Instituto Federal do Rio Grande do Norte
Campus Natal - Central
Diretoria de Gestão e Tecnologia da Informação
Tecnologia em Análise e Desenvolvimento de Sistemas

Título do trabalho

Jamillo G. da S. Santos

Natal-RN
Abril 2017

Jamillo G. da S. Santos

Título

Trabalho de conclusão de curso de graduação do curso de Tecnologia e Análise em Desenvolvimento de Sistemas da Diretoria de Gestão e Tecnologia de Informação do Instituto Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Tecnologo em Análise e Desenvolvimento de Sistemas.

Linha de pesquisa:
Nome da linha de pesquisa

Orientador

Prof. Dr. Eduardo Braulio

TADS – Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
DIATINF – Diretoria Acadêmica de gestão e Tecnologia da Informação
CNAT – Campus Natal - Central
IFRN – Instituto Federal do Rio Grande do Norte

Natal-RN

Abril 2017

Trabalho de Conclusão de Curso de Graduação sob o título *Título* apresentada por Nome completo do autor e aceita pelo Diretoria de Gestão e Tecnologia da Informação do Instituto Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Nome completo do orientador e titulação
Presidente
DIATINF – Diretoria Acadêmica de Gestão e Tecnologia da Informação
IFRN – Instituto Federal do Rio Grande do Norte

Nome completo do examinador e titulação
Examinador
Diretoria/Departamento
Instituto

Nome completo do examinador e titulação
Examinador
Diretoria/Departamento
Universidade

Natal-RN, data da defesa (dia, mês e ano).

Homenagem que o autor presta a uma ou mais pessoas.

Agradecimentos

Agradecimentos dirigidos àqueles que contribuíram de maneira relevante à elaboração do trabalho, sejam eles pessoas ou mesmo organizações.

Citação

Autor

Título do trabalho

Autor: Jamillo G. da S. Santos
Orientador(a): Prof. Dr. Eduardo Braulio

Resumo

O resumo deve apresentar de forma concisa os pontos relevantes de um texto, fornecendo uma visão rápida e clara do conteúdo e das conclusões do trabalho. O texto, redigido na forma impessoal do verbo, é constituído de uma sequência de frases concisas e objetivas e não de uma simples enumeração de tópicos, não ultrapassando 500 palavras, seguido, logo abaixo, das palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores. Por fim, deve-se evitar, na redação do resumo, o uso de parágrafos (em geral resumos são escritos em parágrafo único), bem como de fórmulas, diagramas e símbolos, optando-se, quando necessário, pela transcrição na forma extensa, além de não incluir citações bibliográficas.

Palavras-chave: Palavra-chave 1, Palavra-chave 2, Palavra-chave 3.

Título do trabalho (em língua estrangeira)

Author: Jamillo G. da S. Santos
Supervisor: Prof. Doc. Eduardo Braulio

Abstract

O resumo em língua estrangeira (em inglês *Abstract*, em espanhol *Resumen*, em francês *Résumé*) é uma versão do resumo escrito na língua vernícula para idioma de divulgação internacional. Ele deve apresentar as mesmas características do anterior (incluindo as mesmas palavras, isto é, seu conteúdo não deve diferir do resumo anterior), bem como ser seguido das palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores, na língua estrangeira. Embora a especificação abaixo considere o inglês como língua estrangeira (o mais comum), não fica impedido a adoção de outras linguas (a exemplo de espanhol ou francês) para redação do resumo em língua estrangeira.

Keywords: Keyword 1, Keyword 2, Keyword 3.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

ARASH – *Anthropomorphic Robot Augmented with Sense of Human*, em livre tradução Robô Antropomórfico Aumentado com Sentido de Ser Humano

DOF – Do inglês *Degrees Of Freedom*, ou graus de liberdade

IK – *Inverse Kinematics*, ou cinemática inversa

Sumário

1 Introdução

GUIA:

- Contexto
- Motivação
- O problema
- Soluções similares
- Solução proposta

DRAFT: Nos dias atuais, a robótica está cada vez mais avançada e acessível. Seja em um pequeno robô aspirador de pó autônomo - que até mesmo retorna à estação de recarregamento quando sua bateria está fraca - ou nos sofisticados robôs que automatizam a fabricação de carros, aumentando a produção e diminuindo o custo. E eles vem em todos os tamanhos e formas.

Vivemos em um mundo feito e projetado para humanos. De escadas, portas, e até ferramentas como furadeiras, ou até o mouse são pensados para o uso diário de uma forma humanoide. Desta forma, nada mais obvio que projetar um robô humanoide que adapte-se de forma natural a este ambiente. Porém, a forma humanoide impõe diversos desafios de controle, basta uma breve busca no YouTube por "darpa challange fails".

Normaliz
todas as nomenclaturas do *walking gait*, as vezes chama-se sistema,

GUIA:

Falar das dificuldades na robótica humanoide e fazer uma contextualização da caminhada citando alguns trabalhos recentes.

Em um esforço mútuo, as universidades *Amirkabir University of Technology* (AUT), do Irã, e a *University of Manitoba* (UofM), do Canadá, trabalham juntas para participar da Robocup - competição internacional de futebol de robôs cujo o objetivo é derrotar a seleção campeã mundial na copa de 2050. Nas edições 2015 e 2016, realizada em Hefei (China) e Leipzig (Alemanhã), o sistema que controlou a caminhada de *Arash*; um robô humanóide de 100cm de altura rendeu o 3º lugar na modalidade de futebol em ambos os anos. Também, os 2º e 1º lugar no modalidade do desafio técnico em 2015 e 2016.

Desenvolvido utilizando placa microcontroladora *OpenCM 9.04*, compatível com Arduino, e batizado de *AUT-UofM-Walk-Engine*, o sistema de caminhada foi um dos grandes responsáveis pelo bom desempenho. Entretando, o fato de rodar dentro da *OpenCM9.04* traz uma série de consequências não desejáveis ao palco.

Inicialmente, sendo um sistema complexo, a caminhada deve ser configurada para cada tipo de superfície. As configurações de uma superfície lisa e áspera, como concreto, podem ser bastante diferentes das configurações para caminhar num terreno como a grama artificial utilizada na Robocup. Essas mudanças nas configurações são determinantes para o bom equilíbrio do robô. Com mais de 100 parâmetros, esse processo de configuração requer muito tempo e paciência.

O primeiro problema com a implementação atual é que a configuração se dá através de alterações diretas dos valores dos parâmetros no código-fonte, em seguida compila-se, envia-se a nova versão à *OpenCM9*. Em seguida, inicializa-se o robô e verifica-se a caminhada. Além de massante, este esquema de atualização não favorece uma forma sadia de manter as alterações dos parâmetros organizada. Há uma frequente perda de quais valores de parâmetros são melhores para qual tipo de situação. Uma solução seria criar subversões do sistema para cada cenário, o que desfavorece a correção de *BUGs* e melhorias, já que cada modificação deve ser refletida em várias versões. Uma segunda possibilidade seria criar uma interface de comunicação entre um computador, que funcionaria como um gerenciador, e o *walking gait* via USB. Assim, o computador poderia guardar arquivos de configurações que seriam enviados à placa microcontroladora a medida que fossem necessários, o que seria uma saída simples e eficaz, mas não definitiva, tendo em vista outros problema citados abaixo.

Introduz a expressão ***walking gait***.

O segundo problema com o *walking gait* atual é a dificuldade de implementação de simulações, testes e depuração. A única forma de obter visualização da saída gerada é através do *console serial* da IDE de programação, o que torna o processo de depuração um exercício de abstração com visualização de ângulos em 3 dimensões, ou arriscar-se

aplicando a saída direto aos motores, de alto custo. Em relação a simulação, ela seria até possível, porém seria necessária mais um dispositivo *OpenCM9.04* e algumas modificações no código original para que ele possa ser acoplado à algum framework de simulação de robôs.

O terceiro problema é a forma em que o paradigma estruturado foi utilizada no sistema. A complexidade do sistema implica para uma baixa curva de assimilação do funcionamento em um paradigma orientado a objetos ajudaria consideravelmente na compreensão dos mecanismos dentro do *walking gait*, quais os seus papéis e como eles se comunicam.

O quarto problema é a natureza multi-processada do sistema. "Ao mesmo tempo"em que ângulos devem ser enviados aos motores, a leitura dos sensores devem ser processadas e levadas em consideração para as próximas iterações. No *OpenCM0.04*, esse paralelismo é habilitado através da biblioteca *MapleFreeRTOS*. O fato de já haver muito processamento para rodar o sistema atual limita uma grupo de melhorias a serem desenvolvidas, o que poderia levar a próxima geração de robôs ainda mais estáveis.

Por fim, o quinto problema, que também limita o desenvolvimento da próxima geração, é o tamanho da memória ROM, que guarda o *walking gait*, da placa microcontroladora. Atualmente, o *firmware* da *OpenCM9.04* já foi modificado, via remoção de funções não utilizadas, para diminuir seu tamanho e assim liberar espaço para a versão do sistema.

Dadas, os problemas encontrados no funcionamento do sistema atual, e as considerações listadas até agora, este trabalho propõe mover a execução do componente *walking gait* do microcontrolador *OpenCM09.04* para o computador. Para o projeto da solução, foi lavada em consideração dois aspectos importantes: o desempenho e a interface de comunicação com os motores, decidindo-se assim pela adoção da linguagem *C++* versão 14. Um fator bastante importante foram os recursos da linguagem que vão prover uma arquitetura bem simples e eficiente.

Desta forma, este trabalho tem como objetivo a implementação de uma versão funcional do *walking gait*, melhorando o código atual - de forma a diminuir a curva de aprendizado, assim estimulando novas extensões e otimizações - com uma nova arquitetura orientada a objetos, ativando o sistema com suporte ao ROS, adicionando uma interface web responsiva de configuração.

Conversa com o orientador sobre a lista

1.1 Metodologia

Na metodologia é descrito o método de investigação e pesquisa para o desenvolvimento e implementação do trabalho que está sendo proposto.

Converse com o orientador sobre isto.

1.2 Organização do trabalho

Nesta seção deve ser apresentado como está organizado o trabalho, sendo descrito, portanto, do que trata cada capítulo.

2 Capítulo 2

Definir título

GUIA:

Falar sobre as tecnologias atuais de *walking gait*.

GUIA:

Falar sobre a abordagem tomada para este trabalho.

Este é o primeiro capítulo da parte central do trabalho, isto é, o desenvolvimento, a parte mais extensa de todo o trabalho. Geralmente o desenvolvimento é dividido em capítulos, cada um com subseções e subseções, cujo tamanho e número de divisões variam em função da natureza do conteúdo do trabalho.

Em geral, a parte de desenvolvimento é subdividida em quatro subpartes:

- *contextualização ou definição do problema* – consiste em descrever a situação ou o contexto geral referente ao assunto em questão, devem constar informações atualizadas visando a proporcionar maior consistência ao trabalho;
- *referencial ou embasamento teórico* – texto no qual se deve apresentar os aspectos teóricos, isto é, os conceitos utilizados e a definição dos mesmos; nesta parte faz-se a revisão de literatura sobre o assunto, resumindo-se os resultados de estudos feitos por outros autores, cujas obras citadas e consultadas devem constar nas referências;
- *metodologia do trabalho ou procedimentos metodológicos* – deve constar o instrumental, os métodos e as técnicas aplicados para a elaboração do trabalho;
- *resultados* – devem ser apresentados, de forma objetiva, precisa e clara, tanto os resultados positivos quanto os negativos que foram obtidos com o desenvolvimento

do trabalho, sendo feita uma discussão que consiste na avaliação circunstanciada, na qual se estabelecem relações, deduções e generalizações.

É recomendável que o número total de páginas referente à parte de desenvolvimento não ultrapasse 60 (sessenta) páginas.

2.1 Seção 1

Teste de figura:

Figura 1: Teste de uma figura em formato .png

2.2 Seção 2

Referenciamento da figura inserida na seção anterior: **??**

2.3 Seção 3

Seção 3

2.3.1 Subseção dentro da seção 3

2.3.2 Subseção dentro da seção 3

2.4 Seção 4

Seção 4

3 Arquitetura

ARASH (*Anthropomorphic Robot Augmented with Sense of Human*, em português Robô Antropomórfico Aumentado com Sentido de Ser Humano), além de um herói arqueiro da mitologia iraniana, é um robô de 7,5 Kg e 1 metro de altura. Possuindo 20 graus de liberdade (ou DOF , do inglês *degrees of freedom*), ele tenta imitar a configuração humana, tanto em suas juntas quanto em suas proporções. Em suas juntas, Arash possui motores MX-28, MX-64 e MX-106 (da Robotis Co), dependendo da carga suportada. Como controlador principal, um computador *mini-box* é utilizado em conjunto com uma placa microcontroladora *OpenCM9.04* que serve de interface entre o controlador principal e os motores.

3.1 Visão Geral

Com o passar dos anos, a arquitetura distribuída é vem sendo cada vez mais adotada na robótica. A ideia é que cada componente funcione de forma independente para evitar que uma eventual falha derrube todo o sistema. Componentes individuais podem ser adicionados, ou substituídos, a medida em que melhorias, ou novas capacidades, sejam implementadas.

Na figura 2 é possível observar os principais componentes – e o sentido do fluxo de dados entre eles – envolvidos no processo qualquer tarefa básica. Esses componentes são executados em diferentes camadas do sistema. Na arquitetura, existem 3 camadas que podem ser descritas independentemente: A camada de software, a de software de baixo nível, e a de hardware.

Na camada de *hardware*, a mais inferior na figura 2, estão a câmera, a IMU e os motores utilizados nas juntas. Os motores serão melhor descritos na subseção sobre atuadores (subseção **??**). A IMU, *inertial measurement unit*, que é um sensor eletrônico capaz de medir as forças sofridas por um corpo, é utilizada para ler a orientação do corpo do robô.

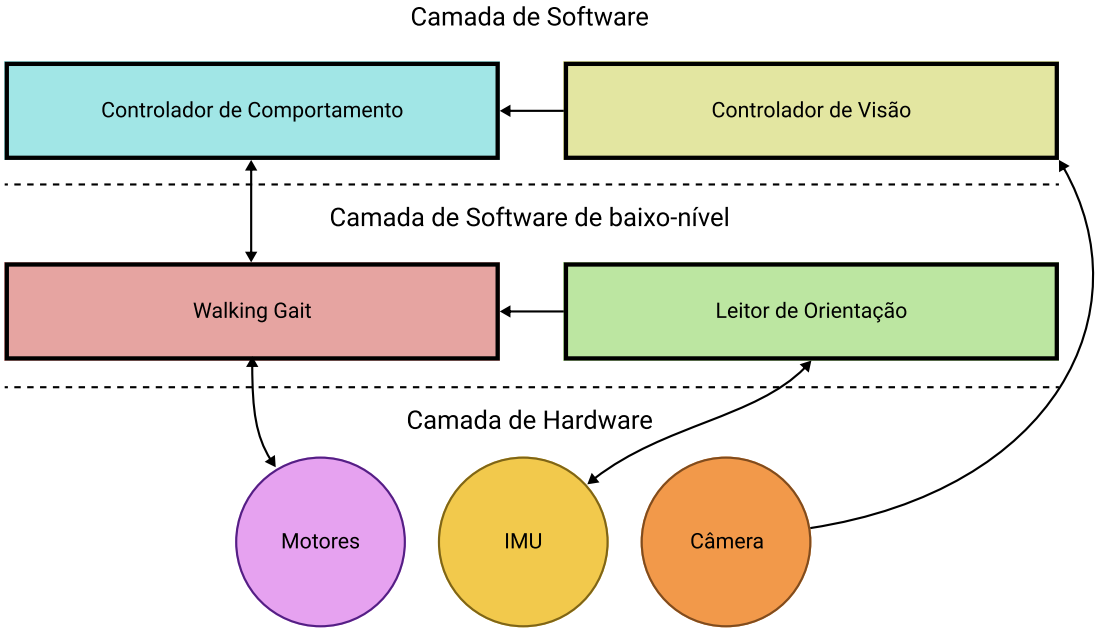

Figura 2: Visão simplificada da arquitetura do sistema e seus fluxos.

A camada de software de baixo nível é responsável pelo processamento de dados da *IMU*, juntamente com a execução do *walking gait*. Esta camada roda dentro da placa microcontroladora *Robotis.Co OpenCM9.04* (com um processador *32bit ARM Cortex-M3*, memória *flash* de 128Kb e 20Kb de *SRAM*), *arduino-like*.

A camada de software é executada dentro do "controlador principal", que é um computador *mini-box PC MAXData QutePC-3001* com um processador 64 bits de dois núcleos Intel© Celeron© 847E (1,1GHz e 2MB de *cache*), rodando o Ubuntu 14.04. Esta camada é responsável por rodar os componentes de software e enviar comandos de controle para a camada inferior de baixo nível.

3.1.1 Visão

Visão não é o foco deste trabalho. Entretanto, uma explanação geral é necessária afim de melhor entender o funcionamento geral do sistema.

O controle de visão é um software que conecta-se via USB com a câmera, analisando as imagens e extraindo informações relevantes à execução da tarefa do robô. De forma geral, podemos abstrair seu funcionamento em duas *threads* distintas:

A primeira é o *loop* principal – representada na figura 3 –, onde, sequencialmente, o controlador obtém a imagem da câmera. Em seguida, a extração de informações é realizada. Esse processo depende da tarefa que o robô deve realizar; tais como detectar a

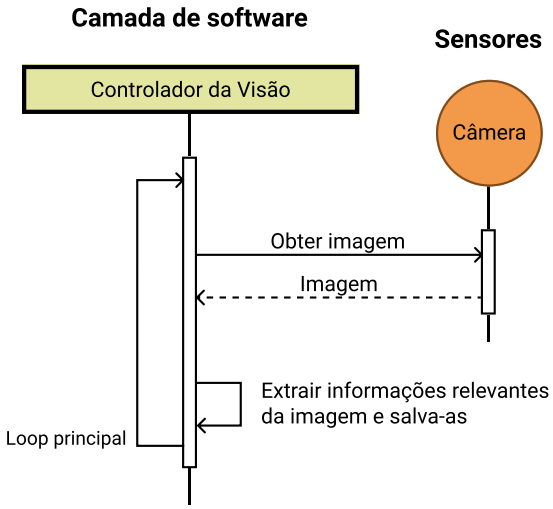

Figura 3: Diagrama de sequência do componente de visão.

bola, os adversários e as linhas do campo, em caso de futebol ou detectar as linhas guia da pista de corrida no caso da maratona. Em seguida, tudo o que foi encontrado é salvo para futuras consultas até que a próxima iteração ocorra e os dados sejam atualizados.

A segunda *thread*, visualizada na figura 4, do controle de visão executa o sistema de comunicação que pode ser implementado de várias maneiras, desde objetos *JSON* sendo transmitidos via *UDP* até um *message broker* com filas de consumo. Ao receber a mensagem de solicitação das informações, responde serializando os dados que ficaram salvos na primeira *thread*, fornecendo as informações com atraso mínimo.

3.1.2 Controlador de Comportamento

O controlador de comportamento implementa a tarefa que deve ser realizada pelo robô. Ele funciona como um gerente que coordena outros componentes, recebendo informações e enviando comandos com ações específicas serem a executadas. Na figura 4 é possível observar a sequência de passos realizados pelo controlador de comportamento.

Usando como exemplo uma partida de futebol, na figura 4 vemos o controlador de comportamento "jogador" consultando o componente de visão, que responde a posição da bola, previamente obtida durante o processo da figura 3. Em seguida, baseado nessas informações o comportamento envia o comando ao *walking gait* para caminhar na direção correta.

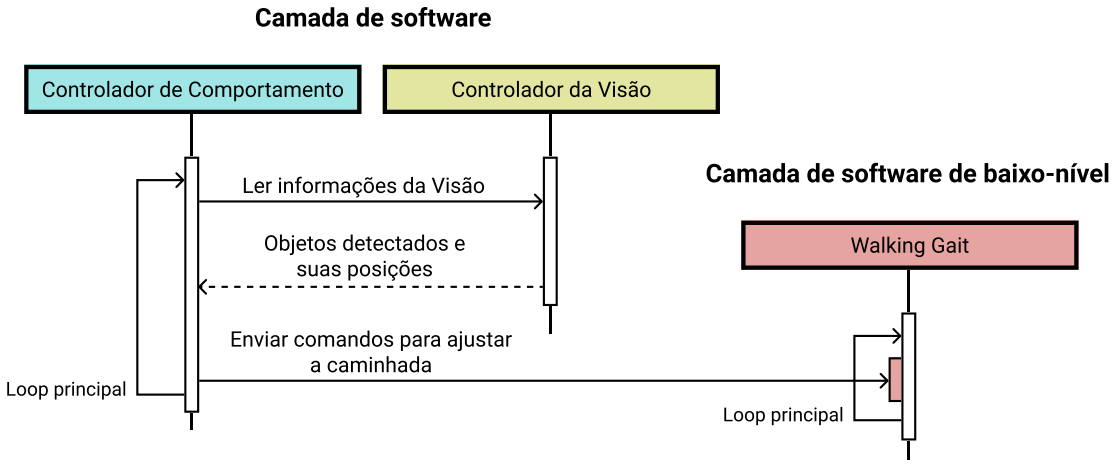

Figura 4: Diagrama de sequência do controlador de comportamento e suas dependências.

3.1.3 *Walking Gait* e Leitor de Orientação

O componente *walking gait* coordena e executa a caminhada. Ele monitora a USB aguardando os comandos de controle que contém as velocidades da caminhada, enviados a partir do controlador principal.

O componente leitor de orientação desempenha um papel importante dentro do *walking gait*. Ele é o responsável pela verificação da orientação da rotação do torso, uma vez que a *IMU* está situada nas costas de Arash. Desta forma, o *walking gait* faz correções nas juntas para compensar um eventual desvio na trajetória.

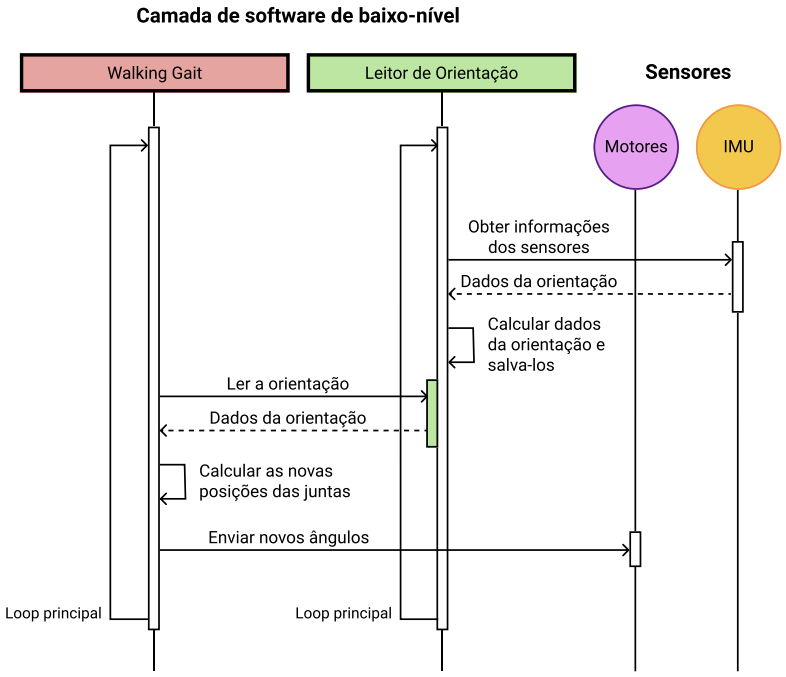

Figura 5: Diagrama de sequência do *walking gait* e do leitor de orientação.

Dentro da placa microcontroladora *OpenCM9.04*, o *walking gait* e o leitor de orientação rodam em *threads* dinstintas, recurso este habilitado pela biblioteca *MapleFreeRTOS*. Na figura 5 observa-se a execução de ambas a *threads* separadamente e a forma de interação entre elas.

No componente leitor de orientação, os dados são coletados da *IMU* e um processo de utilizando filtro de *Kalman* é utilizado para obter ua estimativa mais acurada das orientações que são salvas em variáveis. Já na *thread* que roda o *walking gait*, a orientação é continuamente consultada e utilizada para que os ângulos das juntas sejam atualizados.

Explicar este processo melhor

3.1.4 Graus de Liberdade e Atuadores

PARAGRAFO INTRODUTÓRIO.

Adiciona um parágrafo introdutório.

Tão importante quanto a configuração das juntas é a orientação dos atuadores para que os cálculos funcionem como o esperado.

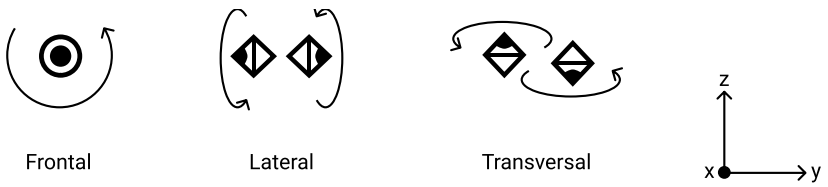

Figura 6: Orientação dos movimentos dos atuadores

A figura 6 exibe as possíveis orientações dos atuadores que são referenciadas ao longo do trabalho como frontal, lateral e transversal.

Atuadores com orientação frontal – ou simplesmente, atuadores frontais – propiciam rotação orientada pelo eixo X, que está aponta para o leitor (para fora do papel). Esse tipo de rotação também é referida como *roll*.

Atuadores laterais apontam para direita ou esquerda. Eles apresentam rotação com base no eixo Y (plano XZ), também conhecida como rotação *pitch*. Analogamente, atuadores transversais apontam para cima ou baixo, com base no eixo Z (plano XY), com movimento de rotação conhecido como *yaw*.

Na figura 7, observa-se o esquema de orientação dos atuadores aplicada no diagrama das juntas de Arash. Ainda na mesma figura, os "triângulos pretos preenchidos" representam apenas o fim de uma cadeia de atuadores. Assim, estes símbolos representam as mãos, que não possuem atuadores, e a ponta da cabeça, onde encontra-se a câmera.

Estas orientações são importantes durante os cálculos dos ângulos enviados às juntas.

Já que a inconsistência do modelo matemático e o robô físico causa inversão de movimentos durante a caminhada, causando reações inesperadas e possíveis danos.

Em Arash, todos os atuadores utilizados são produzidos pela Robotics.Co. Porém, devido a variação de carga nas diversas juntas, foram utilizados diferentes modelos da série MX. Esta decisão de projeto diminui bastante o custo final do robô.

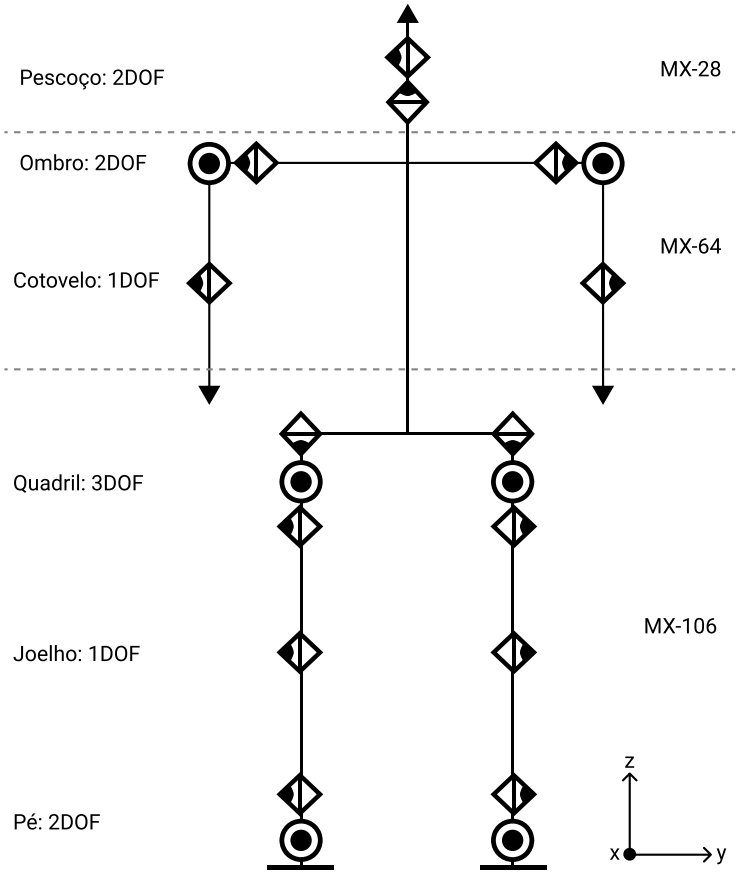

Figura 7: Diagrama de orientação dos atuadores de Arash

No pescoço, onde a carga é bem leve, "MX-28" são suficientes. Dois atuadores, um na posição transversal (o atuador mais baixo) que fornece o movimento panorâmico a cabeça e outro na posição horizontal, fornecendo o movimento de inclinação vertical da cabeça.

Nos braços, que podem sofrer uma carga maior, atuadores "MX-64" são utilizados. Isso é importante já que existem modalidades de competições, como o levantamento de peso, que testam a capacidade do carregamento de cargas.

Para as pernas, foram utilizados atuadores "MX-106" que são mais poderosos que os anteriores. Entretanto, na fase de projeto, simulações mostraram que durante o movimento de levantar-se do chão (em caso da recuperação de uma possível queda), o torque nas juntas do joelho era levado ao máximo suportado pelo "MX-106", podendo assim levar este motor à falha. Desta forma, 2 atuadores sincronizados passaram a formar esta junta, afim de

compartilhar a carga sofrida entre dois motores. Esta junta dupla não oferece nenhum impacto na implementação, já que os motores da série "MX-106" oferecem a capacidade de serem ligados e sincronizados via *hardware*. Assim, a nível de software controla-se apenas 1 único atuador. Desta forma, apesar das 20 DOF, Arash possui 22 motores.

3.2 A nova arquitetura

De acordo com os problemas e solução já discutidos em seções anteriores, esta seção irá apresentar as mudanças realizadas na arquitetura original apresentada na seção 3.1.

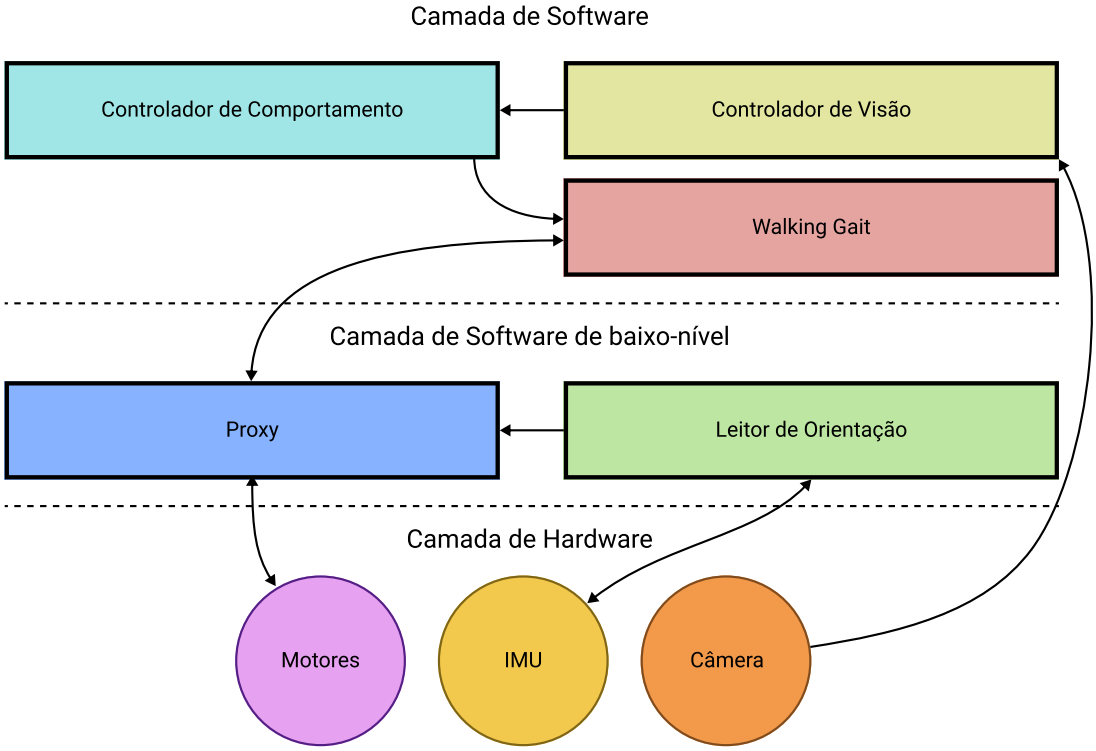

Figura 8: Diagrama de orientação dos atuadores de Arash.

A figura 8 mostra o *walking gait* – agora na camada de software – funcionando dentro do controlador principal, e um novo compontente chamado *proxy* é apresentado como interface os componentes de hardware e o *walking gait*. O papel do novo componente é de servir realmente como um *proxy* enviando comandos recebidos do controlador principal aos motores e também receber a orientação do componente leitor de orientação e fornecê-la ao controlador principal.

O controlador de comportamento continua a enviar os comandos de controle ao *walking gait*. Porém, ao invés de usar comandos binários via USB, agora utiliza objetos serializados via *JSON* via rede.

Especifi
esses co-
man-
dos

Entre o *walking gait* e o *proxy* a comunicação ocorre via USB @ 1 Mbps. Comandos de controle utilizando o *dynamixel protocol 1.0* são gerados e enviados diretamente do *walking gait* ao *proxy*, que apenas os encaminha aos motores. Esta estratégia foi adotada para habilitar a possibilidade da utilização de motores de diferentes tipos, não apenas da Robotis Co. Assim, apenas o *walking gait* é responsável pela geração de destes comandos específicos de cada motor enquanto o *proxy* apenas os encaminha.

Na camada de software de baixo nível, o leitor de orientação fornece os dados de orientação ao *proxy* que encarrega-se de disponibiliza-los ao *walking gait*.

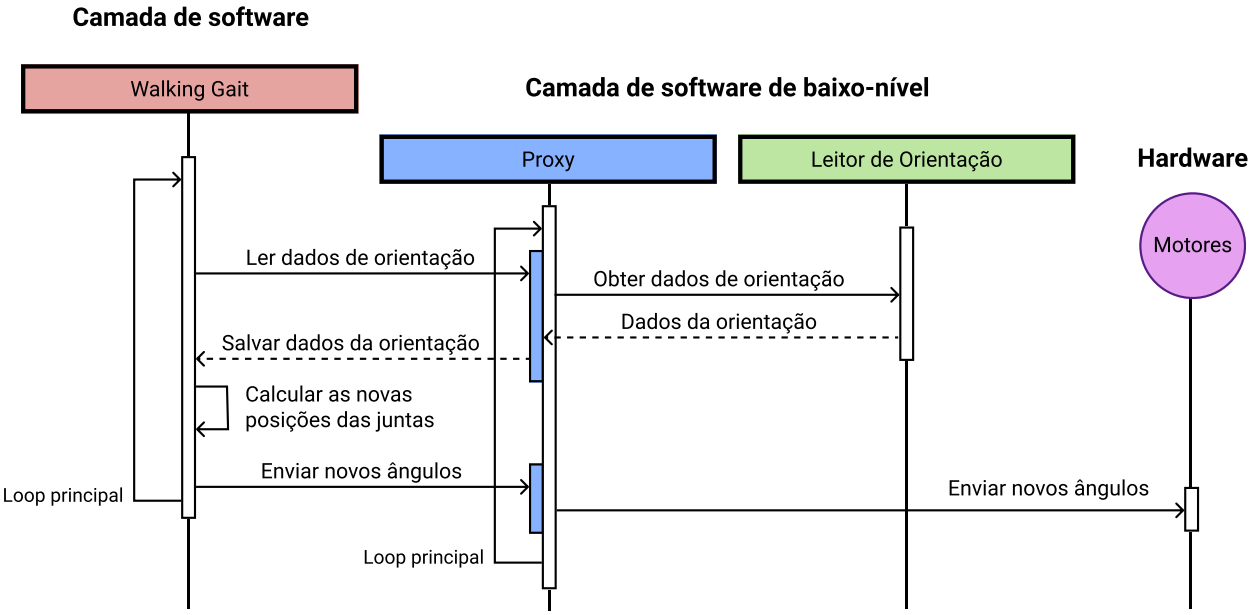

Figura 9: Diagrama de sequência simplificado com as modificações na arquitetura.

Na figura 9, é possível observar o diagrama de sequência com os componentes que foram atingidos pela mudança na arquitetura.

Nessa nova versão da arquitetura os cálculos dos ângulos das juntas são realizados pela nova implementação do *walking gait*. Para tanto, ele precisa dos dados de orientação que devem ser fornecidos pelo componente leitor de orientação, que ainda encontra-se dentro da *OpenCM9.04*. Após os cálculos realizados agora o *walking gait* deve comunicar-se com o *proxy* que encaminhará os novos ângulos aos motores.

3.3 *Walking gait*

Nesta seção detalhes arquiteturais da implementação do *walking gait* serão descritos.

O *walking gait* é o componente que coordena a caminhada mantendo o estado drobô,

sabendo onde cada perna encontra-se e o seu estágio durante a caminhada. Desta forma, quando algum evento de perturbação, ou de ajuste, é disparado o *walking gait* inicia a atualização das juntas a partir do estado atual. Assim, podemos descrever a caminhada como um evento contínuo no tempo.

Para realizar a tarefa de forma adequada, o componente faz uso de multi-processamento. Três *threads* dentro do componente são responsáveis por tarefas específicas essenciais ao complemento do comportamento geral da caminhada: A *thread* de rede, a *thread* atualização dos motores e a *thread* principal.

3.3.1 *Thread* principal: Da geração da trajetória até a aplicação às juntas

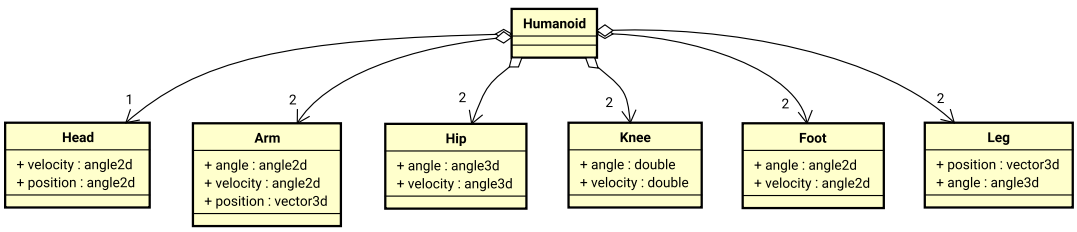

Figura 10: Diagrama de domínio do *walking gait*.

A figura 10 mostra o diagrama de domínio das classes relevantes aos dados que o componente manipula. Nela podemos ver a classe $Humanoid$ é uma agregação de diversas outras classes que representam cada parte de Arash. É possível também observar que nas classes $Head$, Arm, Hip, $Knee$, $Foot$ existem dados ângulo e velocidade. Isso se dá por causa que componente utiliza essas informações para calcular o próximo estado do sistema em caso de algum distúrbio. Nota-se também, que a classe Leg aparece de forma ambígua, já que existem as definições de Hip, $Knee$ e $Foot$. Porém, a classe Leg, representa os dados da cinemática inversa de uma perna, enquanto as demais classes guardam o estado atual das partes que representam.

Nota-se, também, a notação dos dados utilizados: $angle2d$, $angle3d$ e $vector3d$. Note que todos os tipos possuem o sufixo numérico, que indica a quantidade de dimensões aquela classe guarda, e a letra, que indica o tipo de seus dados, no caso d que significa *double*. Para ângulos, 2 dimensões significa que as dimensões *pitch* e $roll$ são guardadas; já para 3 dimensões as orientações $roll$, *pitch* e yaw são guardadas. Para vetores, os valores x, y e z são guardados.

3.3.2 Servidor de controle

A *thread* de rede roda um servidor *UDP* que aceita objetos serializandos em *JSON* contendo informações de controle.

3.3.3 Abstração dos motores

4 A matemática dos movimentos

Para caminhar, Arash move suas juntas de uma forma sincronizada mantendo o equilíbrio. Cada passo é planejado e possíveis perturbações são compensadas o mais breve possível afim de preservar a trajetória e evitar quedas. O processo de caminhada é dividido em ciclos. Cada ciclo é dado pelo movimentos de passo da perna esquerda seguido pelo movimento da perna direita. A cada momento no tempo do ciclo, levando em consideração também outras variáveis, as trajetórias fornecem a posição e orientação dos pés do robô.

Falar sobre walking gait omnidirecional

Citar o ZMP

O conceito de caminhada omnidirecional envolve o controle através de 3 velocidades: V_x, V_y, V_ω. Onde V_x é a velocidade em direção à frente, com valores positivos para a frente e negativos para caminhar de costas; V_y é a velocidade lateral com onde passos para o lado são definidos com valores positivos implicando em movimentos laterais à direita; e, finalmente, V_ω é a velocidade de rotação por passo com valores positivos indicando rotação também à direita.

$$-1.0 <= V_x <= 1.0 \quad , \quad -1.0 <= V_y <= 1.0 \quad , \quad -1.0 <= V_\omega <= 1.0 \tag{4.1}$$

Valores normalizados entre -1.0 e 1.0 são utilizados para indicar as velocidades. Significando 0.0 como o valor de velocidade nula, ou sem movimento, 1.0 o valor de velocidade máxima e -1.0 o valor de velocidade máxima no sentido oposto. Valores normalizados foram adotados para abstrair valores de velocidades absolutos já que futuras aplicações deste trabalho podem utilizar diferentes configurações de robôs. Desta forma, o controlador de comportamento pode abstrair a configuração do robô que está rodando-o e enviar velocidades relativas. Adicionalmente, valores normalizados provém a integração perfeita entre as funções trigonométricas para a definição das trajetórias senoidais adotadas por *Kamiri et al.*

Com os valores de velocidades definidos, é preciso .

terminar este pará-

A cada iteração novas dois vetores T, de transferência, e R, de rotação, de cada pé são obtidas. Os elementos dos vetores T e R são detalhados na seção 4.2. Em poder desses vetores, utiliza-se os conceitos da cinemática inversa para gerar os ângulos de cada junta. Finalmente, os ângulos gerados são enviados aos motores – como especificado na subseção 3.3.3 – e o a iteração é finalizada.

$$T = [Foot_x, \quad Foot_y, \quad Foot_z], \quad R = [Foot_{roll}, \quad Foot_{pitch}, \quad Foot_{yaw}] \tag{4.2}$$

4.1 Orientação

Durante a implementação deste trabalho, nenhuma modificação foi realizada no componente de orientação originalmente desenvolvido por Kamiri *et al.* Entretanto, para a implementação do cálculo da trajetórias é necessário algumas informações sobre o resultado final do componente de orientação.

No final do processamento do componente de orientação, obtém-se como saída um vetor com de aceleração linear nos eixos x e y, como mostra a equação 4.3. Então, o vetor *Accel* é utilizado para o cálculo da detecção de "empurrões", ou distúrbios, como visto nas equações 4.4 e 4.5, que representação a variação na aceleração linear entre a iteração atual e a anterior.

$$Accel = [Accel_x \quad Accel_y] \tag{4.3}$$

$$Push_x = Accel[x]_t - Accel[x]_{t-1} \tag{4.4}$$

$$Push_y = Accel[y]_t - Accel[y]_{t-1} \tag{4.5}$$

4.2 Trajetórias

Como definido na equação 4.2, os vetores T e R são o resultado da trajetória calculada a cada iteração. Cada elemento de seus elementos é dado por funções específicas em cada eixo correspondente (KARIMI; SADEGHNEJAD; BALTES, 2016):

$$Foot_x = \left(\frac{-cos(t)+1}{2}\right) \times \left(\frac{tanh(V_x + Push_x) + H_{leg}}{\pi}\right) \tag{4.6}$$

$$Foot_y = \big((-sin(t) \times B_{swing}) + (cos(t)+1)\big) \times \left(\frac{tanh(V_y + Push_y) + H_{leg}}{2\pi}\right) \tag{4.7}$$

$$Foot_z = \left(\frac{sin(t)}{t+1/2}\right) \times \left(\frac{tanh(V_x + V_y) \times H_{leg}}{\pi}\right) \tag{4.8}$$

$$Foot_{roll} = 0 \tag{4.9}$$

$$Foot_{pitch} = 0 \tag{4.10}$$

$$Foot_{yaw} = \left(\frac{sin(V_\omega)+1}{2}\right) \times \left(\frac{-cos(t)+1}{2} \times V_\omega\right) \tag{4.11}$$

Onde t representa o tempo atual normalizado em cada ciclo; B_{swing} representa a quantidade de compensação de balanço lateral que o corpo deve efetuar para compensar a dinâmica do movimento das pernas; $Push$ é a quantidade de distúrbio causado por desvios na trajetória, ele pode ser definido em dois eixos x e y.

Nota-se que as funções $Foot_{roll}$ e $Foot_{pitch}$ são definidas como 0 pois, em resultados experimentais, os efeitos se mostraram nulos (KARIMI; SADEGHNEJAD; BALTES, 2016).

4.3 Cinemática Inversa

Descobrir a posição de um parte do robô a partir dos ângulos de suas juntas, *forward kinematics*, é uma tarefa fácil e pode ser realizada eficientemente. Porém, o oposto – dado um ponto e calcular os ângulos das juntas que alcancem aquele ponto – não é uma tarefa tão simples. Enquanto a problemas envolvendo a *forward kinematics* sempre possuem apenas uma única solução, já na IK (do inglês *inverse kinematics* ou cinemática inversa) podem haver múltiplas soluções ou nenhuma (SPONG; HUTCHINSON; VIDYASAGAR, 2005). Pode-se observar isto ao tocar a ponta do próprio nariz, existem diversas combinações de posições do ombro, cotovelo, pulso e falanges que resolvem este problema.

Aplicando a teoria da cinemática inversa ao problema proposto, Kamiri *et al* propôs uma solução direta aproveitando-se das limitações impostas pela configuração das juntas de Arash.

$$P = [Hip_{yaw}, Hip_{roll}, Hip_{pitch}, Knee, Foot_{pitch}, Foot_{roll}] = iK(T, R) \tag{4.12}$$

Na equação 4.12, vemos a definição do vetor P com os valores de todas as juntas. O vetor P é a saída da função iK que recebe o vetor de T e R já definidos na equação 4.2.

O processo de formulação das equações inicia-se a partir do nó principal do quadril descendo para o fim da cadeia, nos pés. Desta forma, a figura 11 mostra uma tomada de cima de Arash auxiliando na visualização das equações 4.13

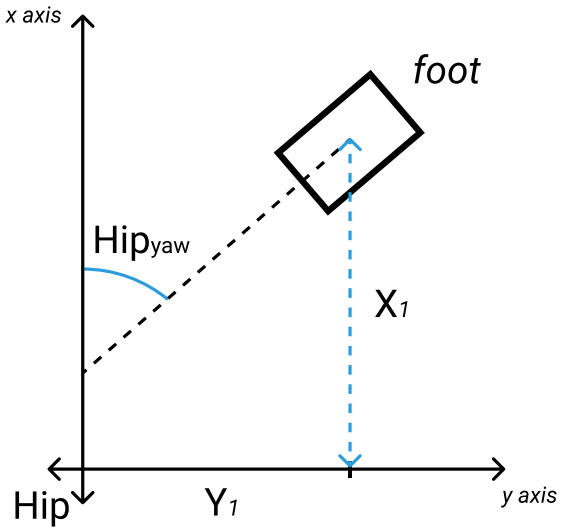

Figura 11: Diagrama da visão superior da estrutura de Arash que representa da equação 4.13 até 4.16

(KARIMI; SADEGHNEJAD; BALTES, 2016)

$$P[Hip_{yaw}] = R_{yaw} \tag{4.13}$$

$$X_2 = T_x cos(R_{yaw}) + T_y sin(R_{yaw}) \tag{4.14}$$

$$Y_2 = -T_x sin(R_{yaw}) + T_y cos(R_{yaw}) \tag{4.15}$$

$$Z_2 = T_z \tag{4.16}$$

Seguindo o processo de parametrização dos ângulos das juntas, observa-se na figura 12 a visão frontal de Arash provendo uma visualização geométrica das equações 4.17 até 4.21.

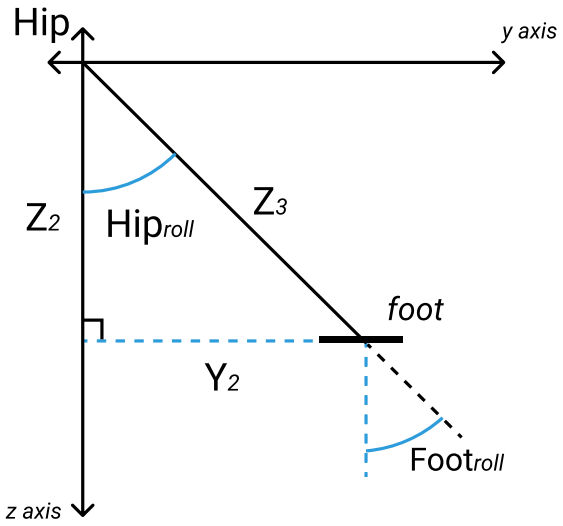

Figura 12: Diagrama da visão frontal de Arash que representa da equação 4.17 até 4.21 (KARIMI; SADEGHNEJAD; BALTES, 2016)

$$P[Hip_{roll}] = atan\left(\frac{Y_2}{Z_2}\right) \tag{4.17}$$

$$P[Foot_{roll}] = -P[Hip_{roll}] + R_{roll} \tag{4.18}$$

$$X_3 = X_2 \tag{4.19}$$

$$Y_3 = Y_2 \tag{4.20}$$

$$Z_3 = \sqrt{{Y_2}^2 + {Z_2}^2} \tag{4.21}$$

Seguindo o processo de parametrização dos ângulos das juntas, observa-se na figura 12 a visão frontal de Arash provendo uma visualização geométrica das equações 4.17 até 4.21.

$$d_{foot} = \sqrt{Z_3^2 + X_3^2} \tag{4.22}$$

$$\alpha = atan\left(\frac{X_3}{Z_3}\right) \tag{4.23}$$

$$\beta = acos\left(\frac{d_{foot}}{2 \times Leg_{len}}\right) \tag{4.24}$$

$$P[Hip_{pitch}] = \alpha + \beta \tag{4.25}$$

$$P[Knee] = -2\beta \tag{4.26}$$

$$P[Foot_{pitch}] = \gamma = -\alpha + \beta + R_{pitch} \tag{4.27}$$

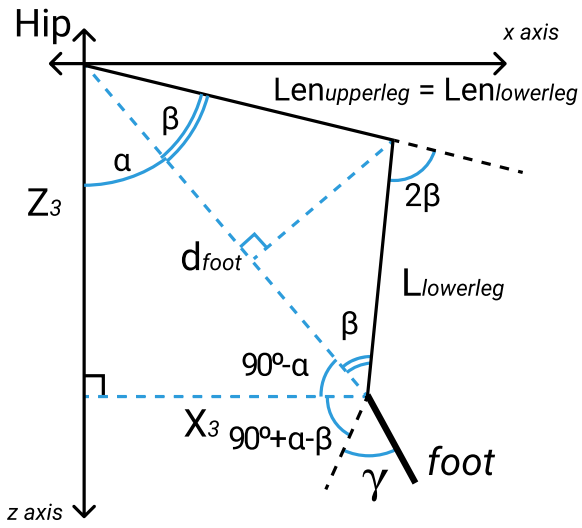

Figura 13: Diagrama da visão lateral de Arash que representa da equação 4.17 até 4.21 (KARIMI; SADEGHNEJAD; BALTES, 2016)

Adicionalmente, em cada junta, é somado um parâmetro de deslocamento – não incluso nas equações –, também conhecido como *offset*, para compensar possíveis erros provenientes de desalinhamentos no momento de montagem do robô, ou folgas nas engrenagens dos atuadores.

Finalmente, com as equações definidas o *walking gait*, com o auxílio dos componentes de orientação e *proxy*, é apto a fazer os cálculos necessários para determinar os movimentos de caminhada.

5 Capítulo 5

GUIA:

Detalhar implementação, integração e simulação e testes.

5.1 Seção 1

GUIA:

Detalhar implementação.

5.2 Seção 2

GUIA:

Detalhar simulação.

5.3 Seção 3

GUIA:

Detalhar simulação.

Seção 3

6 Considerações finais

As considerações finais formam a parte final (fechamento) do texto, sendo dito de forma resumida (1) o que foi desenvolvido no presente trabalho e quais os resultados do mesmo, (2) o que se pôde concluir após o desenvolvimento bem como as principais contribuições do trabalho, e (3) perspectivas para o desenvolvimento de trabalhos futuros, como listado nos exemplos de seção abaixo. O texto referente às considerações finais do autor deve salientar a extensão e os resultados da contribuição do trabalho e os argumentos utilizados estar baseados em dados comprovados e fundamentados nos resultados e na discussão do texto, contendo deduções lógicas correspondentes aos objetivos do trabalho, propostos inicialmente.

6.1 Principais contribuições

Texto.

6.2 Limitações

Texto.

6.3 Trabalhos futuros

Texto.

Referências

KARIMI, M.; SADEGHNEJAD, S.; BALTES, J. *Online Omnidirectional Gait Modifications for a Full Body Push Recovery Control of a Biped Robot*. 2016.

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

APÊNDICE A – Primeiro apêndice

Os apêndices são textos ou documentos elaborados pelo autor, a fim de complementar sua argumentação, sem prejuízo da unidade nuclear do trabalho.

ANEXO A – Primeiro anexo

Os anexos são textos ou documentos não elaborado pelo autor, que servem de fundamentação, comprovação e ilustração.

Todo list