launch` desse. Para cada uma das juntas é verificado que o respectivo *handle* é exposto pela interface de *hardware* do robô. É nessa função, também, que o controlador se inscreve nos tópicos `/controller/command` e `/imu/data`. A árvore cinemática do robô sobre a plataforma é obtida pela função `kdl_parser::treeFromString()` a partir do parâmetro `robot_description` e armazenada numa variável do tipo `KDL::Tree`. O Apêndice A apresenta essa árvore completa. Através dos parâmetros `chain/tip` e `platform_chain/root`, obtém-se a cadeia cinemática referente apenas ao robô e a plataforma. O *solver* recursivo de Newton-Euler é criado para essa cadeia.

Além disso, são obtidos os parâmetros referêntes à ${}^{t}P_{imu}$, ${}^{enu}R_{p}$, ${}^{0}R_{enu}$, os quais são passados através de parâmetros *roll, pitch* e *yaw* e transformados em uma variável `KDL::Rotation`, que representa uma matriz de rotação. Os ganhos dos controladores são obtidos do servidor de parâmetros e armazenados em matrizes da biblioteca Eigen (GUENNEBAUD; JACOB et al., 2010), pois os elementos da KDL utilizam essas matrizes. Por fim, os vetores referentes a variáveis de juntas são redimensionados para o número de juntas do robô em conjunto com a plataforma.

A função `starting()` é chamada pelo gerenciador de controladores quando o con-

trolador é inicializado. As condições iniciais do robô e da plataforma são passadas para as variáveis de juntas. Também é feita a configuração da política do escalonador, (LAGES, 2016) apresenta os detalhes dessa configuração e também da configuração do sistema operacional para permitir execução em tempo real do controlador.

A implementação da lei de controle é realizada na função `udate()`, chamada pelo gerenciador de controladores a cada intervalo de amostragem. As funções `getVelocity()` e `getPosition()` são usadas para obter e armazenar as velocidades e posições das juntas do robô. Para calcular as posições, velocidades e aceleração de junta da plataforma as variáveis que armazenam a leitura do sensor IMU são atualizadas a cada chamada da função `imuCB()`, a função de *callback* do tópico da IMU. No ROS a IMU representa a orientação por um quaternion e aceleração e velocidade angular por vetores. Os dados da IMU são convertidos para ângulos, velocidades e acelerações de junta da plataforma com as equações (18), (19) e (23). A Equação (18) é computada a paritr da função `GetRPY()` da biblioteca KDL, que extrai os ângulos de *roll*, *pitch* e *yaw* de um objeto do tipo `Rotation` que armazena ${}^{p}R_t(q_p)$. O cálculo de ${}^{p}R_t(q_p)$ é realizado a partir da Equação (15). A Equação (23) não foi implementada no controlador, dessa forma foi utilizado $\ddot{q_p} = 0$. Isto implica no não cancelamento total do termo $F_p(q_p, \dot{q}_p, \ddot{q}_p)$ por parte da realimentação linearizante. Dessa forma entende-se que a performance do sistema para acelerações de junta da plataforma elevadas pode ser prejudicada. Para o cálculo da aceleração virtual foi implementado um PID, pois através desse pode-se obter um PD quando $K_i = 0$. Com as variáveis de junta da plataforma é possível resolver a Equação (13) pelo método recursivo de Newton-Euler. Os torques calculados são aplicados as juntas do robô com uma chamada da função `setCommand()`. Para a implementação completa do pacote `platform_computed_torque_controller` consultar Oliveira (2019b).

3.4 Descrição da Simulação

Para avaliar a performance do controlador sobre o efeito de movimentos na base o sistema foi testado no ambiente de simulação do Gazebo. A análise computacional considerou uma referência do tipo salto aplicada a cada uma das juntas do robô em um mesmo instante de tempo. Os seguintes controladores foram implementados no ROS e simulados no Gazebo:

CTCNC: Controlador de torque calculado sem compensação para o movimento da base, com lei de controle descrita pelas equações (5) e (10).

CTCPD: Controlador de torque calculado com compensação para o movimento da base, com lei de controle descrita pelas equações (13) e (10).

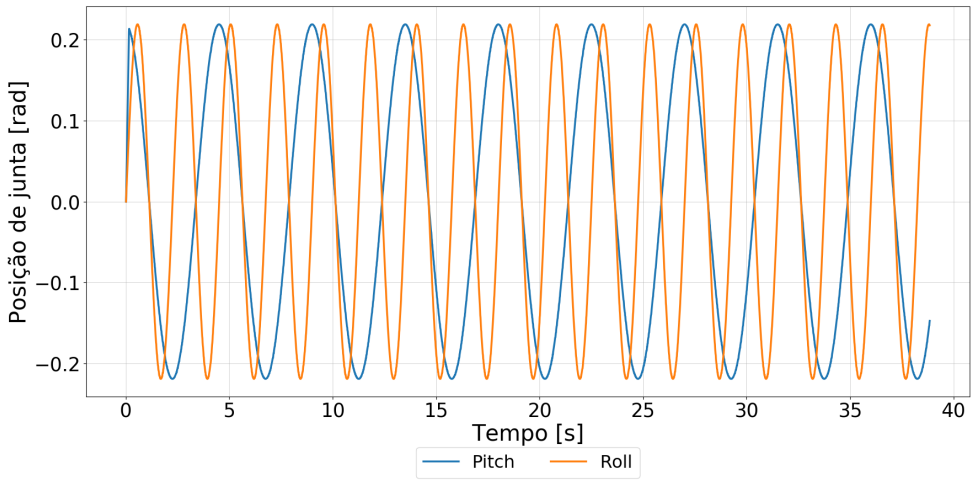
CTCPID: Controlador de torque calculado com compensação para o movimento da base e PID, com lei de controle descrita pela Equação (13) e $\nu = \ddot{q}_r + K_p(q_r - q) + K_i \int_p^t (q_r - q)dt + K_d(\dot{q}_r - \dot{q})$.

Para o movimento da plataforma, um sinal senoidal foi aplicado a cada uma de suas juntas, conforme (WRONKA; DUNNIGAN, 2011). Esse sinal é descrito por:

$$q_p(t) = \begin{bmatrix} a_{roll}\,\text{sen}\left(\frac{2\pi}{T_{roll}}t + \phi_{roll}\right) \\ a_{pitch}\,\text{sen}\left(\frac{2\pi}{T_{pitch}}t + \phi_{pitch}\right) \end{bmatrix} \tag{25}$$

com $a_{roll} = a_{pitch} = 0.219\,\text{rad}$, $T_{roll} = 4.5\,\text{s}$, $T_{pitch} = 2.25\,\text{s}$, $\phi_{roll} = \pi/2\,\text{rad}$ e $\phi_{pitch} = 0$. A Figura 7 apresenta o gráfico das posições geradas para as juntas da plataforma a partir da Equação (25).

Figura 7: Ângulos de *roll* e *pitch* para a plataforma.



Fonte: Elaborada pelo autor.

Os ganhos para o controlador PD foram calculados para obter uma resposta ao salto sem sobrepasso. Para o sistema descrito pela Equação (9), as matrizes diagonais de ganho proporcional e derivativo foram escolhidas com $k_p = \omega_n^2$ e $k_i = 2\xi\omega_n$. Com $\xi = 1$ e $\omega_n \approx \frac{4}{\xi T_s}$, onde T_s é o tempo de acomodação para um erro de 2%. Para o projeto do controlador PID, um sistema com equação característica de terceira ordem surge. Dessa forma o controlador foi projetado através do método de root locus. A Tabela 2 apresenta os ganhos obtidos.

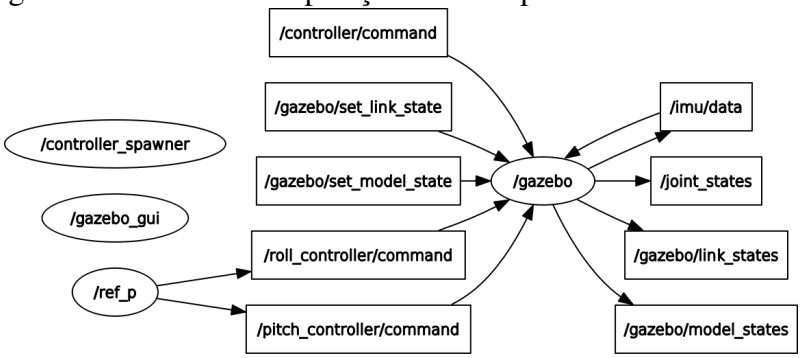
Tabela 2: Ganhos dos Controladores.

Ganho	CTCNC	CTCPD	CTCPID
K_p	25	25	30
K_d	10	10	10
K_i	0	0	1

A Figura 8 apresenta o grafo computacional dos nodos e tópicos do ROS utilizados para a simulação. O nodo `ref_p` publica a referência para a plataforma nos tópicos de referência de seus respectivos controladores de juntas `/roll_controller/command` e `/pitch_controller/command`. O *plugin* do controlador é executado em `/gazebo` e recebe referências enviadas ao tópico `/controller/command` geradas através de um *script* de *shell*. O tópico `/imu/data` é o tópico no qual o *plugin* que simula o sensor de IMU publica suas leituras e também o tópico que o controlador está inscrito para obter esses dados, por isso existem duas flechas indicando troca de informações com o tópico e o `/gazebo`.

Como medida de desempenho para os controladores utilizados, foi calculado o erro RMS (RMSE) da posição das juntas do robô com relação à referência. Para tanto pode-se usar:

Figura 8: Gráfico de computação do ROS para o sistema descrito.



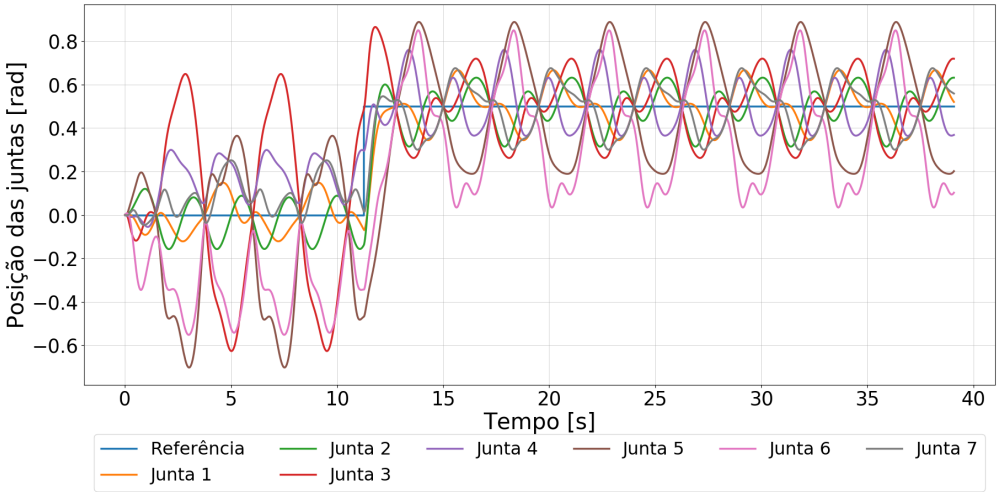
Fonte: Elaborada pelo autor.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(q_r[i] - q[i])^2}{N}} \tag{26}$$

4 RESULTADOS

Primeiro, o controlador CTCNC foi avaliado para medir o efeito da perturbação no sistema. Um degrau de referência de 0.5 rad foi aplicado a cada uma das 7 juntas do manipulador simultaneamente e a resposta do sistema é apresentada na Figura 9. É possível observar que o rastreamento de referência foi bastante prejudicado e as posições das juntas do robô oscilam devido ao movimento da plataforma. Para quantificar o desempenho do sistema o RMSE para posição de cada uma das juntas foi calculado e os resultados podem ser consultados na Tabela 3. O erro RMS total para esse teste foi de 1.2348 rad, indicando que o controlador não rejeita apropriadamente as perturbações geradas pelo movimento da base.

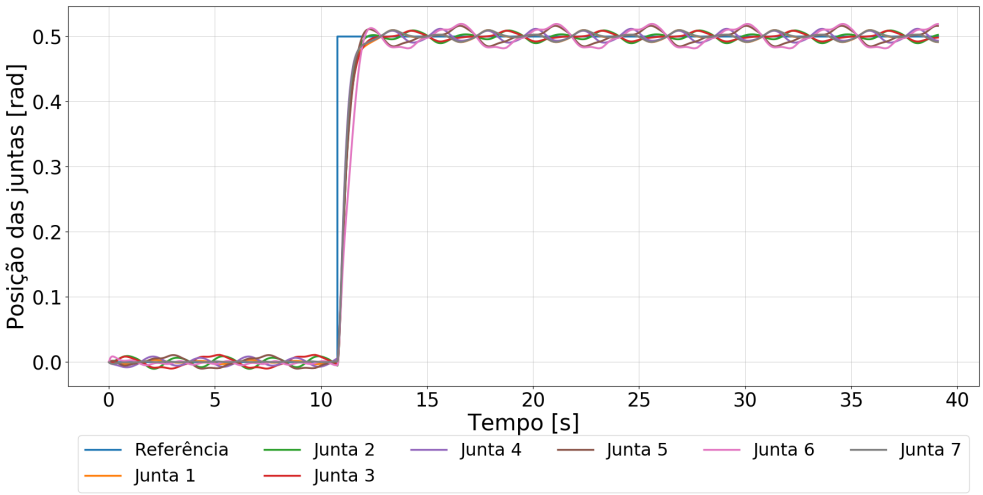
Figura 9: Resposta ao salto de CTCNC.



Fonte: Elaborada pelo autor.

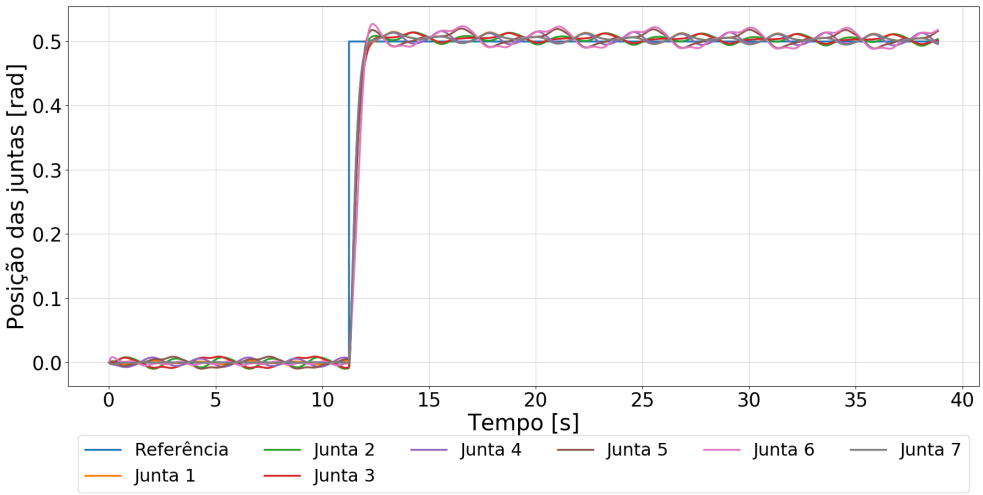
Em seguida, o controlador CTCPD foi avaliado para a mesma referência e os resultados são apresentados na Figura 10. Uma melhora considerável na performance do sistema pode ser observada e o erro RMS total do teste foi de 0.2927 rad. Apesar do melhor desempenho notou-se um certo *offset* da resposta. Para remover esse *offset* foi utilizado um controlador PID para o sistema linearizado, esse é o controlador CTPID, cuja resposta ao salto pode ser visualizada na Figura 11. Com o emprego deste controlador obteve-se mais uma melhora no desempenho, reduzindo o erro RMS de posição de junta para 0.2735 rad. Apesar desse controlador reduzir o erro em regime permanente, sua resposta transitória é levemente deteriorada devido a um pequeno sobrepasso que se acomoda lentamente.

Figura 10: Resposta ao sato de CTCPD.



Fonte: Elaborada pelo autor.

Figura 11: Resposta ao salto de CTCPID.

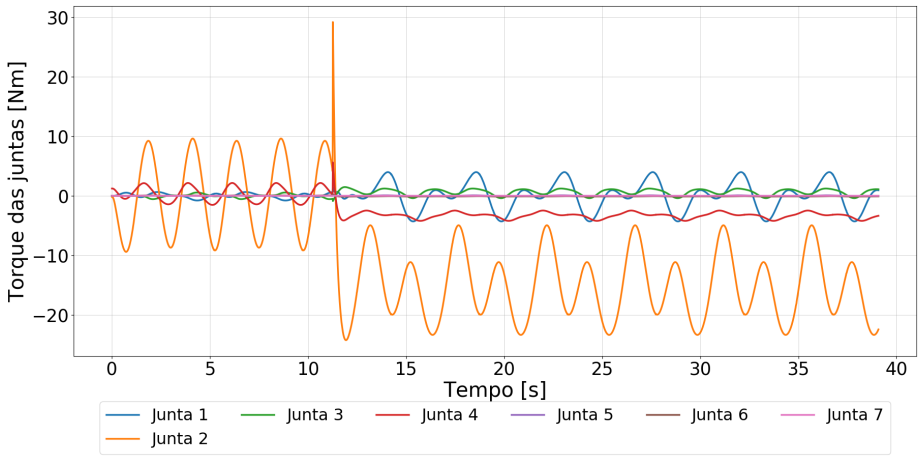


Fonte: Elaborada pelo autor.

As figuras 12, 13 e 14 apresentam os torques aplicados nas juntas do robô para CTCNC, CTCPD e CTCPID, respectivamente. Pode ser observado que os controladores com compensação do movimento da base aplicam torques menores às juntas. A exceção é para o CTCPID em mudanças de referência, quando esse produz um valor de torque maior que de todos os outros controladores, devido ao seu ganho proporcional maior.

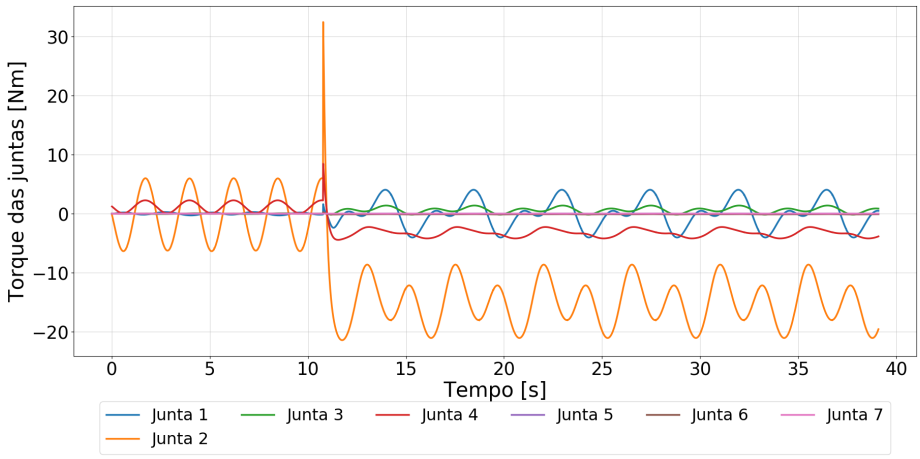
A avaliação dos controladores foi realizada através do RMSE, calculado pela Equação (26). A Tabela 3 apresenta esse erro para cada uma das juntas do manipulador. A partir da soma dos erros pode-se constatar uma grande melhora dos controladores CTCPD e CTCPID com relação ao CTCNC. Além disso o controlador CTCPID obteve os melhores resultados para esse experimento.

Figura 12: Torques aplicados nas juntas com CTCNC.



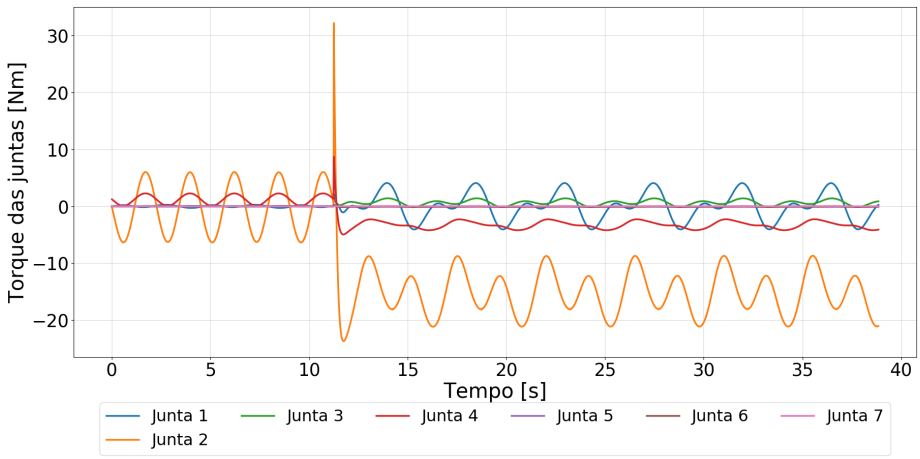
Fonte: Elaborada pelo autor.

Figura 13: Torques aplicados nas juntas com CTCPD.



Fonte: Elaborada pelo autor.

Figura 14: Torques aplicados nas juntas com CTCPID.



Fonte: Elaborada pelo autor.

Tabela 3: Erro RMS para posição de junta [rad].

Junta	CTCNC	CTCPD	CTCPID
1	0,1000	0,0404	0,0380
2	0,1038	0,0420	0,0389
3	0,2498	0,0418	0,0394
4	0,1466	0,0391	0,0368
5	0,3046	0,0418	0,0391
6	0,3088	0,0477	0,0443
7	0,1204	0,0398	0,0369
Total	1,3339	0,2927	0,2735

5 CONCLUSÕES

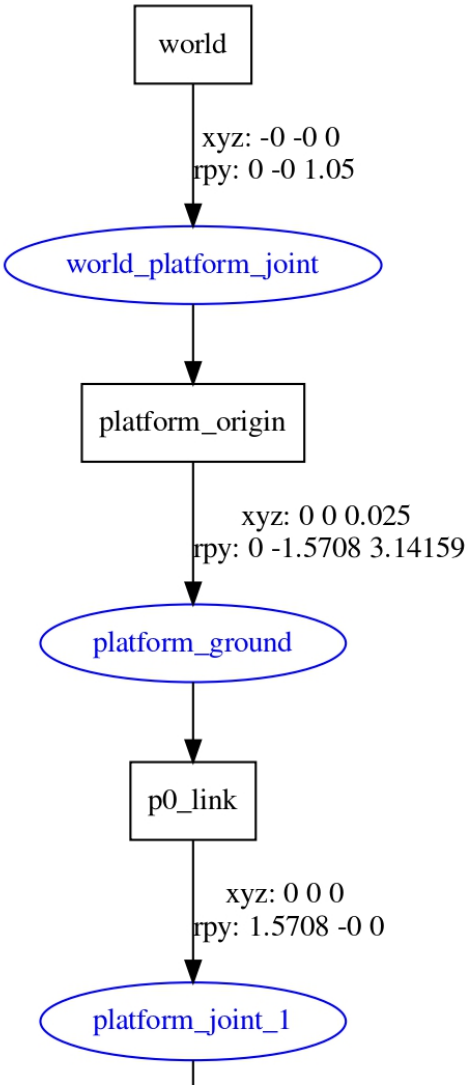
Neste trabalho foi apresentada a implementação de um controlador para um manipulador robótico sobre uma base não inercial. Um modelo simplificado de uma plataforma com dois graus de liberdade foi desenvolvido no formato URDF. Assumiu-se movimentos de *roll* e *pitch*. Uma lei de controle MIMO, não linear foi implementada no ROS através do *framework* `ros_control`. Uma simulação com um controlador sem compensação do movimento da plataforma foi realizada e esse apresentou um desempenho fraco, não conseguindo fazer as posições de junta convergir para a referência e servindo de motivação para a implementação de uma solução mais específica para o problema de controle.

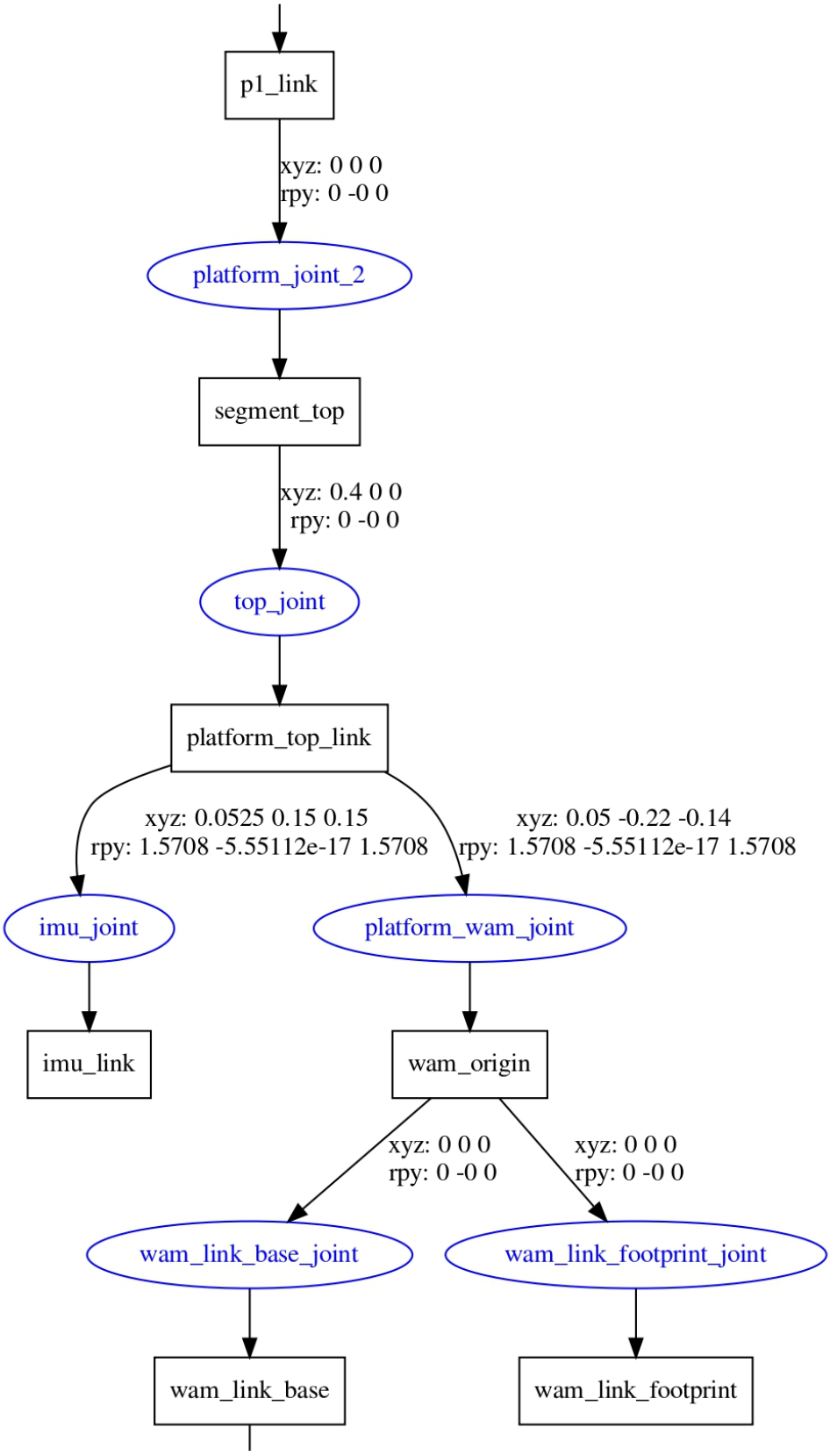
Para compensar o movimento da base, dois controladores foram propostos. Ambos utilizaram a lei de controle por torque calculado, um método de controle baseado em modelo, dessa forma assume-se que tanto modelo, quanto parâmetros do sistema são conhecidos. A linearização da dinâmica do manipulador também assume o conhecimento do movimento da plataforma, dessa forma uma IMU foi utilizada e o equacionamento para transformar suas leituras em variáveis de junta da plataforma foi apresentado.

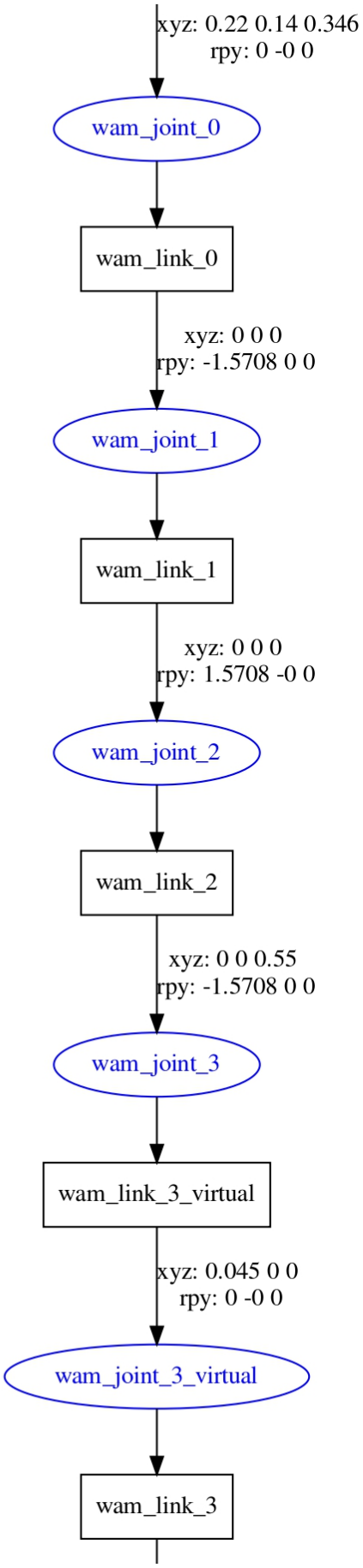
O CTCPID apresentou erro RMS total 0.0192 rad menor que o CTCPD para o seguimento de referência de posição de junta. Ambos os controladores apresentaram torques comparáveis, com a exceção de que o CTCPID produz torque maior quando uma nova referência é aplicada. Em comparação com CTCNC, ambos os controladores apresentaram uma melhora significativa no desempenho do sistema. Além disso, o CTCNC demandou de maiores torques para tentar compensar o movimento da plataforma, quando comparado aos controladores que cancelam a maior parte das não linearidades.

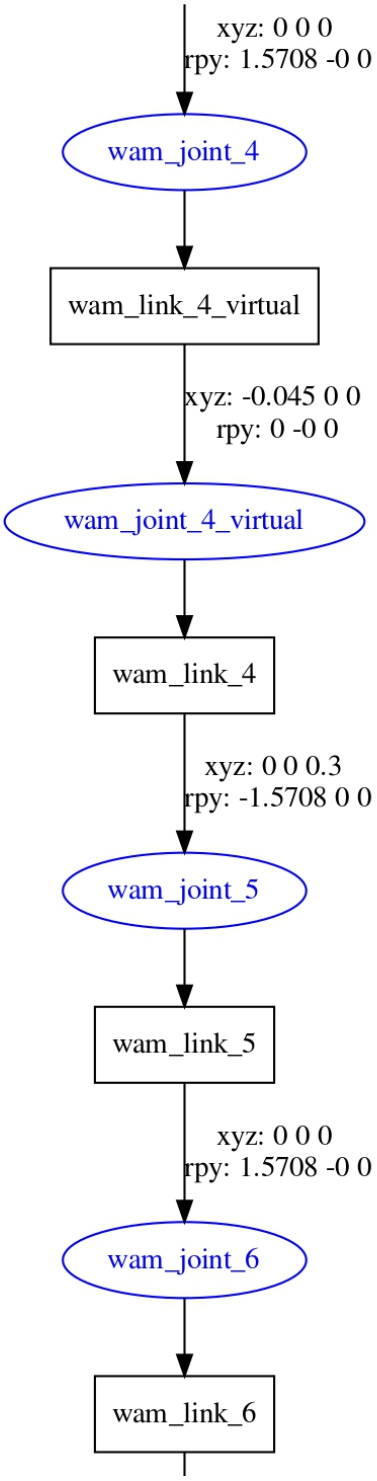
Em trabalhos futuros será realizada a integração dos controladores implementados neste trabalho com o robô Barrett WAM real sobre uma plataforma de Stewart. Devido ao uso do ROS e do *framework* do `ros_control` essa integração pode ser simplificada. A princípio pode-se utilizar o mesmo código do controlador executado com a simulação, no *hardware* real do robô. Para isto deve-se alterar a implementação da classe `hardware_interface::RobotHW` utilizada com o Gazebo, para a interface com o *hardware* do robô. Essa classe deve garantir uma interface de *hardware* para a IMU. Uma pequena adequação no controlador deve ser feita para esse utilizar os dados da IMU provindos do *handle* dessa. Essa alteração remove a função de *callback* para o tópico `/imu/data`, permitindo a execução do controlador em tempo real. Também se está trabalhando na implementação da Equação (23), para garantir que a lei de controle realize um cancelamento completo das não linearidades.

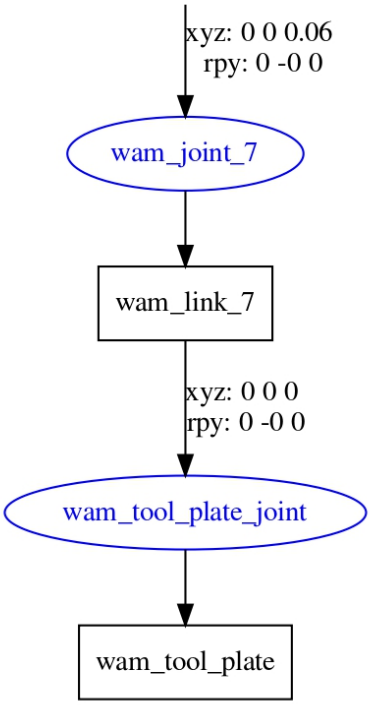
APÊNDICE A ÁRVORE CINEMÁTICA DO SISTEMA



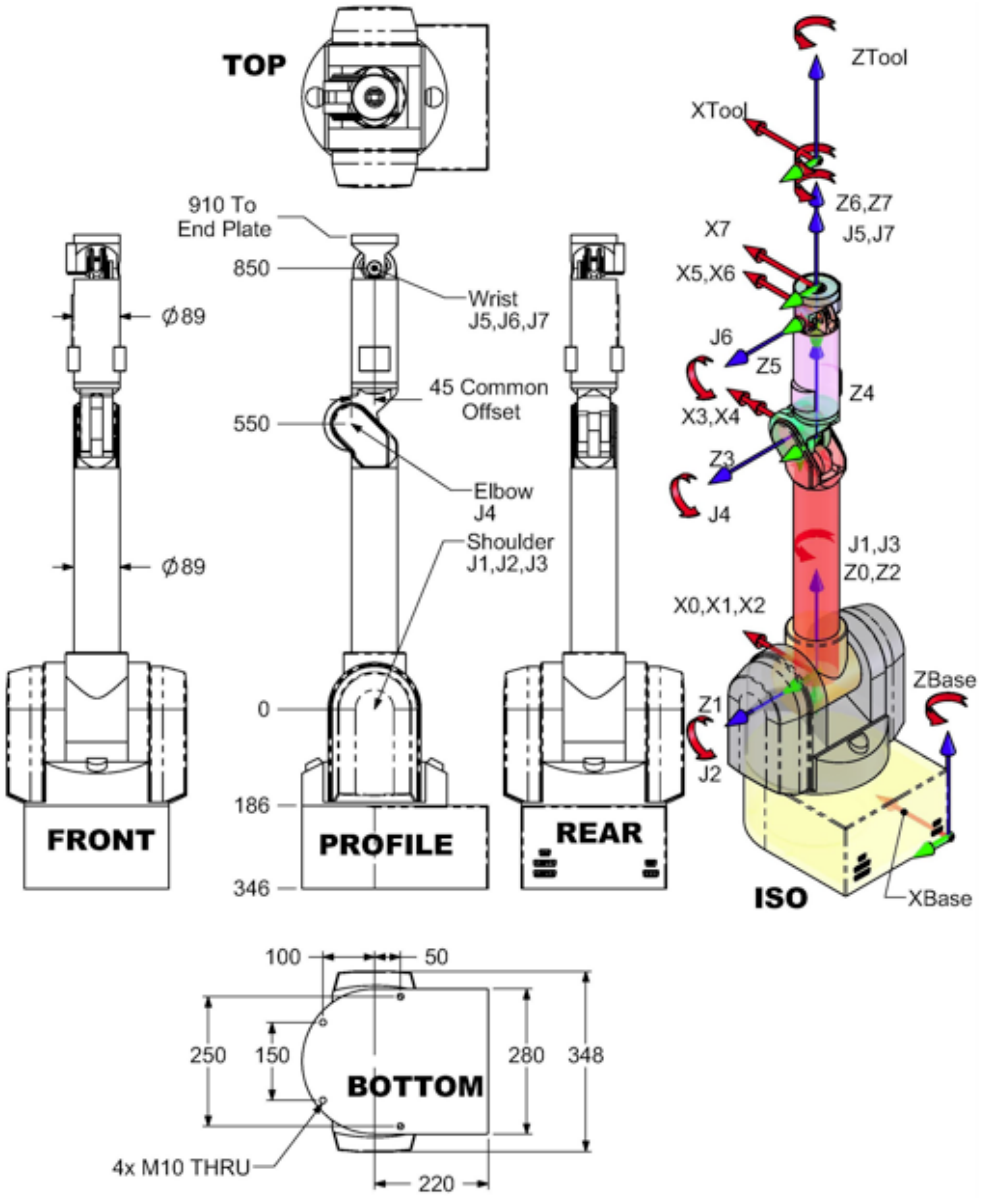

p1_link
xyz: 0 0 0
rpy: 0 -0 0
platform_joint_2
segment_top
xyz: 0.4 0 0
rpy: 0 -0 0
top_joint
platform_top_link
xyz: 0.0525 0.15 0.15
rpy: 1.5708 -5.55112e-17 1.5708
xyz: 0.05 -0.22 -0.14
rpy: 1.5708 -5.55112e-17 1.5708
imu_joint
platform_wam_joint
imu_link
wam_origin
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
wam_link_base_joint
wam_link_footprint_joint
wam_link_base
wam_link_footprint


xyz: 0.22 0.14 0.346
rpy: 0 -0 0
wam_joint_0
wam_link_0
xyz: 0 0 0
rpy: -1.5708 0 0
wam_joint_1
wam_link_1
xyz: 0 0 0
rpy: 1.5708 -0 0
wam_joint_2
wam_link_2
xyz: 0 0 0.55
rpy: -1.5708 0 0
wam_joint_3
wam_link_3_virtual
xyz: 0.045 0 0
rpy: 0 -0 0
wam_joint_3_virtual
wam_link_3


xyz: 0 0 0
rpy: 1.5708 -0 0
wam_joint_4
wam_link_4_virtual
xyz: -0.045 0 0
rpy: 0 -0 0
wam_joint_4_virtual
wam_link_4
xyz: 0 0 0.3
rpy: -1.5708 0 0
wam_joint_5
wam_link_5
xyz: 0 0 0
rpy: 1.5708 -0 0
wam_joint_6
wam_link_6


xyz: 0 0 0.06
rpy: 0 -0 0
wam_joint_7
wam_link_7
xyz: 0 0 0
rpy: 0 -0 0
wam_tool_plate_joint
wam_tool_plate

ANEXO A DIMENSÕES BARRETT WAM 7 GDL



Fonte: (BARRETT, 2011)

REFERÊNCIAS

BARRETT. *WAM User Manual*. Cambridge, MA, 2011.

BARROS, T. T. T.; LAGES, W. F. A Mobile Manipulator Controller Implemented in the Robot Operating System. In: PROCEEDINGS for the Joint Conference of 45th International Symposium on Robotics and 8th German Conference on Robotics. Munich, Germany: VDE Verlag, 2014. p. 121–128. ISBN 978-3-8007-3601-0.

BRUYNINCKX, H. Open Robot Control Software: The OROCOS Project. In: PROCEEDINGS of the 2001 IEEE International Conference on Robotics and Automation. Seoul, South Korea: IEEE Press, 2001. p. 2523–2528.

CHITTA, S. et al. ros_control: A generic and simple control framework for ROS. *The Journal of Open Source Software*, 2017a. DOI: 10.21105/joss.00456. Disponível em: <http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf>.

CHITTA, S. et al. ros_control: A generic and simple control framework for ROS. *The Journal of Open Source Software*, 2017b. DOI: 10.21105/joss.00456. Disponível em: <http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf>.

DUNNIGAN, M. W.; WRONKA, C. M. Comparison of control techniques for a robotic manipulator with base disturbances. *IET Control Theory Applications*, v. 5, n. 8, p. 999–1012, mai. 2011. ISSN 1751-8652. DOI: 10.1049/iet-cta.2010.0331.

FOOTE, T.; PURVIS, M. *Standard Units of Measure and Coordinate Conventions*. 2010. Disponível em: <https://www.ros.org/reps/rep-0103.html>.

FU, K. S.; GONZALES, R. C.; LEE, C. S. G. *Robotics Control, Sensing, Vision and Intelligence*. New York: McGraw-Hill, 1987. (Industrial Engineering Series). ISBN 0-07-022625-3.

GOODWIN, G. C.; SIN, K. S. *Adaptive Filtering, Prediction and Control*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984. (Prentice-Hall Information and System Sciences Series).

GUENNEBAUD, G.; JACOB, B. et al. *Eigen v3*. 2010.

KOENIG, N.; HOWARD, A. Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In: PROCEEDINGS of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004). Sendai, Japan: IEEE Press, set. 2004. v. 3, p. 2149–2154. DOI: 10.1109/IROS.2004.1389727.

KOK, M.; HOL, J. D.; SCHÖN, T. B. Using Inertial Sensors for Position and Orientation Estimation. *Foundations and Trends in Signal Processing*, v. 11, n. 1-2, p. 1–153, 2017. DOI: 10.1561/2000000094.

LAGES, W. F. Implementation of Real-Time Joint Controllers. In: KOUBAA, A. (Ed.). *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Switzerland: Springer International Publishing, 2016. v. 625. (Studies in Computational Intelligence). p. 671–702. ISBN 978-3-319-26052-5.

LAGES, W. F.; HENRIQUES, R. V. B. Real-Time Point Stabilization of a Mobile Robot using Model Predictive Control. In: IEEE. PROCEEDINGS of the 16th Latin American Robotics Symposium / 7th Brasilian Robotics Symposium. Rio Grande, RS, Brazil: [s.n.], out. 2019.

MACIEL, E. H. Desenvolvimento de um modelo simplificado dos membros inferiores de um robô bıpede utilizando ROS, 2014.

OLIVEIRA, C. *oliverzila/platform: Platform Package TCC*. Versão v1.0.0. Nov. 2019a. Disponível em: <https://doi.org/10.5281/zenodo.3556991>.

OLIVEIRA, C. *oliverzila/platform_computed_torque_controller: Platform Computed Torque Controller TCC*. Versão v1.0.0. Nov. 2019b. Disponível em: <https://doi.org/10.5281/zenodo.3556993>.

OLIVEIRA, C. E. P. DE; LAGES, W. F.; HENRIQUES, R. V. B. Control of a Manipulator Mounted on anIndependent Controlled Moving Platform. In: IFAC. IFAC Proceedings Volumes. Berlin, Germany: Elsevier, 2019. To appear.

QUIGLEY, M. et al. ROS: an Open-Source Robot Operating System. In: PROCEEDINGS of the IEEE International Conference on Robotics and Automation, Workshop on Open Source Robotics. Kobe, Japan: IEEE Press, mai. 2009.

ROS.ORG. *Documentation - ROS Wiki*. 2019a. Disponível em: <http://wiki.ros.org>.

ROS.ORG. *Messages - ROS Wiki*. 2019b. Disponível em: <http://wiki.ros.org/Messages>.

ROS.ORG. *Nodes - ROS Wiki*. 2019c. Disponível em: <http://wiki.ros.org/Nodes>.

ROS.ORG. *ROS*. 2010. Disponível em: <http://www.ros.org>.

ROS.ORG. *Topics - ROS Wiki*. 2019d. Disponível em: <http://wiki.ros.org/Topics>.

SADRAEI, E.; MOGHADDAM, M. On a Moving Base Robotic Manipulator Dynamics. *International Journal of Robotics, Theory and Applications*, K.N. Toosi University of Technology, v. 4, n. 3, p. 66–74, 2015. ISSN 2008-7144. Disponível em: <http://ijr.kntu.ac.ir/article_13764.html>.

SICILIANO, B.; KHATIB, O. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN 354023957X.

SLOTINE, J.-J. E.; LI, W. *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice–Hall, 1991.

WHITNEY, D. E. The Mathematics of Coordinated Control of Prosthetic Arms and Manipulator. *Trans. ASME J. Dynamic Systems, Measurement and Control*, ASME, v. 94, n. 4, p. 303–309, dez. 1972. ISSN 0022-0434. DOI: 10.1115/1.3426611. Disponível em: <https://doi.org/10.1115/1.3426611>.

WIT, C. C. DE; BASTIN, G.; SICILIANO, B. (Ed.). *Theory of Robot Control*. 1st. Berlin, Heidelberg: Springer-Verlag, 1996. ISBN 3540760547.

WRONKA, C. M. *Modelling and Control of a Robotic Manipulator Subject to Base Disturbances*. Mai. 2010. Ph.D. Thesis (Electrical, Electronic and Computer Engineering) – Heriot-Watt University, Edinburgh.

WRONKA, C. M.; DUNNIGAN, M. W. Derivation and analysis of a dynamic model of a robotic manipulator on a moving base. *Robotics and Autonomous Systems*, v. 59, n. 10, p. 758–769, 2011. ISSN 0921-8890. DOI: https://doi.org/10.1016/j.robot.2011.05.010.