Curso de Ciência da Computação
Universidade Estadual do Mato Grosso do Sul

Sistema de Controle para Bípede Microcontrolado

Esmael Dias Prado

Prof. Dr. Rubens Barbosa Filho (Orientador)

Dourados - MS
2015

Sistema de Controle para Bípede Microcontrolado

Esmael Dias Prado

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Esmael Dias Prado e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Dourados, 16 de Novembro de 2015.

Prof. Dr. Rubens Barbosa Filho
(Orientador)

Curso de Ciência da Computação
Universidade Estadual do Mato Grosso do Sul

Sistema de Controle para Bípede Microcontrolado

Esmael Dias Prado

Novembro de 2015

Banca Examinadora:

- Prof. Dr. Rubens Barbosa Filho (Orientador)
  Ciência da Computação - UEMS

- Prof. Dr. Cleber Valgas Gomes Mira
  Ciência da Computação - UEMS

- Prof. Dr. Osvaldo Vargas Jaques
  Ciência da Computação - UEMS

À Deus.
Dedico também a minha esposa Elaine, pelo carinho, compreensão e companheirismo.

"Comece fazendo o que é necessário, depois o que é possível e, de repente, você estará fazendo o impossível."

(São Francisco de Assis)

Agradecimentos

Agradeço a todos aqueles que de uma forma contribuíram para a conclusão deste projeto, primeiramente a um Deus supremo razão do meu existir e fonte de segurança nos momentos de insegurança, em especial ao meu professor orientador Dr. Rubens Barbosa Filho, por ter acreditado e ajudado com o seu conhecimento na elaboração em todas as fases do projeto.

Aos meus amigos, Ronnei Peterson por ter ajudado nos testes iniciais em software Delphi e ao Dr. Marcelo Teixeira com as suas praticidades colaborou com inúmeras ideias, também ao apoio que me deram nos momentos de muita dificuldade, permitindo obter forças.

Ao técnico em mecânica da UFGD José Carlos Venturin, que auxiliou com os desenhos do protótipo em CAD.

Os professores, mestre Delair Osvaldo Martinelli Júnior e Dr. Dalton Pedroso de Queiroz, que ajudaram diretamente com o conhecimento em grande valor. Especialmente ao professor Dr. Fabrício Sérgio de Paula que colaborou com as ideias iniciais do código de controle eficiente para os servos.

Ao amigo Carlos Fabiano Capato que colaborou com diversas idéias.

Aos professores do curso de Ciência da Computação da UEMS, por todo o conhecimento, paciência e dedicação oferecidas para a formação de profissionais qualificados. A amizade de vocês também foi muito importante durante toda a fase do curso.

Também a todos os autores bibliográficos referenciados neste projeto, que contribuíram nos estudos e no desenvolvimento deste projeto.

E principalmente aos meus familiares por terem me apoiado durante todo o trajeto do curso, por sempre acreditarem em meu potencial e por tudo que possam ter feito por mim, principalmente pelo amor e carinho.

Resumo

O objetivo deste trabalho é o desenvolvimento de um robô bípede de pequena escala e baixo custo para aplicação didática. Os robôs são ferramentas úteis para diferentes propósitos, incluindo trabalhos em condições difíceis e perigosas. Outras aplicações podem ser destacadas como aplicações médicas, militares, vigilância, salvamentos, serviços subaquáticos e exploração do espaço. Na medicina exoesqueleto robóticos são utilizados como auxiliares nas operações, serviços de enfermagem e reabilitação de pessoas com deficiência. Estes sistemas robóticos podem ser integrados e servir a diferentes propósitos: sociais, educativos, terapêuticos, exploração e diversão, destacando-se nas aplicações a substituição do trabalho humano. A produção do protótipo didático permitiu a compreensão de conceitos fundamentais da física dos movimentos, reciclagem seletiva e aplicação dos princípios de robótica por meio da utilização de peças obtidas do lixo tecnológico para a redução de custos da pesquisa. As métricas, escala e construção do protótipo foram realizadas conforme seleção das peças, montagem e produção do robô. A seguir foram desenvolvidos os algoritmos, interfaces, protocolos de comunicação, testes do protótipo, adequações, tabulação e discussão dos resultados experimentais. A construção do bípede robótico mostrou-se viável apesar dos custos e complexidade da montagem do protótipo e das adequações para o seu funcionamento. Os comandos de controle foram realizados por computador e/ou por memória interna para a execução dos ciclos de marcha. No projeto considerou-se o aprimoramento na técnica de elaboração dos circuitos eletrônicos de baixo ruído. O desenvolvimento da fonte de alimentação de comutação de baixa tensão e corrente elevada foi necessária para a realização do projeto. O protótipo foi controlado por algoritmos de computador em Java e módulo de interface microcontrolado. O *software* embutido no microcontrolador "PIC18F4550" e interface "USB" permitiu o controle simultâneo de quinze atuadores. O *software* utilizado converte os valores de movimento em hexadecimal e os envia para a memória interna da interface, incluindo o gerenciamento da bateria. Os estudos da dinâmica dos movimentos e dos algoritmos de inteligência artificial não foram realizados devido à complexidade do projeto.

Palavras-chave: *Robótica*, *Sistema Bípede*, *Servo Motores*, *Microcontrolador*, *Controle*.

Abstract

The objective of this work is the development of a small-scale biped robot and low cost for didactic application. Robots are useful tools for different purposes, including work in difficult and dangerous conditions. Other applications can be highlighted as medical, military, surveillance, rescue, underwater services, and space exploration. In medical practice, exoskeleton robots are used as auxiliary operations, nursing, and rehabilitation services for people with disabilities. These robotic systems can be integrated and serve different purposes: social, educational, therapeutic, exploration and fun, specially in the replacement of human labor. The production of a didactic prototype allowed the understanding of fundamental concepts of physical movements, selective recycling and application of the principles of robotics by using parts obtained from technological waste to reduce research costs. Metrics, scale and construction of the prototype were carried out as selection of parts, assembly, and production of the robot. Moreover, we developed algorithms, interfaces, communication protocols, prototype testing, adjustments, tabulation, and discussion of the experimental results. The construction of the biped robot is viable despite the cost and complexity of assembly of the prototype and adjustments for its operation. The control commands were carried out by computer and/or internal memory of the execution of gait cycles. In this project considers the improvement in the preparation technique of electronic circuits of low noise. In this project considers the improvement in the preparation technique of electronic circuits of low noise. The development of a low voltage and high current switching power supply was needed to carry out the project. The prototype was controlled by computer algorithms in Java and a microcontroller interface module. The embedded software in the microcontroller "PIC18F4550" and "USB" interface, allows the simultaneous control of fifteen actuators. The software converts the motion in hexadecimal values and sends them to the internal memory of the interface, including the battery management. Studies of the dynamics of movements and artificial intelligence algorithms were not done due to the complexity of the project.

Keywords: *Robotics*, *Biped System*, *Servo Motors*, *Micro Controller*, *Control*.

Sumário

Lista de Abreviaturas e Siglas

IEEE Inglês: Engineering in Medicine and Biology Society 2

RAM Inglês: Random Access Memory 8

ROM Inglês: Read Only Memory 8

EEPROM Inglês: Electrically-Erasable Programmable Read-Only Memory 8

ADC Inglês: Analog-to-Digital Converter 8

DAC Inglês: Digital-to-Analog Converter 8

GND Inglês: Graduated Neutral Density Filter 9

PWM Inglês: Pulse Width Modulation 10

Li-Po Baterias de Polímero de Íon Lítio 19

CPU Inglês: Central Processing Unit ou Unidade Central de Processamento 35

ICSP Inglês: In Circuit Serial Programming 35

USB Inglês: Universal Serial Bus 35

I2C Inglês: Inter-Integrated Circuit, Barramento Serial Desenvolvido Pela Philips 35

PICKit-2 Inglês: Development Programmer/Debugger from Microchip 41

RISC Inglês: Reduced Instruction Set Computer ou Computador com um Conjunto Reduzido de Instruções 41

SERB Software Embarcado Robô Bípede 44

HID Inglês: Human Interface Devicer 45

CDC Inglês: Communication Device Class 45

MSB Inglês: Mass Storage Device 45

RBPC Robô Bípede - PC Controle 61

PIC Família de microcontroladores fabricados pela Microchip Technology 79

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos primórdios da robótica, a ficção científica descrita nas obras de Asimov (1969) remete às reflexões sobre impactos sociais que os avanços tecnológicos nesta área provocaram em seu tempo. Em um momento de efervescência científica, as ideias de Asimov incentivaram a busca por robôs humanoides.

Atualmente houve um aumento exponencial na velocidade dos processadores, bem como a redução de peso dos materiais utilizados e a miniaturização dos circuitos eletrônicos. Deste modo, chegou-se ao desenvolvimento de unidades autônomas[1] com o objetivo de realizar uma gama de trabalhos nas condições mais adversas. Dentre as aplicações de destaque estão o desarmamento de bombas, a inspeção em áreas perigosas ou inóspitas, o tratamento de lixo tóxico, a exploração subaquática e espacial, cirurgias, mineração, busca e resgate, entre outro abordado por Romano (2002).

Outro exemplo de robô moderno de grande importância para o Brasil foi apresentado na abertura da Copa do Mundo 2014. Nesta apresentação, um robô em forma de exoesqueleto foi controlado pelo pensamento de seu usuário. Este robô permitiu a um homem paraplégico conseguir dar o chute inicial em uma bola no jogo de estreia da Copa do Mundo. Este trabalho é resultado de uma pesquisa descrita no artigo de Fitzsimmons et al. (2009) e abordado por Araújo Neto (2013). Foi uma representação grande e nobre dentre as possibilidades das aplicações da robótica e a tecnologia bípede em benefício da humanidade.

Recentemente Gerasimenko et al. (2015), melhorou o sistema de controle do exoesqueleto por meio de controle cerebral. Utilizou-se a própria força muscular do usuário

[1]São compostas por um sistema automático de controle pelo qual os mecanismos verificam seu próprio funcionamento, efetuando medições e introduzindo correções, sem a necessidade da interferência do homem a fim de realizar uma determinada atividade ou trabalho. Em seu uso moderno, a automação pode ser definida como uma tecnologia que utiliza comandos programados para operar um dado processo, combinados com retroação de informação para determinar que os comandos sejam executados corretamente, frequentemente utilizados em processos antes operados por seres humanos Dorf (2001).

para gerar o movimento. Contudo o auxílio mecânico de um exoesqueleto simplificado será necessário e os músculos debilitados deverão possuir articulações com motores. Este sistema é mais prático por ser mecanicamente simplificado. A demonstração do projeto foi apresentada na conferência Internacional do IEEE[2], conforme exibe a **Figura 1.1**

Figura 1.1: Mark Pollock e o treinador Simon O'Donnell[3].

Os sistemas automatizados e autômatos tendem em alguns casos a substituir o trabalho humano, podendo alguns robôs ter o objetivo de divertir, entreter e ou fins terapêuticos.

Segundo Rocha (2011), os avanços recentes recentes nas tecnologias da informação e na comunicação foram de tamanha relevância que provocaram mudanças sobre as mais diversas áreas do conhecimento. Estas mudanças Interferiram nas áreas econômica, social e política por meio de um processo acelerado e irreversível.

As tecnologias de robôs humanoides evoluíram abruptamente, no entanto suas limitações são enormes com relação ao que se espera desse tipo robô.

O estudo da locomoção sobre pernas aplicado em robótica é um dos problemas mais desafiantes. Isto se deve ao fato de o controle e a simulação de bípedes ter demorado mais a avançar do que os outros robôs multipernas Demasi (2012), devido a maior exigência de

[2]Conferência Internacional Anual do IEEE: *37th Annual International Conference of the IEEE (Engineering in Medicine and Biology Society)*, Conferência Intitulada: ***"Iron 'ElectriRx' Man: Overground Stepping in an Exoskeleton Combined with Noninvasive Spinal Cord Stimulation after Paralysis."*** Citengine (2015).

[3]Crédito: Cortesia de Mark Pollock. Site aborda o conteúdo da palestra: ***"Iron 'ElectriRx" Man: Overground Stepping in an Exoskeleton Combined with Noninvasive Spinal Cord Stimulation after Paralysis."*** ScienceDaily (2015).

estabilidade e maior complexidade de projeto Katic (2002).

De acordo com Toscano (2011), a busca de maior mimetismo das características humanas em sistemas mecânicos tem atraído os pesquisadores pela fascinação dos sistemas bípedes e humanoides. O avanço do desenvolvimento de manipuladores antropomórficos de forma sintética humana completa, deu início a um novo paradigma dentro da robótica com relação à imitação do deslocamento humano.

Dentro dessa perspectiva que integra os robôs no ambiente social humano, surge o conceito de robô social, que se comporta de forma favorável aos seus objetivos e os de sua comunidade Duffy (2003).

As pesquisas sobre deslocamento humano aplicado em máquinas do tipo de locomoção bípede justificam-se por sua mobilidade similar a do ser humano. Este tipo de locomoção permite acessar quase todos os tipos de terrenos, com obstáculos, superfície inclinadas, rugosas, subir escadas e até mesmo andar sem atuadores Westervelt et al. (2007).

Os robôs bípedes apresentam potencialidades de uso em áreas perigosas ao ser humano, como área militar e reabilitação de deficientes físicos. Para isto é necessário explorar cada vez mais o potencial dos robôs bípedes com relação aos métodos eficientes de modelagem matemática e da geração de trajetórias mais estáveis, pois até agora não apresentaram a mobilidade e destreza comparada ao ser humano Toscano (2011).

Segundo Honda (2015)[4] e Silva (2008), o desafio do desenvolvimento de um sistema robótico bípede é enorme. Projetar um robô que pode imitar as complexidades do movimento humano e genuinamente ajudar as pessoas não é uma tarefa fácil.

Pode-se tomar como exemplo, o robô ASIMO, que levou mais de duas décadas de estudo, pesquisa e testes antes dos engenheiros da Honda alcançarem o seu objetivo Sakagami et al. (2002). As versões da evolução do robô ASIMO são ilustradas na **Figura 1.2**.

Diante deste cenário, o desafio deste projeto passa a ser a concepção, desenvolvimento e a aplicação de testes em um robô humanoide bípede baseado em pesquisas já desenvolvidas no meio acadêmico.

[4] *Honda Motor Company Limited*: **Instituto Honda de Pesquisa Tecnológica e Companhia Limitada**, é um dos mais importantes fabricantes de automóveis e motocicletas do mundo. Em 1986, os engenheiros da Honda propuseram a criar um robô que anda. Mediante a evolução originou o robô Humanóide ASIMO. Abordou a questão: “Por que criar um robô humanóide?”.

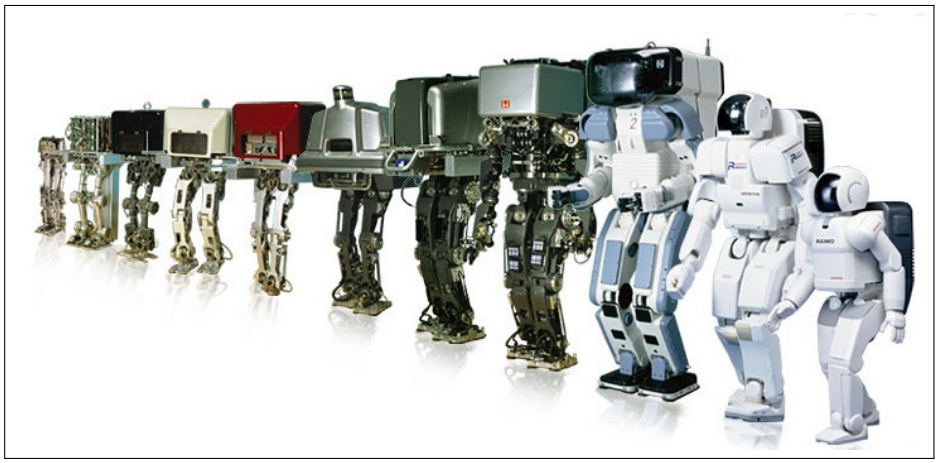

Figura 1.2: Versões de evolução do robô ASIMO desenvolvido pela HONDA S/A. Fonte: Honda (2015).

1.1 Objetivos

O objetivo principal deste trabalho é o desenvolvimento de um protótipo robótico humanoide bípede de pequena escala e baixo custo financeiro.

1.1.1 Objetivos Específicos

Os objetivos específicos do trabalho são:

- Estudo e revisão bibliográfica dos modelos robóticos de sucesso presentes na literatura especializada da área.
- Criação de um módulo de interface microcontrolado com software embarcado para controle simultâneo de atuadores.
- Desenvolvimento de uma fonte chaveada de baixa tensão e alta corrente.
- Desenvolvimento de um módulo de comando via microcomputador com interface USB.
- Desenvolvimento de um módulo de memória com protocolo de conversão de decimal para hexadecimal.
- Desenvolvimento de um módulo de monitoramento de energia.

Resultados esperados:

- Desenvolver um protótipo de robô bípede, comandado por computador e ou via memória interna que execute ciclos de caminhada conforme é abordado por Silva (2001).
- Construir um protótipo de robô bípede com características peculiares, levando em consideração as limitações de orçamento e recurso.
- Apresentar um protótipo com alto nível de complexidade de mecânica, eletrônica, programação, alto fluxo de dados, controle simultâneo de motores com hardware e tempo limitado. Possibilitado pela "USB" de 64 *Bytes* por pacote, um comando único de dados a todos os motores e o controle dos temporizadores do microcontrolador.
- Colaborar com a ampliação do conhecimento sobre estudo e desenvolvimento de robôs bípedes a divulgação de conhecimento por meios de artigos, relatórios técnicos, páginas de "WEB", entre outros.

1.2 Justificativa

O desenvolvimento deste projeto tem por objetivo o potencial de melhorar o processo de aprendizagem em cursos tecnológicos de computação, diante da dificuldade de entendimento de alguns conceitos abstratos que poderiam ser explorados através da aplicação prática de princípios da robótica e da dificuldade de aquisição dos Kits robóticos.

Por menor que sejam os robôs humanoides, os investimentos tendem a ser altos. Desta forma, o desafio passa a ser a elaboração de uma lista de componentes de baixo custo que possa atender o objetivo proposto. Utilizando-se do lixo tecnológico para redução do custo, auxiliar outros projetos de pesquisa com materiais alternativos e ideias de desenvolvimento robótico.

Interesse particular pela robótica, possibilidade de trabalhar em um projeto que envolva o hardware e software e a compreensão abrangente sobre controle de sistemas via computador. É Importante o amadurecimento deste tipo de conhecimento por parte da comunidade acadêmica, bem como despertar uma consciência na sociedade para a necessidade e importância deste tipo de estudo e pesquisa.

1.3 Metodologia

A metodologia adotada neste trabalho segue as seguintes etapas:

1. Estudo e revisão bibliográfica da literatura da área;
2. Elaboração das métricas e escalas a serem utilizadas na construção do robô;
3. Elaboração e construção dos materiais utilizados no desenvolvimento do robô;
4. Montagem e encaixe das peças;
5. Desenvolvimento dos algoritmos, interfaces e protocolos de comunicação;
6. Testes com o protótipo;
7. Escrita do Trabalho de Conclusão de Curso.

1.4 Organização do Texto

Este projeto foi dividido em sete capítulos para conter toda a estrutura. No **Capítulo 1** é apresentada a introdução. O **Capítulo 2** apresenta a fundamentação teórica é as definições dos componentes e montagens. No **Capítulo 3** temos a montagem do sistema e a apresentação dos recursos adicionais para viabilização do projeto. No **Capítulo 4** temos a abordagem do sistema eletrônico microcontrolado do Bípede. O **Capítulo 5** discute o desenvolvimento do sistema embarcado e programa do computador. No **Capítulo 6** são discutidos os testes e movimentos do robô. Por fim, no **Capítulo 7** são apresentas as conclusões sobre o projeto e discute projetos futuros.

Capítulo 2

Robótica BÍPEDE

2.1 Fundamentos

Atualmente o mundo moderno apresenta centenas de soluções robóticas extremamente complexas para os mais diferentes tipos de aplicação. Dentre estas soluções, certamente por ser o mais famoso, o robô bipede ASIMO recebe o maior destaque. O ASIMO foi desenvolvido no Japão pelos engenheiros da HONDA S/A. O projeto iniciou em 20 novembro de 2000 Honda (2007). Este possui 1,3 metro de altura, 54 quilogramas e possui 20 processadores. Curiosamente, seu nome não possui referência ao escritor russo de ficção científica Isaac Asimov. Em japonês, ASIMO significa "também com pernas", mas seu nome provêm da sigla em inglês "*Advanced Step in Innovative Mobility*" ou seja "Desenvolvimento Avançado em inovações para Mobilidade" Hirose and Ogawa (2007).

São inúmeras as funcionalidades atribuídas ao robô ASIMO. Dentre as quais podemos citar o fato de andar em superfícies irregulares, virar, pegar em coisas e reconhecer pessoas através das suas câmeras que funcionam como olhos. ASIMO pode correr com movimentos fluidos e atingir[1] 6 Km/h. Além disso, possui sistema de comunicação que lhe confere a capacidade de trabalhar em equipe, partilhar informações e coordenar tarefas. Apresenta também um sistema de ligação desenvolvido que partilha informações como sua localização ou qual tarefa está desempenhando. Além disso, na versão "P3", o ASIMO ganhará mais sensores que darão sensibilidade aos dedos para que ele seja capaz, por exemplo, de abrir uma garrafa térmica Hirose and Ogawa (2007).

2.2 Componentes

Os robôs apresentam diversas partes, tais como: estrutura física, sistema de sensores, sistema de energia ou alimentação e sistema de controle. Estes sistemas constituintes nos

robôs possuem funções previamente definidas para atender as funcionalidades necessárias para suas aplicações. Assim, robôs podem ser construídos utilizando estruturas metálicas para realizar tarefas robustas ou estruturas plásticas para movimentos delicados ou com maior leveza. Do mesmo modo, robôs podem conter em suas estruturas sensores variáveis que percebem condições específicas como temperatura, presença, cor, gravidade, etc.

O sistema de alimentação e o sistema de controle são os mais importantes. Estes sistemas possibilitam ao robô associar o movimento e o processamento dos dados de entrada e saída via circuito eletrônico.

2.2.1 Microcontrolador PIC

Os microcontroladores são dispositivos eletrônicos que possuem em sua estrutura a integração de todos componentes necessários para o funcionamento de um processador, como memórias RAM, ROM e EEPROM, componentes lógicos e aritméticos usuais, periféricos de entrada e saída, conversores analógico/digitais (ADC), conversores digitais/analógicos (DAC), frequência de *clock*, entre outros. Um microcontrolador permite que os softwares de controle sejam embarcados diretamente em sua estrutura, isto é, tratando-se de um controlador embutido. Normalmente por apresentar uma velocidade de processamento muito inferior a um microprocessador, os microcontroladores são adequados para controlar pequenas tarefas em funções dedicadas, como o controle de eletrodomésticos e o controle de aplicações industriais. A **Figura 2.1** apresentam os componentes existentes na estrutura de um microcontrolador.

Outra vantagem ao se utilizar microcontroladores é o seu baixo consumo de energia, normalmente na casa dos miliwatts, devido a capacidade de entrar em modo de espera (*Sleep* ou *Wait*) no qual aguarda por uma interrupção ou evento externo, como o acionamento de uma tecla ou um sinal que chega via uma interface de dados.

O consumo destes microcontroladores em modo de espera pode chegar à casa dos nanowatts, tornando-os ideais em aplicações com baixo consumo de energia. Especificamente, este baixo consumo de energia é uma qualidade decisiva nos projetos de construção de protótipos de robôs bípedes. Neste projeto, optou-se pelo uso de um microcontrolador PIC 18F4550 com taxa de *clock* de 48 MHz e possui um consumo em repouso da ordem de 29 microwatts.

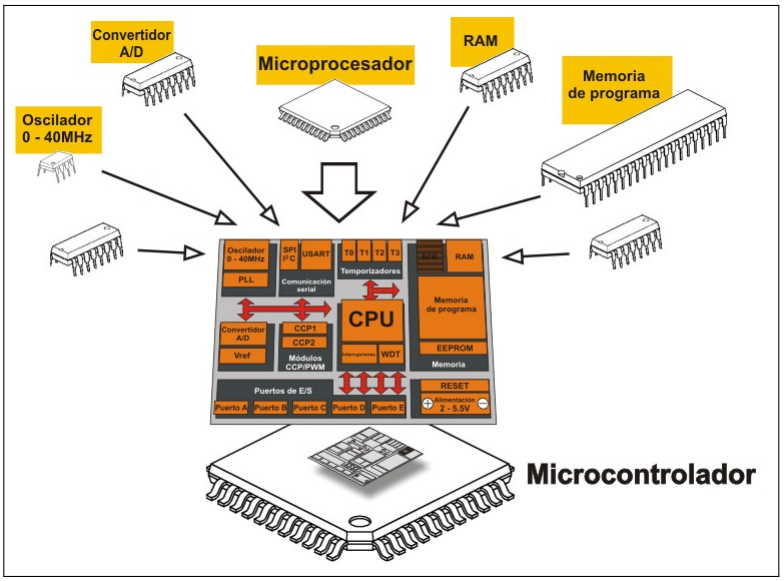

Figura 2.1: Componentes de um microcontrolador.
Fonte: MikroElektronika (2015).

2.2.2 Servo Atuador

Os servomotores são dispositivos compostos por um motor elétrico acoplado a um dispositivo dedicado de controle que permite o manejo preciso de sua rotação de 0 a +90 graus e de 0 a -90 graus, ou seja, até 180 graus, podendo chegar a 270 graus em alguns casos especiais dependendo do modelo.

Os servosmotores geralmente não propiciam velocidade ou potência compatível aos sistemas hidráulicos ou pneumáticos, mas apresentam as vantagens de terem mais precisão e exigirem menor espaço.

O funcionamento clássico de um servomotor corresponde a um sistema de volta fechado, ou seja, ele trava ao finalizar seu curso de meia volta. Isto ocorre porque no dispositivo de controle há um potenciômetro acoplado a uma engrenagem. A variação da resistência é proporcional à posição do braço do servo. Assim, a resistência é usada pelo dispositivo de controle eletrônico para determinar sua posição e do mesmo modo gerar um sinal de erro caso esta posição não corresponda à posição desejada, fazendo com que o motor retorne a posição inicial (zero grau).

Para o acionamento do servomotor são necessários 3 fios: o fio de alimentação com 5 Volts (normalmente de cor vermelha), o fio de controle que recebe ou 0 ou 5 Volts (normalmente de cor branca ou amarela) e o fio comum com 0 Volts ou GND (normalmente de cor preta). O controle da posição do braço do servomotor é realizada por meio de um

sistema conhecido pela sigla PWM, do inglês "*Pulse Width Modulation*" ou "Molulação por Largura de Pulso". Os servosmotores possuem diversas vantagens que justificam sua aplicação para a construção de protótipos de robôs de pequeno porte.

- Permitem controle preciso e eficiente;
- Envolvem estruturas simples e de fácil manutenção;
- Não requerem fonte de energia com alta tenção;
- São relativamente silenciosos e;
- Seu custo é relativamente baixo.

A **Figura 2.2** apresenta um modelo de servomotor e a **Figura 2.3** apresenta a representação de um sinal de controle por PWM, respectivamente.

Figura 2.2: Ilustração de um servomotor.

Fonte: PyroElectro.com (2012)

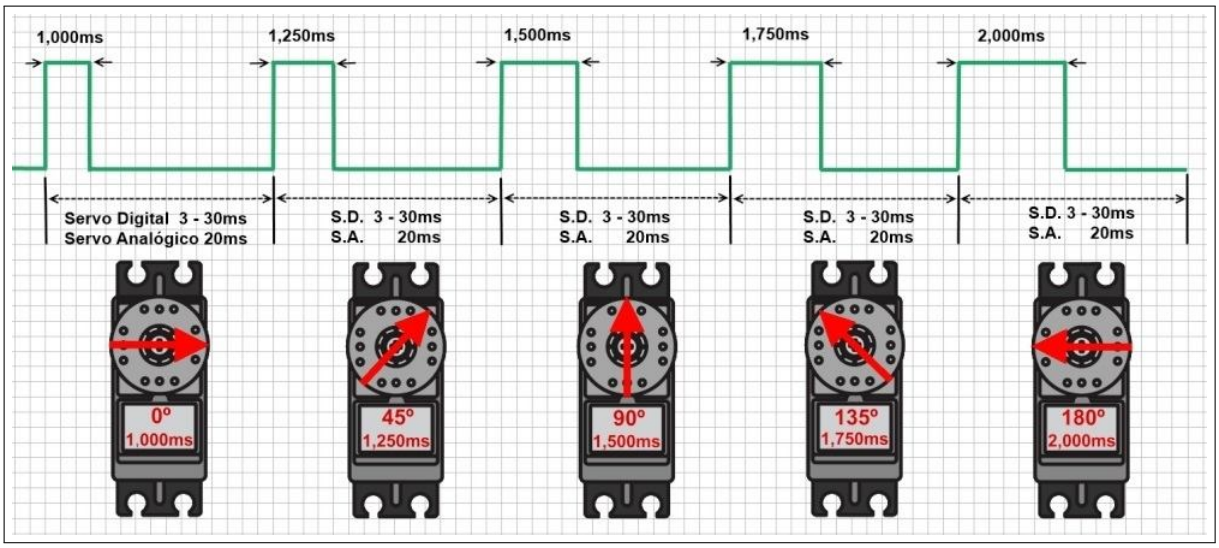

Figura 2.3: Ilustração do sinal de controle dos servomotores por "PWM" .
Imagem desenvolvida conforme referência: Futaba (2015).

2.2.3 Modelos Bípedes

Diversos centros de pesquisa espalhados pelo mundo têm efetuado estudos e desenvolvidos métodos para a construção de robôs bípedes para as mais diversas finalidades (Iida et al., 2009).

Um fator limitante a este projeto é o custo associado à aquisição de peças e componentes. O fato do seres humanos possuírem grande variedade de movimentos em seus membros, tanto inferiores quanto superiores, faz com que esta reprodução do movimentos em um robô bípede torne a execução do projeto de alto custo.

Alguns modelos existentes no mercado comercial e seus respectivos custos são apresentados nas **Figuras 2.4** e **2.5**, respectivamente.

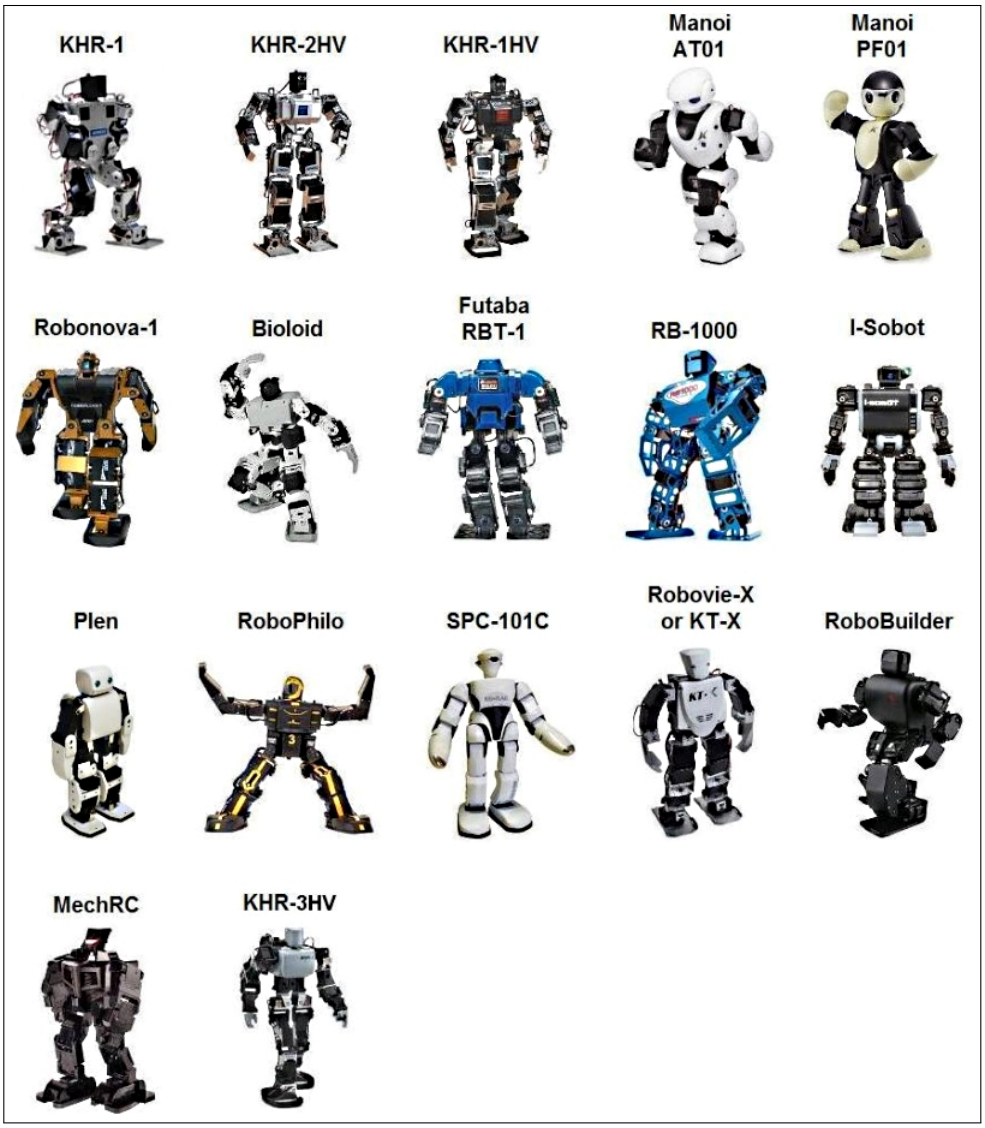

Figura 2.4: Modelos de diversos robôs bípedes e seus respectivos nomes. Fonte: LimoncelloDigital (2011).

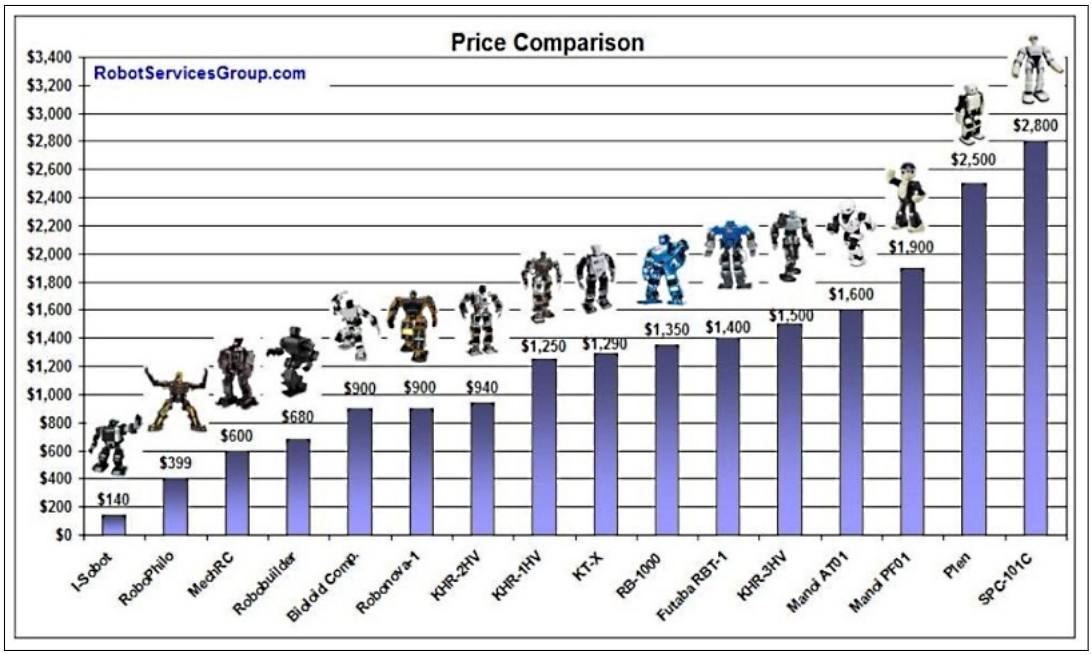

Figura 2.5: Representação da escala de valores dos robôs bípedes médios.
Fonte: LimoncelloDigital (2011).

A **Figura 2.6** mostra o robô Bioloid, o qual foi utilizado como referência para a definição ao tamanho médio do Robô Bípede, em relação aos servos utilizados. O robô Bioloid tem um projeto que apresenta muita agilidade nos movimentos, bem como uma aparência biológica mais amigável.

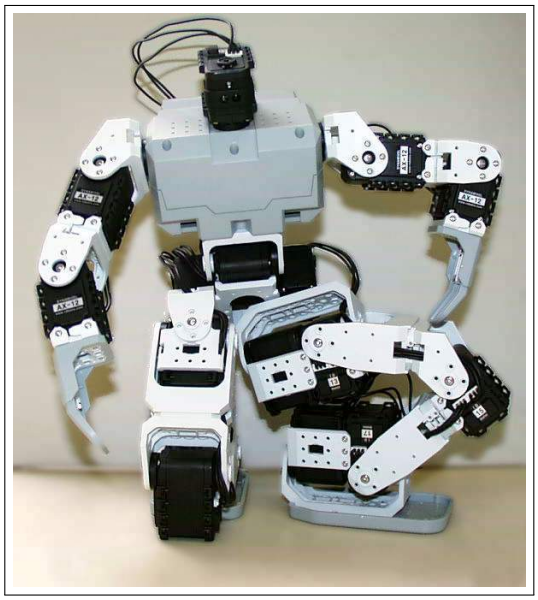

Figura 2.6: Robô Bioloid utilizado como referência.
Fonte: TheOldRobotsWebSite (2009).

As **Figuras 2.7** e **2.8** apresentam as peças de montagem do modelo desenvolvido. É importante observar o posicionamento dos quinze servomotores e as peças correspondentes da estrutura e do suporte, bem como os eixos e seus respectivos rolamentos destinados a proporcionar os graus de liberdade para promover os movimentos do robô.

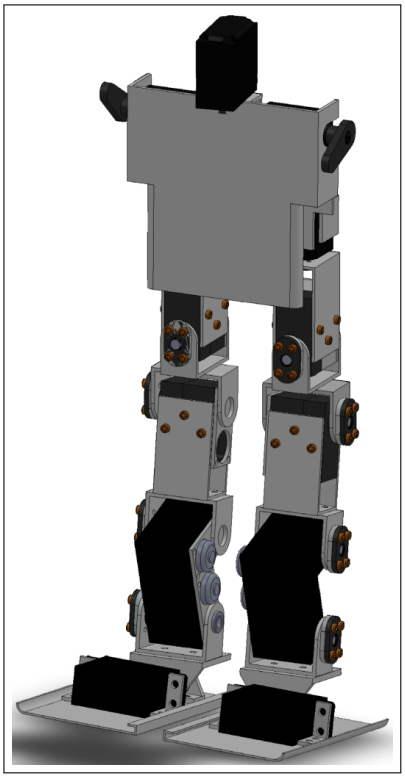

Figura 2.7: Disposição dos servomotores, dos eixos e das peças utilizadas no protótipo do Robô Bípede desenvolvido.

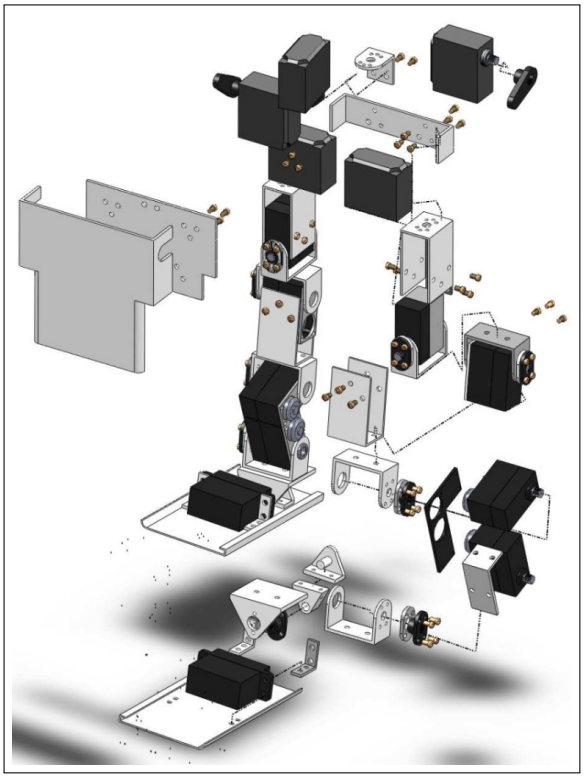

Figura 2.8: Vista explodida das peças utilizadas no Robô Bípede desenvolvido.

Definidos a estrutura e o modelo mecânico, podemos efetivamente seguir para construção do bípede e definições de seus movimentos. No capítulo 3 são escolhidos os componentes, bem como a adaptação dos materiais do lixo tecnológico disponíveis e todos os processos de usinagem das peças e montagem. Também serão serão abordados os componentes periféricos e auxiliares, tais como circuito de teste para servos, fonte de alimentação e bateria.

Capítulo 3

Projeto Eletromecânico

3.1 Movimentos Iniciais do Robô Bípede

Definir os processos de movimento antes de concretizar a montagem física do sistema bípede, permite avaliar possíveis adequações estruturais caso haja necessidade. Os valores iniciais e finais para cada posição de máximo e mínimo dos 15 motores foram utilizados como terminadores de estado para os movimentos de cada membro que foram ajustados empiricamente. Também ajustamos empiricamente as funções contidas nas bibliotecas "servo.h", "pacote_USB_direto.h" e a "função executa_cenario_memoria_local()" que atuam nos servomotores a fim de realizar os movimentos desejados.

Após definições de valores iniciais de movimentação, que foram incluídas na biblioteca "servo_init.h", também foi estabelecidos os tempos de acionamento do PWM para cada servomotor. A velocidade de acionamento e o modo de movimento do robô que pudesse ser mais suave e natural possível. Por este motivo, para a composição dos movimentos de caminhar, há necessidade de acionamento conjunto de mais de um motor simultaneamente.

A **Figura 3.1**, ilustrada por Silva (2001), procura demonstrar as fases de apoio e oscilação da perna direita e esquerda. Estas fases irão formar os pontos estratégicos dos movimentos empíricos, que irão compor os acionamentos conjuntos dos servomotores para definir o movimento do robô bípede. O sistema eletrônico do robô humanóide deve ser capaz de controlar os motores do robô de modo completamente autônomo. O sistema eletrônico deve ainda possuir todo o software embarcado que permita o robô caminhar por tempo razoável.

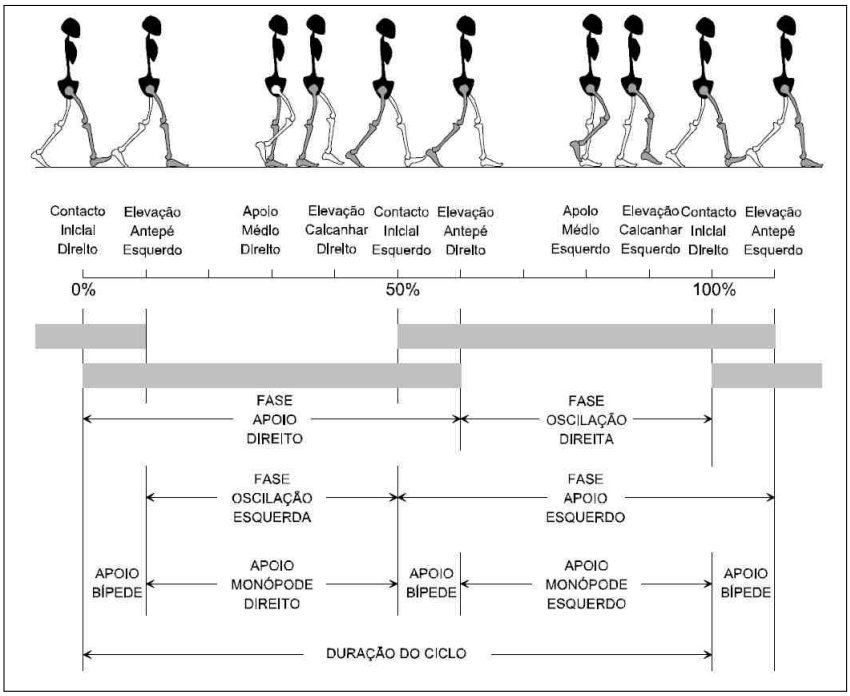

Figura 3.1: Diagrama de tempo de acionamento dos motores para os movimentos iniciais do Robô Bípede.

Fonte: Silva (2001).

3.2 Construção Mecânica do Bípede

Como já relatado no capítulo 2, por menor que sejam os robôs humanóides os investimentos geralmente serão altos. Entretanto, na montagem deste protótipo de robô bípede foram utilizados, dentro do possível, peças e partes de materiais reciclados ou em desuso com a finalidade de redução do custo de montagem. Dentre elas, utilizou-se rolamentos de drives de disquete 5 1/4" e CD-ROM, retalhos de placas e perfis de alumínio entre outros materiais.

O uso dos perfis de alumínio, reciclados a partir de ferro velho, teve por finalidade utilizar as características mais relevantes deste metal, tais como: sua leveza, sua boa resistência mecânica, sua maleabilidade, entre outros, atendendo as necessidades do protótipo e a facilidade na montagem. O uso do alumínio permitiu a redução das dimensões das peças de modo a reduzir o peso no modelo final. Além disso, optou-se por utilizar um número reduzido de peças diferentes, tanto para simplificar o projeto, quanto para minimizar a força de torque exercida pelos servos motores. A **Figura 3.2** apresenta algumas das etapas de montagem dos pés e tornozelos do robô bípede.

Na **Figura 3.2 (a)** visualiza-se os servos adquiridos mediante importação para a redução do custo de aquisição. Fatores relevantes, para a estética e esforço do motor são os ajustes das dimensões mecânicas, testes eletroeletrônicos incertos, a variação do centro de massa das peças e partes que foram considerados no desenvolver do projeto de construção do protótipo.

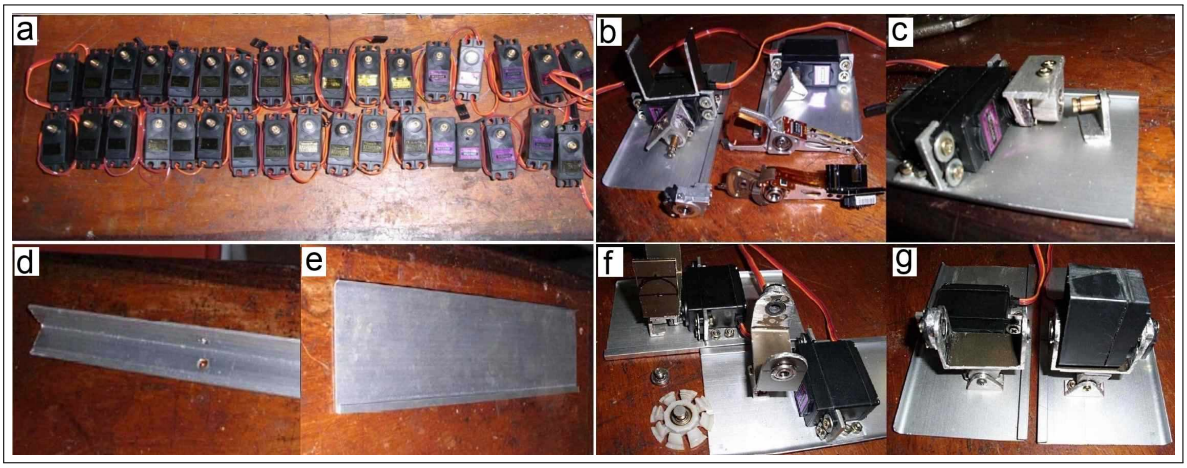

Figura 3.2: Ilustração dos pés, tornozelos e componentes.

Pode-se observar também que para a base dos pés, foram utilizados um apagador de quadro branco e uma cantoneira de suporte de isoladores de cerca elétrica. Ainda ressalta a utilização de rolamentos adaptados junto aos tornozelos, proveniente do cabeçote de leitura de dois discos rígidos.

Na **Figura 3.3** é apresentada algumas das etapas do processos de montagem das juntas utilizando peças reutilizadas de driver de disquete 5 1/4, um perfil de alumínio utilizado como suporte e um dissipador para transistor de uma televisão utilizado como espaçador da distancia do eixo do servomotor.

Dentre as etapas de montagem, certamente a que apresentou maior dificuldade devido ao número de articulações foi o joelho. A **Figura 3.4** apresenta as modificações necessárias para montagem desta etapa. É um um fator limitante a largura e a altura desta peça em relação a proporção desejada sem a alteração significativa do centro de massa do robô. Para esta adequação, as tampas originais dos servomotores foram removidas e, manualmente desenvolvidas com a finalidade de se reduzir o tamanho total do conjunto. Em decorrência desta alteração nas dimensões do servomotor, houve também a necessidade de realocar a placa eletrônica de controle local dos servomotores e criar uma isolação elétrica da mesma. Nesta etapa foram utilizados 2 servomotores na montagem de cada joelho.

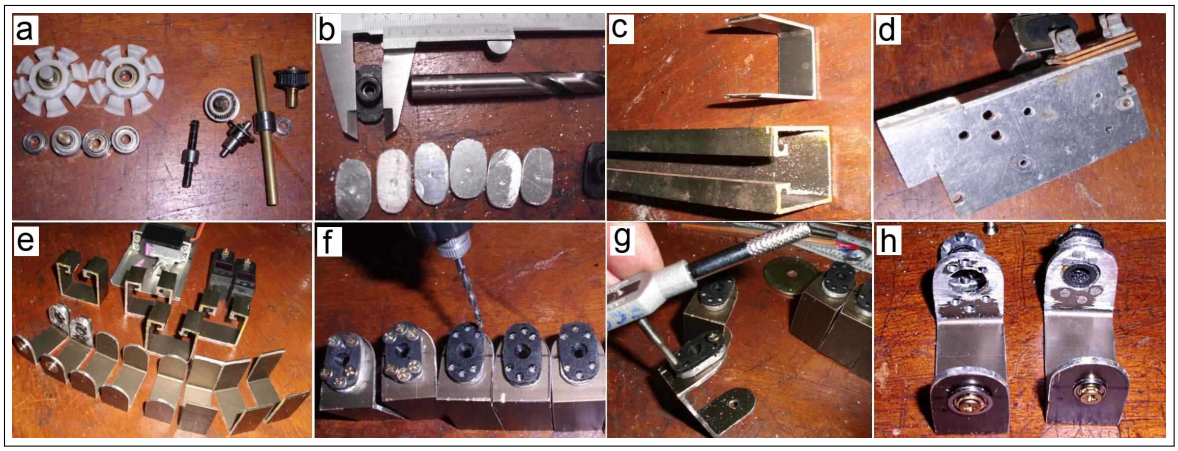

Figura 3.3: Ilustração das juntas e componentes.

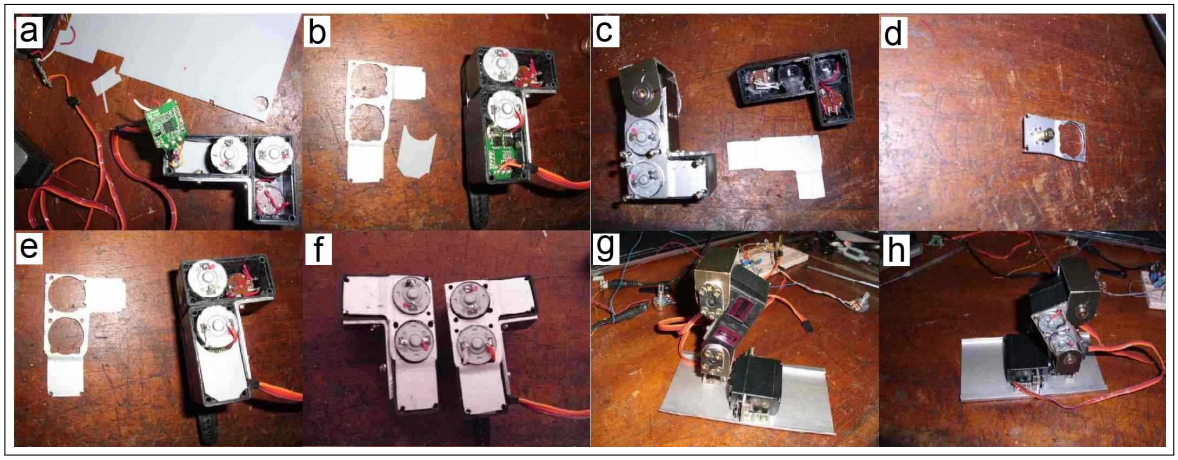

Figura 3.4: Ilustração dos joelhos e componentes.

A **Figura 3.5** apresenta as etapas de montagem da perna (coxa) e do tronco do robô e da cabeça. Utilizou uma pá de alumínio de um ventilador de teto como a proteção peitoral e um dissipador de calor de um aparelho amplificador. A proteção de pá de alumínio teve como finalidade de abrigar as baterias de polímero de íon lítio (Li-Po) e a chapa do dissipador de calor utilizado para a fixação das demais partes junto ao tórax do robô. Na região do tronco foi utilizado outra chapa de alumínio proveniente de um dissipador de calor de transistor de um aparelho amplificador, o qual foi utilizado na fixação dos demais servos do tronco e, a cabeça foi fixada por um pedaço de cantoneira de alumínio.

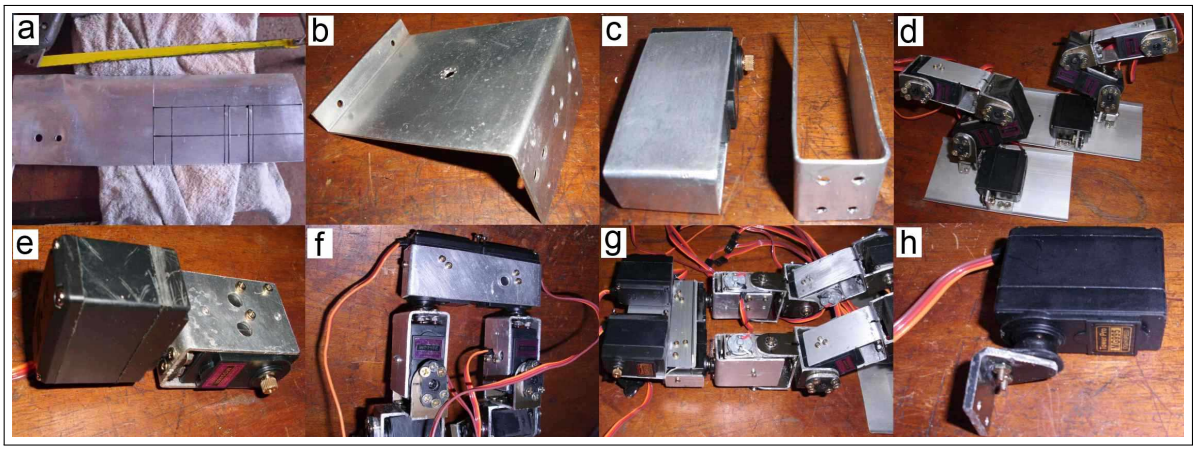

Figura 3.5: Ilustração das coxas, cintura, peito, cabeça e componentes.

As **Figura 3.6** e **3.7** respectivamente apresentam a comparação entre modelagem em CAD e a ilustração do projeto mecânico do robô bípede depois de concluído.

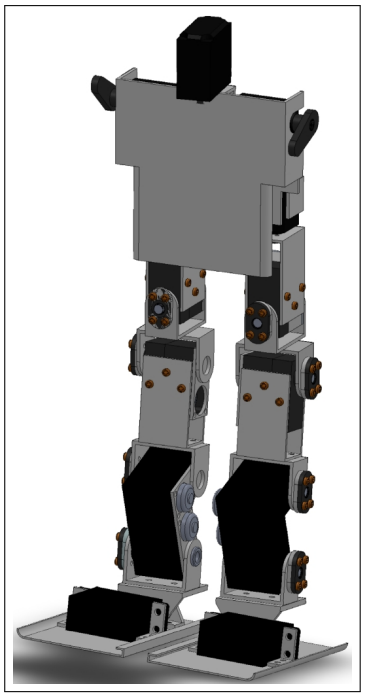

Figura 3.6: Modelagem em SolidWorks do Robo Bípede

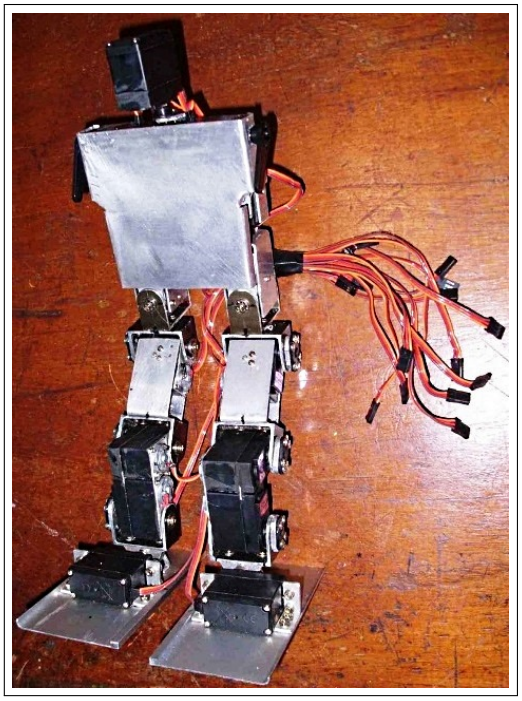

Figura 3.7: Projeto mecânico do Robô Bípede concluído

As medidas e detalhes de cada peça do robô bípede é apresentado na **Figura 3.8**.

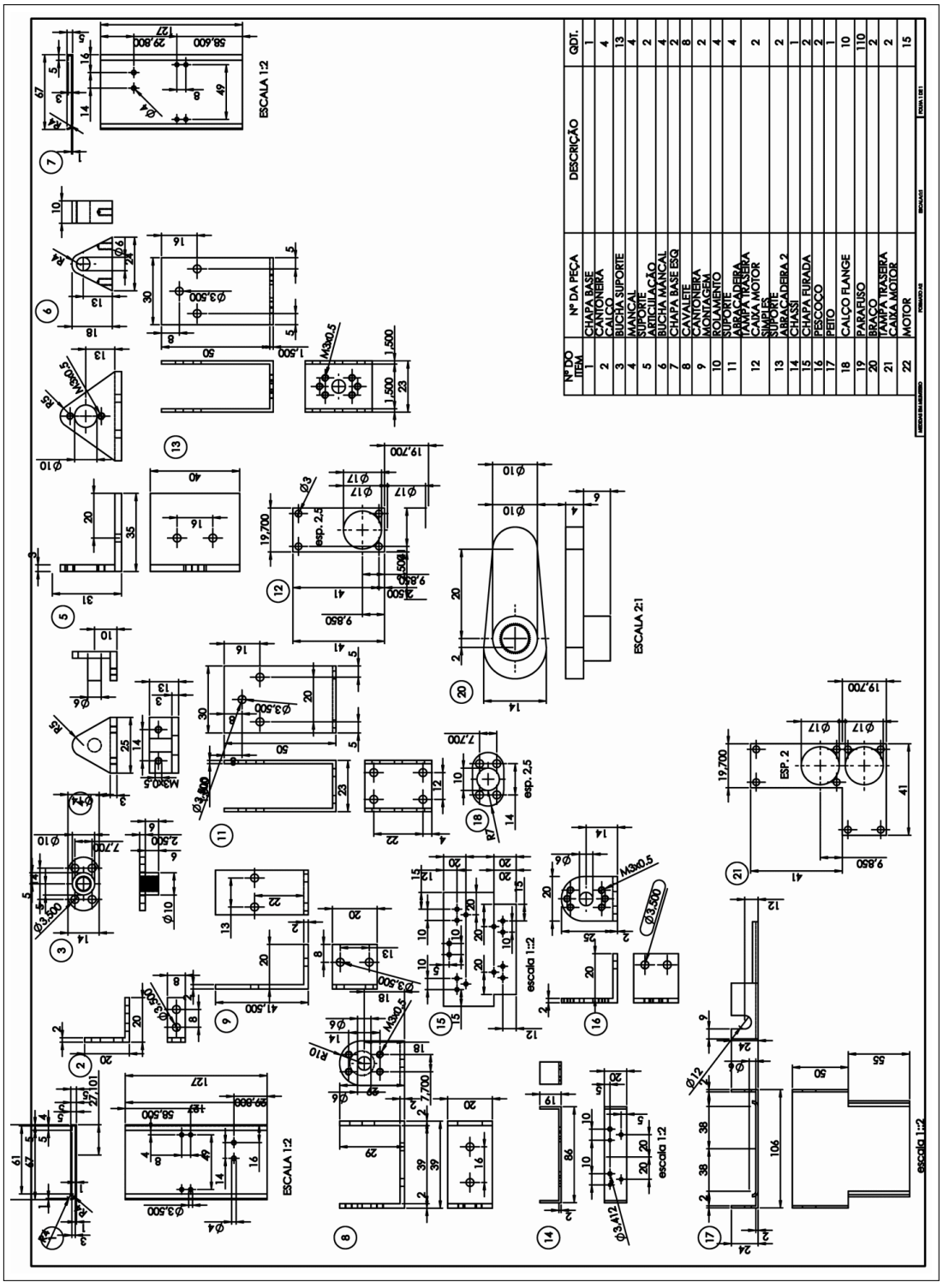

Figura 3.8: Modelagem em SolidWorks.

3.3 Gerador de Sinal e Teste para o Servomotor

Durante o desenvolvimento do projeto mecânico foi necessário verificar o posicionamento e as limitações de movimento de cada servomotor. Considerando o movimento máximo de 180 graus, procurou-se obter o melhor aproveitamento do ângulo entre os limites extremos de cada eixo.

Para verificar os ângulos é necessário gerar pulsos elétricos com intervalos precisos e ajustáveis para comandar os movimentos dos servos. Embora os pulsos estão dentro de um intervalo previamente estudado, não era conhecida a precisão dos valores para os servos adquiridos. Avaliou-se que alguns microsegundos poderiam levar o servo a movimentar fora dos limites mecânico de 0 a 180 graus das travas limitadoras podendo ocasionar a destruição das engrenagens ou mecanismos internos. Gerar os pulsos com microcontroladores é muito difícil dado que são monotarefas e dependeriam de um *timer* interno para controlar os pulsos com precisão. Por este motivo é importante saber os valores corretos dos intervalos de tempo.

Para resolver o problema foi desenvolvido o circuito auxiliar apresentado na **Figura 3.9**, adaptado do circuito de sirene descrito por McCOMB (2001). O circuito permitiu o controle por botões de forma linear e verificar no osciloscópio os pulsos gerados.

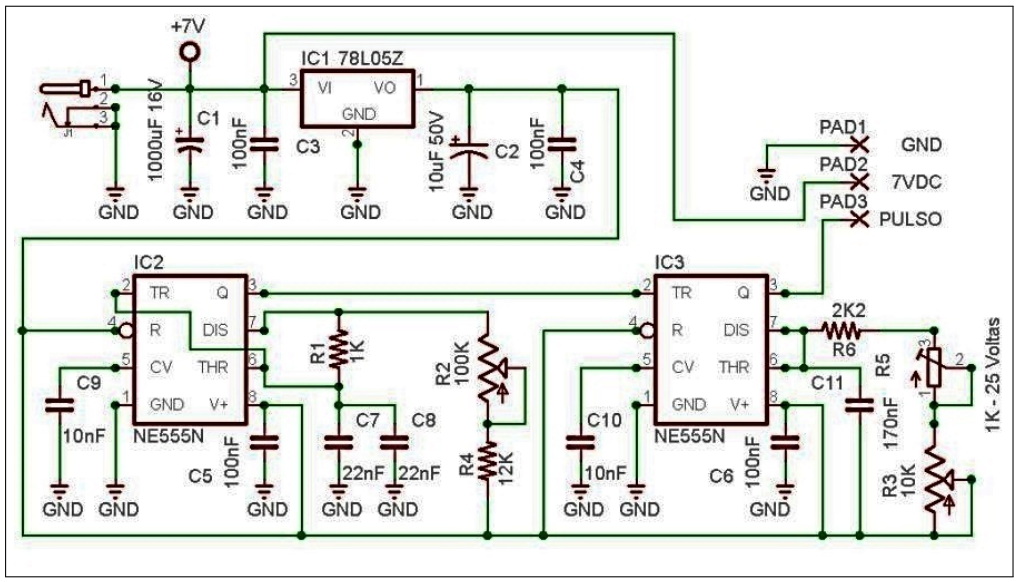

Figura 3.9: Circuito do servo controle.

Na **Figura 3.9** temos o "IC1" responsável por estabilizar a tensão em 5 volts. O "IC2" gera o tempo dos pulsos ativos do PWM denominado "T1" e o ajuste do tempo é realizado

pelo botão "R2". Quanto ao "IC3" vai definir o tempo inativo do PWM denominado de "T2" e ajustado pelo botão "R3". As **Figuras 3.10** e a **3.11** mostram, respectivamente, a montagem da placa eletrônica e suas trilhas.

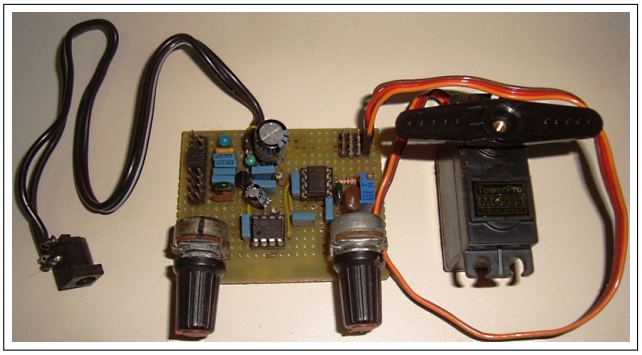

Figura 3.10: Placa do servo controle montado.

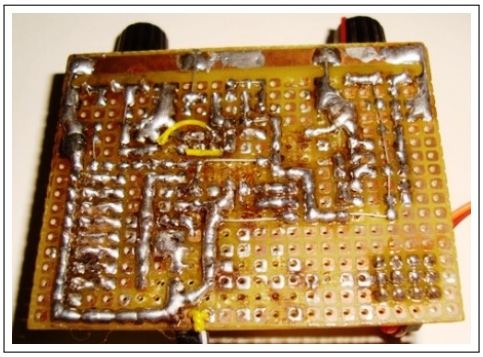

Figura 3.11: Trilhas da placa eletrônica do servo controle.

Na **Figura 3.12** temos o sinal produzido pelo servo controle no osciloscópio, sendo que os dois botões giratórios permitem o ajuste do tempo "T1"e "T2" respectivamente, para ajustar o PWM dos servomotores.

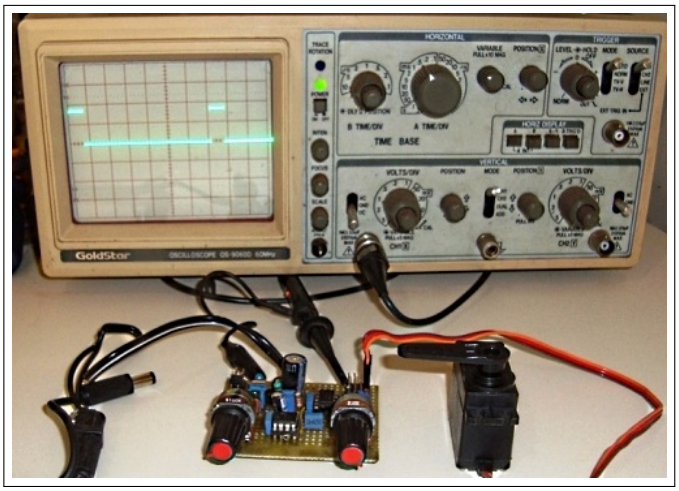

Figura 3.12: Sinal produzido pela placa servo controle no osciloscópio.

Observa-se que a variação do tempo "T2" entre 4,7 a 10,92 milissegundos não influencia no posicionamento, porém o aumento do tempo de "T1" entre 0,723 a 2,4 milissegundos levou o servo mudar de posição 0 ° para posição 180 ° conforme mostrado nas **Figuras 3.13**, **3.14**, **3.15** e **3.16**.

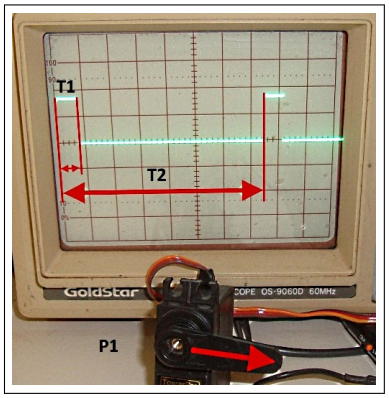

Figura 3.13: Sinal do servo, T1 valendo 0,723 ms para posição 0° e T2 em 10,92 ms.

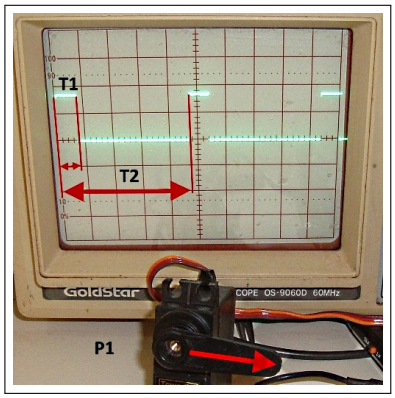

Figura 3.14: Sinal do servo, T1 valendo 0,723 ms para posição 0° e T2 em 4,7 ms.

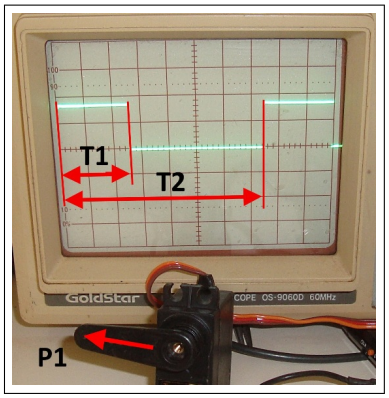

Figura 3.15: Sinal do servo, T1 valendo 2,4 ms para posição 180° e T2 em 10,92 ms.

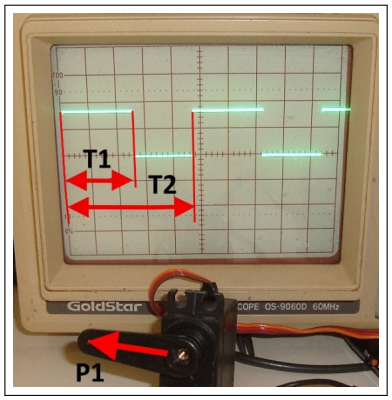

Figura 3.16: Sinal do servo, T1 valendo 2,4 ms para posição 180° e T2 em 4,7 ms.

3.4 Conversor - Fonte de Alimentação

Uma vez que os servomotores são constituídos por um motor C.C., seu comportamento elétrico característico influenciará no circuito. Devido a composição do induzido ser uma bobina, esta possuirá uma indutância L que irá-se opor a passagem da corrente elétrica variável, por meio do fenômeno de reatância indutiva XL.

A reatância indutiva está diretamente relacionada a variação da corrente. Caso seja uma corrente senoidal, ela é definida conforme a **Equação 3.1**. Sendo 2π a constante

da variação senoidal e F a frequência da corrente elétrica, conforme Villate (2012).

$$XL = 2\pi \times F \times L \tag{3.1}$$

Contudo os servomotores não são comandados por correntes senoidais, seu controle é realizado por pulsos de alta frequência em onda quadrada variável (PWM). Considera-se a reatância indutiva consequência da força contra eletromotriz f.c.e.m. denotado por ε. Esta tensão contrária é produzida pela variação magnética de auto indução, ocasionada no momento da energização da bobina L. Deste modo ao aplicar uma tensão da fonte sobre uma bobina, esta produzirá uma tensão com polaridade igual a da fonte impedindo a passagem da corrente, o que provoca o efeito da reatância indutiva XL. Com o passar do tempo o campo se estabiliza reduzindo a f.c.e.m. e consequentemente a corrente elétrica irá aumentar.

Ao se retirar a tensão aplicada sobre o indutor este produzirá um efeito análogo, porém inverso. Com a abertura da chave haverá um rápido isolamento elétrico nos contatos na chave reduzindo corrente do indutor. Neste momento o fluxo magnético que é dependente da corrente irá consequentemente variar, induzindo uma força eletromotriz f.e.m de polaridade oposta a fonte, o que eleva o potencial e, em alguns casos, rompe o isolamento elétrico da chave. Este potencial é tão elevado quanto a razão da variação de abertura da chave, conforme mostra a **Equação 3.2**.

$$\varepsilon = -L \cdot \frac{\Delta i}{\Delta t} \tag{3.2}$$

Onde Δi representa a variação da corrente em função do inverso da variação do tempo Δt, que multiplicado pela indutância L definirá o valor da f.e.m representada por ε, descrito por Villate (2012).

Esta força contra eletro motriz f.c.e.m estará presente em cada motor, comandado por PWM. A interação entre os servomotores ocasionará pertubações nos circuitos eletrônicos que estiverem interconectados. Segue o exemplo de ruído para um motor comandado por PWM na **Figura 3.17**. Primeiro podemos observar a tensão sem interferências em modo PWM, quando não há um motor conectado, conforme ilustra a **Figura 3.17(a)**. Posteriormente temos os picos de tensão adicionados, devido ao efeito da f.c.e.m. quando o motor foi conectado ao PWM, como exibido na **Figura 3.17(b)**.

A **Figura 3.17(c)** mostra uma aproximação para melhor observar o ruído apresentado na **Figura (b)**. A **Figura (d)** exibe a comparação da tensão do PWM com relação a f.e.m., este caso específico, estudado por Banakara (2013), a tensão do PWM era de 420 Volts, mas a f.e.m. acrescentou um pulso de 1000 Volts nos terminais do motor. Esta instabilidade elétrica é conhecida como "Interferência Eletromagnética" (EMI). A fim de minimizar este efeitos, deve-se levar em consideração um projeto adequado de placa ou circuito, filtros de "EMI" e fonte apropriada.

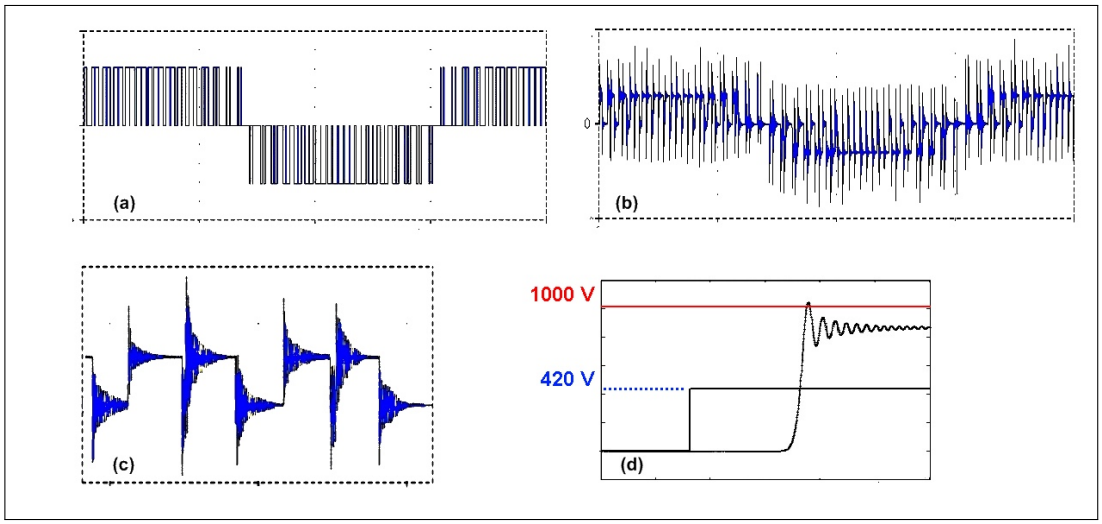

Figura 3.17: (a) Tensão modo PWM com motor desconectado, (b) Tensão do PWM com o motor conectado, (c) Detalhe da interferência sobre a tensão de PWM, (d) Valor da f.e.m. em comparação ao PWM.

Imagem modificada de Banakara (2013).

Os pulsos no sentido horário e anti-horário com EMI são verificados diretamente nos terminais do motor para um dos servomotores. Além disso, também ocorrem pulsos com EMI na alimentação com bateria em aproximadamente 7 volts. Estes sinais com EMI são respectivamente apresentados nas **Figuras 3.18** e **3.19** para o sentido horário e anti-horário do giro do servomotor utilizado.

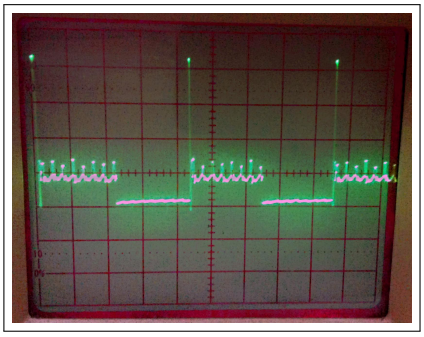

Figura 3.18: Sinal do PWM gerado nos terminais do motor em sentido horário com os pulsos da f.e.m.

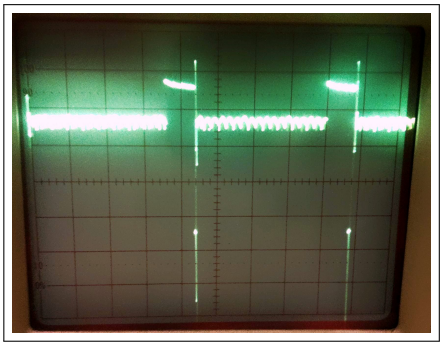

Figura 3.19: Sinal do PWM gerado nos terminais do motor em sentido anti-horário com os pulsos da f.e.m.

Observe que os pulsos de EMI atingem valores bem maiores que a tensão nominal da bateria em PWM. Outra característica importante dos motores C.C. é a elevada corrente de partida. Quanto maior for o troque, maior será a corrente inicial ou de funcionamento. Na **Figura 3.20** temos a ondulação da tensão da bateria, devido a corrente de partida em mistura com a EMI, em um único servomotor. Um motor com carga baixa próximo a 10 porcento da sua capacidade máxima de 15 Kgf/cm, provocou uma ondulação de aproximadamente 0,3 Volts.

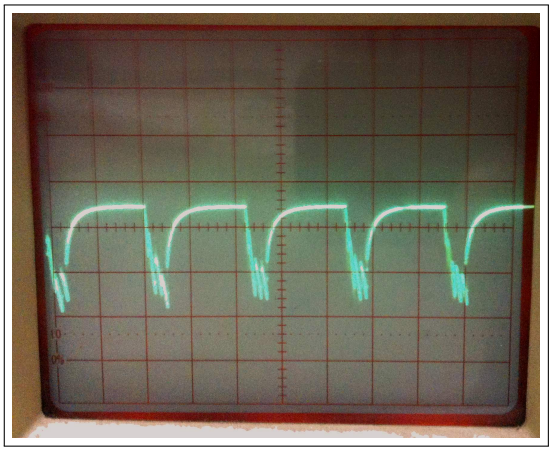

Figura 3.20: Ondulação na tensão da bateria devido a corrente de funcionamento do servomotor.

Devido a complexidade dos sistemas eletrônicos e a percepção da existência de ruídos elétricos capazes de queimar os microcontroladores e os circuitos do dispositivo de controle dos servomotores, foi necessário desenvolver uma fonte de corrente contínua capaz de suportar as interferências eletromagnéticas provenientes dos servomotores.

O controle conjunto dos diversos aturadores podem induzir ruídos que resultam na ativa a proteção da fonte, movimentos inesperados ou mesmo danificando permanentemente o microcontrolador. Deste modo, uma fonte de baixo custo foi desenvolvida reutilizando parte de uma fonte chaveada marca “Dr. Hank”, 250W, modelo “PX/PW-250W” ilustrada na **Figura 3.21**.

Figura 3.21: Fonte PC utilizada para modificação.

Apesar do tamanho reduzido, o acionamento dos servomotores pode consumir uma corrente inicial ou de pico da ordem de 3 Ampères em uma fração de segundos (microssegundos). Este comportamento, na média de funcionamento, possui uma razão definida da tensão pela carga, resultando uma corrente próxima a 1 Ampère.

De modo geral, ao analisar a corrente de 1 Ampère isoladamente considera-se pequena. Entretanto, quando se considera que o valor da corrente é acumulativa e se verifica a necessidade de usar quinze motores na construção do protótipo deste projeto, a corrente de pico resultante em poucos microssegundos pode chegar a assustadora ordem de 45 Ampères. Para fins de comparação, esta corrente é maior que os 40 Ampères do chuveiro elétrico residencial.

Considerando estas características elétricas do acionamento e controle dos servomotores, tem-se um desafio quanto ao desenvolvimento da fonte de alimentação e a escolha da bateria correspondente.

Os servomotores consomem mais potência conforme o aumento da tensão de alimentação. Eles trabalham variando a tensão dentro da faixa de 4.5 Volts a 7.2 Volts, contudo toleram valores de até 8.4 Volts por um período curto de tempo; não recomendado pelo fabricante ultrapassar 7.5 Volts.

A característica mínima da fonte para o projeto deverá fornecer 7 Volts com ótima estabilidade e suportar corrente constante até 15 Ampères e um pulso inicial de corrente da ordem de 40 Ampères.

Comercialmente não foi encontrada fonte com estas especificações. Por causa disto adaptou-se um circuito *Step-Dow* (conversor DC/DC abaixador) que permite converter os 12 Volts para 7 Volts com alta eficiência e elevar a corrente nominalmente de 17 Ampères para próximo de 29 Ampères. Foram adicionados outros circuitos contra alta e baixa tensão a fim de proteger os servomotores contra a queima, quando operado fora do

especificado.

Com a ajuda de grandes bancos de capacitores que vão armazenar e suprir corrente nos picos instantâneos, aliado ao consumo de energia nos fios e indutores do circuito, que limitarão o picos instantâneos de corrente, e é possível permitir que a fonte do computador consiga suprir a energia do robô bípede.

A fonte desenvolvida trabalha em 7 Volts e sua corrente pode permanecer continuamente em 10 Ampères sem redução no nível de tensão. A partir deste valor a fonte reduz gradativamente a tensão para uns 5 Volts até uns 15 Ampères para amortizar as corrente de partida e admite pulsos instantâneos maiores que 20 Ampères. Entretanto, se houver uma continuidade desta corrente, a tensão será reduzida até a corrente se estabilizar em valores bem menores para a proteção da fonte.

Durante o desenvolvimento da fonte foi necessário realizar testes de carga. Para estes testes, foram utilizadas lâmpadas halogenas especiais de 24 Volts e 150 Watts. Essas lâmpadas se encaixaram perfeitamente aos testes porque possuem uma corrente inicial muito alta, e após acesa consomem cerca de 2,5 Ampères.

Ao se associar quatro lâmpadas em paralelo obtém-se um pulso perto dos 30 Ampères e posteriormente o brilho é estabilizado. Este estado permanece em 10 Ampères aproximadamente, característica esta muito semelhante ao funcionamento dos servomotores na condições de maior consumo de corrente.

Após a criação de diversos projetos muitos testes e calibrações, conseguiu-se manter a tensão estável desde a sua partida até as lâmpadas acenderem normalmente. A **Figura 3.22** mostra as lâmpadas acessas durante os testes de desenvolvimento.

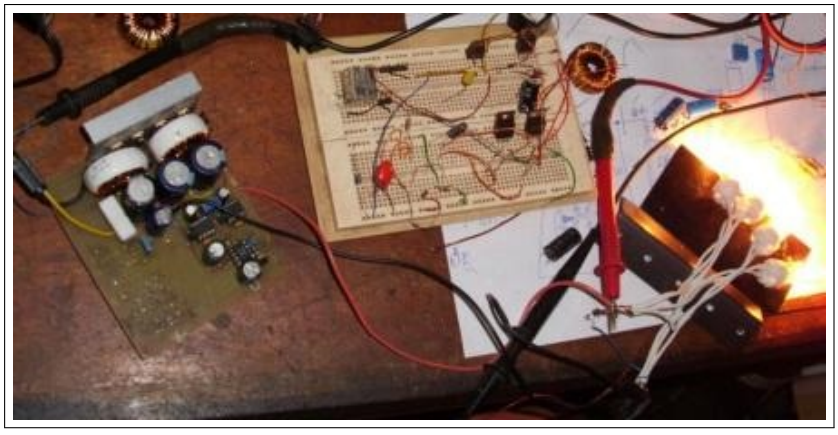

Figura 3.22: Placa da fonte em desenvolvimento - Teste de carga.

Estudos prévios com uma fonte de computador comercial acarretou o desligamento imediato da fonte quando apenas duas lâmpadas halógenas foram conectadas diretamente

aos 12 Volts de entrada. Por outro lado, estudos realizados em uma fonte de computador incrementada com o circuito *Step-Daw* permitiu ligar simultaneamente até cinco lâmpadas halógenas sem desarmar a proteção.

Neste projeto, não estava previsto como objetivo principal a construção de fonte de corrente contínua de 7 Volts e 15 Ampère. Assim sendo não foi realizado o desenvolvimento detalhado do circuito, bem como o estudo do projeto existente da fonte utilizada como base para a modificação. Desta forma, uma ilustração dos componentes utilizados para a incrementação da fonte chaveada é apresentada nas **Figuras 3.23**, **3.24** e **3.25**.

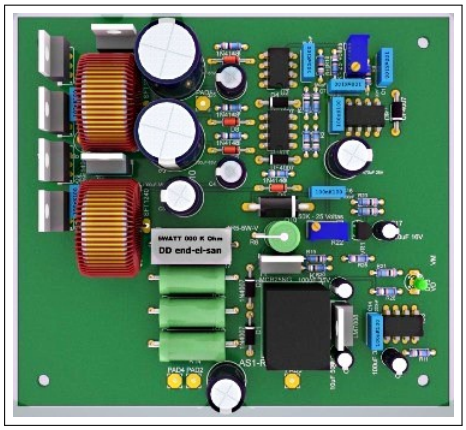

Figura 3.23: Placa fonte - Projeto em CAD vista dos compomentes.

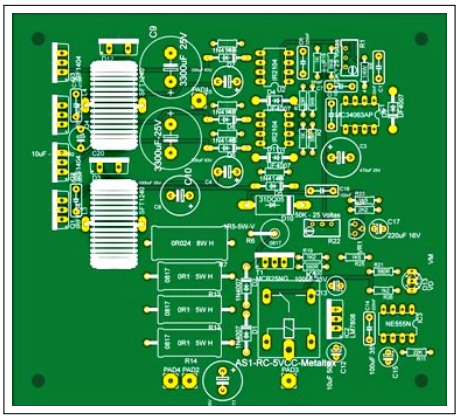

Figura 3.24: Placa fonte - Projeto em CAD vista do leiaute superior.

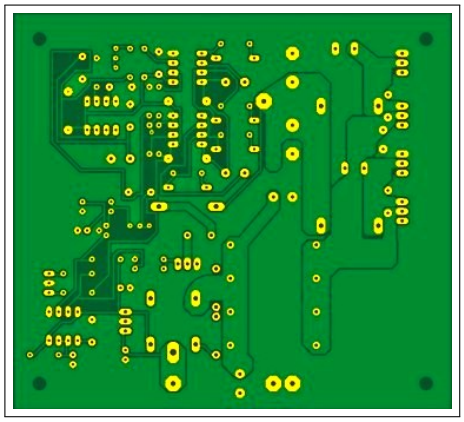

Figura 3.25: Placa fonte - Projeto em CAD vista do leiaute inferior.

Para o término da montagem da placa utilizou-se o método "*Manhattan-style*" demos-

trado nas **Figuras 3.26** e **3.27**.

Para a estabilização do circuito da fonte e seus componentes optou-se pelo uso de aterramento (fio comum) independente para cada componente do sistema, conforme ilustrado na **Figura 3.27**.

Figura 3.26: Placa fonte - Componentes.

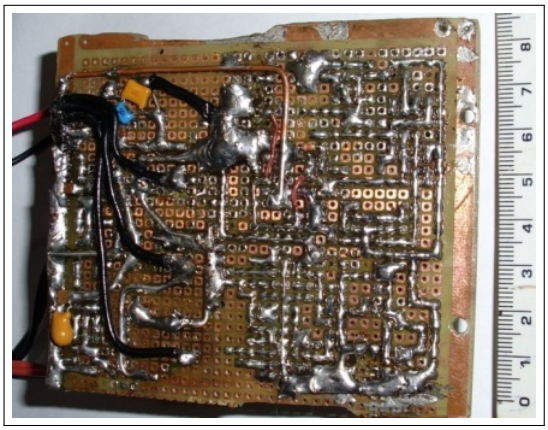

Figura 3.27: Placa fonte - Circuito *"Manhattan-style"*.

A fonte desenvolvida inclui ainda saídas extras estabilizadas de tensão independentes da saída principal (7 Volts e 15 Ampère). Este procedimento teve o objetivo de evitar possíveis interferências no processador. Foi montado um circuito independente utilizando um regulador por meio do circuito integrado (conhecido como regulador monolítico) "LM7808" com propósito de fornecer 8 volts e 1 Ampère estabilizado para alimentar o circuito regulador de tensão exclusivo do processador.

Também utilizou um segundo regular monolítico com o circuito integrado "LM7805" para fornecer tensão independente regulada de 5 volts para testes de circuitos auxiliares, conforme mostra as **Figuras 3.28** e **3.29** . As etapas de montagem dos elementos da fonte na **Figura 3.30**.

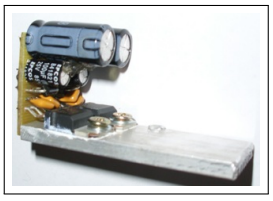

Figura 3.28: Placa fonte - Regulador monolítico.

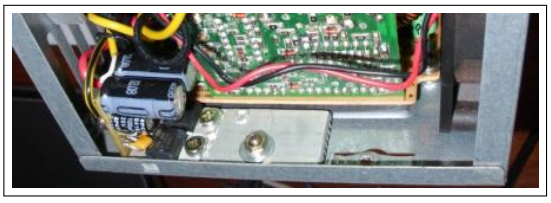

Figura 3.29: Placa fonte - Regulador monolítico instalado.

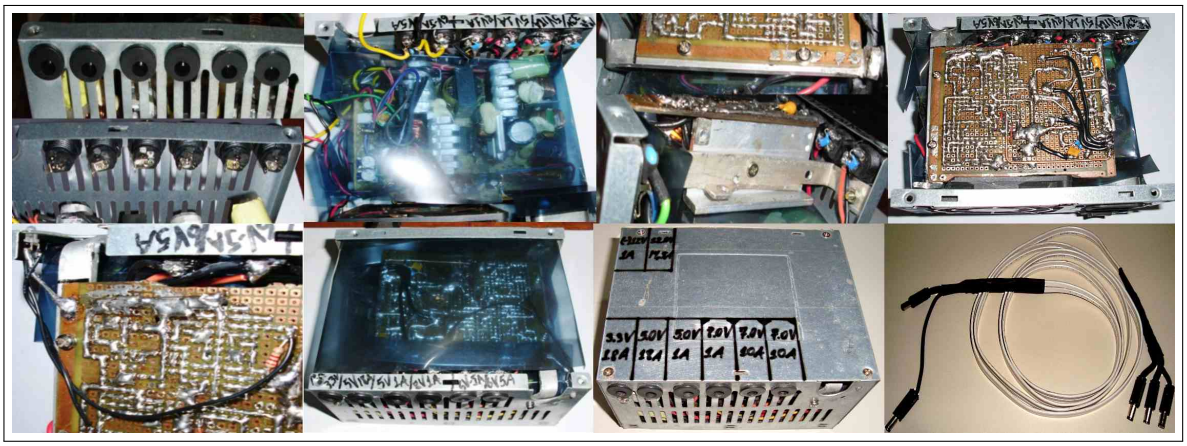

Figura 3.30: Ilustração dos alguns passos de montagem fonte.

De forma semelhante à que foi procedida com a construção da fonte de alimentação, a escolha de uma bateria interna não esteve isenta dos problemas de interferências e quedas de energia.

Graças as novas tecnologias de bateria disponíveis no mercado, tais como a bateria de *Íon de Lítio*, que possuem excelente relação peso e energia.

A bateria de *Íon Lítio* possui como limite inferior a tensão de 3,6 Volts (quando descarregada) e o limite superior de 4,2 Volts (quando totalmente carregada). Esta bateria é extremamente sensível aos limites superior e inferior, os quais se ultrapassados podem danificar permanentemente esta bateria.

Por se tratar de uma novidade no mercado nacional, as baterias de *Íon lítio* possuem um custo muito elevado para aquisição. Outra variante para a bateria de lítio é a bateria de *Polímero de Íon Lítio* (Li-Po) ilustrada na **Figura 3.31**.

Estas baterias tem a mesma capacidade de carga e característica de tensão, com a diferença conseguir entregar um pulso de corrente instantâneo elevado. Este pulso é expresso em "Coulomb" (C) e a bateria utilizada tem a capacidade de 15 Coulomb.

Figura 3.31: Bateria de Li-Po utilizada.

A bateria de *polímero de lítio* é capaz de fornecer 15 ampères instantaneamente sem que provoque queda substancial na tenção, sendo esta característica indispensável em regime contínuo. ela é capaz de fornecer 1 ampère durante uma hora ou 2 ampères por meia hora e assim por sucessivamente.

A bateria utilizada neste projeto apresentou aproximadamente 30 gramas e também possuem reduzida dimensão. Este peso e dimensão são significativamente leve e pequeno respectivamente se comparados com outras baterias encontradas no mercado com a mesma capacidade de potência elétrica.

É ilustrado na **figura 3.32** a colocação das duas baterias de "Li-Po" na chapa de proteção peitoral do robô bípede. Foi utilizado um carregador "E_sky" modelo "EK2-0851".

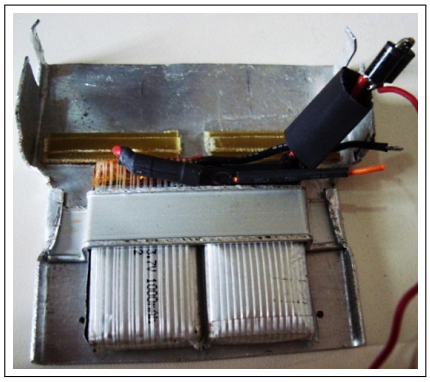

Figura 3.32: Instalação da bateria de Li-Po na chapa protetora peitoral do Robô Bípede.

São ilustrados nas **Figuras 3.33** e **3.34** o carregador utilizado e o seu esquema eletrônico, respectivamente. Um módulo supervisor das baterias de Li-Po foi codificado no processador para monitorar a tensão a fim de evitar danos às baterias.

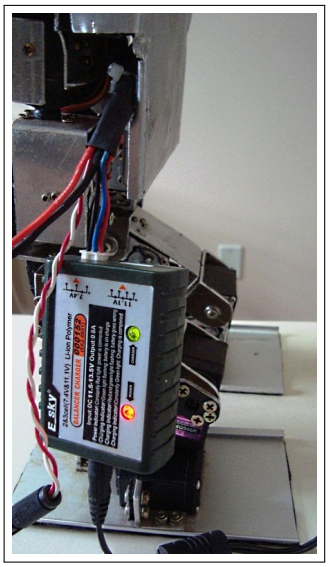

Figura 3.33: Carregador de bateria Li-Po.

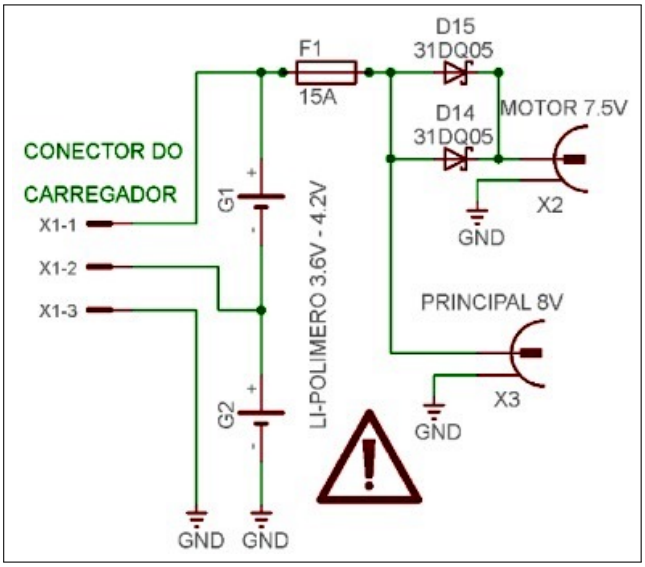

Figura 3.34: Circuito eletrônico da bateria de Li-Po do Robô Bípede.

Concluída a montagem mecânica e os circuitos auxiliares, seguimos para o desenvolvimento do *hardware* e *firmware*. Primeiramente será definido o circuito eletrônico e sua montagem. Posteriormente serão realizadas as definições do *software* embarcado, bem como as ferramentas de programação, os protocolos de comunicação e fluxograma do sistema embarcado no capítulo 4.

Capítulo 4

Projeto - Circuitos Digitais

4.1 Circuito Eletroeletrônico

Durante o desenvolvimento do *hardware* para CPU, observou-se que o mesmo seria de grande complexidade tanto para o desenvolvimento do circuito, quanto para a programação e testes. Deste modo, para facilitar a compreensão e descrição, o desenvolvimento do *hardware* foi dividido em seguintes etapas:

- Desenvolvimento do regulador monolítico de tensão 5 Volts e proteções,
- Programação do microcontrolador por ICSP,
- Comunicação via porta USB e o microcontrolador PIC,
- Leitura do botão INICIAR pela entrada digital,
- Acionamento das saídas digitais para os LEDs indicadores TESTE e MONITOR,
- Leitura das duas entradas analógicas para monitoramento das dontes de energia,
- Leitura do banco de memória 1, 2 e 3, via barramento I2C,
- Leitura do relógio de tempo real via barramento I2C,
- Desenvolvimento da interface lógica de 5 Volts para 7 Volts,
- Controle dos servomotores pela interface lógica,
- Desenvolvimento da placa de filtragem de ruído e conexão dos servos,
- Definição de todos pontos de entradas, saídas e definição do *fuses* internos do microcontrolador PIC,

- Desenvolvimento e desenho do circuito eletrônico final,
- Montagem final da placa eletrônica,
- Integrar todas as funcionalidades do *firmware* e controle dos fluxos de dados do microcontrolador PIC e o sistema bípede via USB.

Cada etapa não precisa necessariamente seguir esta ordem, pois em cada uma surge novos problemas e desafios, que muitas vezes é necessário retornar a alguma etapa para corrigi-las. Um exemplo específico de correção importante durante o projeto foi o desenvolvimento da "Interface lógica de 5 volts para 7 volts". O processador envia pulsos de 5 volts e os servos estavam trabalhando com 7 volts proveniente da bateria, deste modo havia uma incompatibilidade do sinal, que durante alguns testes queimou processadores e ocasionou falhas de funcionamento. O interfaceamento resolveu o problema.

Como a placa eletrônica principal já havia sido montada, foi preciso fazer uma placa eletrônica vertical para se adaptar ao circuito, conforme mostra a **Figura 4.1**.

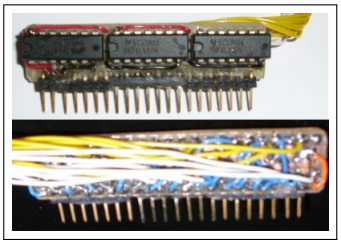

Figura 4.1: Placa driver e interface de 5 V para 7 V.

Por conta dos vários testes, a montagem tipo *"Manhattan-Style"* do circuito eletrônico foi utilizada devido ao tempo reduzido para a implementação do projeto. Foi tomada a decisão de não se desenvolver a placa de circuito impresso para o projeto. A **Figura 4.2** mostra o resultado da montagem da placa principal. Ao final foi obtido o esquema do circuito eletrônico para CPU, ilustrado na **Figura 4.3**.

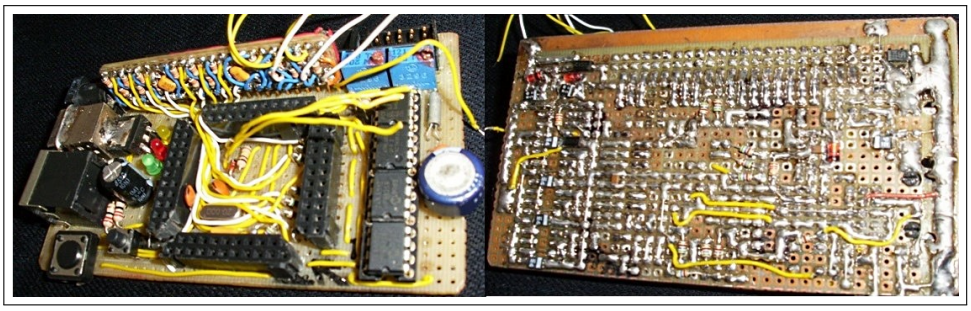

Figura 4.2: Montagem do circuito eletrônico da placa principal - *"Manhattan-style"*.

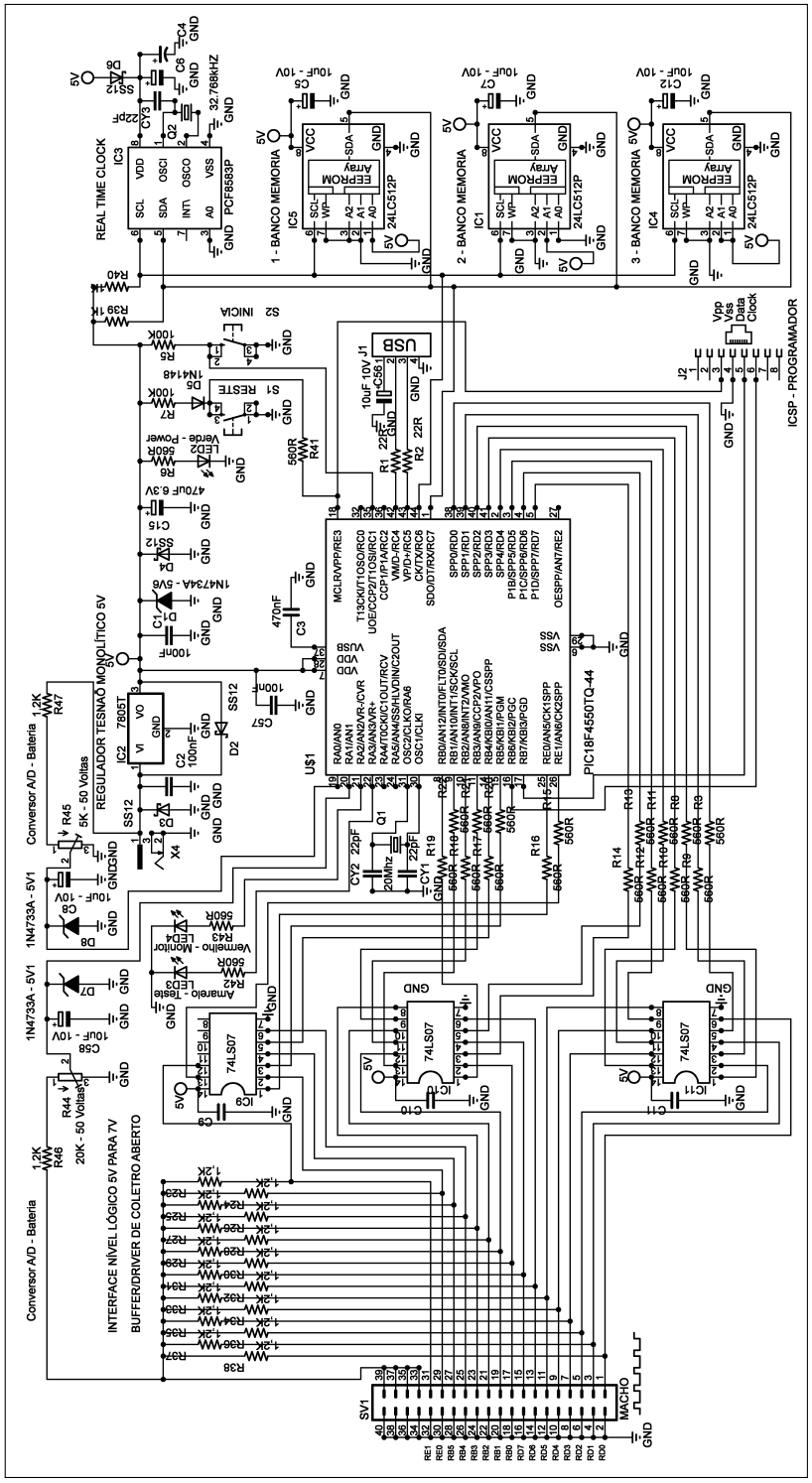

Figura 4.3: Circuito eletrônico da placa principal desenvolvido em CAD Eagle.

Conforme explicado no capítulo 3, tanto os ruídos quanto as interferências tiveram destaque no desenvolvimento da placa de filtragem de ruído e da conexão dos servos.

O valor residual e periódico da tensão que sobrepõe ao valor médio de tensão da fonte C.C., é denominado de tensão de ondulação "Vr(rms)". A fim de reduzir a tensão de ondulação usa-se um filtro capacitivo. Como referência de cálculo para tensão de ondulação "Vr(rms)" e filtragem da fonte alimentação, temos a **Equação 4.1** apresentado por Boylestad and Nashelsky (2004).

$$Vr(rms) = \frac{Icc}{4\sqrt{3}FC} \tag{4.1}$$

Onde Icc é o valor da corrente da fonte em miliamperes, F é a frequência da ondulação e C é o valor do capacitor em microfaraday. Contudo a aplicação desta equação não é fácil, pois a ondulação da tensão da fonte não era proveniente de uma retificação de uma senoidal de um transformador em onda completa com 120 Hertz, mas de transformadores chaveados, baterias e variação da corrente dos servos e seus ruídos elétricos.

De modo prático pode-se estipular 1.000 microfaraday para cada ampères consumido em uma fonte convencional usando a **Equação 4.1**, para uma ondulação próxima a 20 porcento. Devido as diversas variáveis a serem incluídas nos cálculos, foi instalado aproximadamente 34.000 microfaraday, na qual foi uma montagem viável e suficiente para que a ondulação fosse aceitável. Foi adicionado um indutor em conjunto aos capacitores a fim de formar um filtro conhecido como Filtro PI, para melhorar a filtragem de EMI. O circuito da placa anti-ruído e conexão dos servos é ilustrado na **Figura 4.4**.

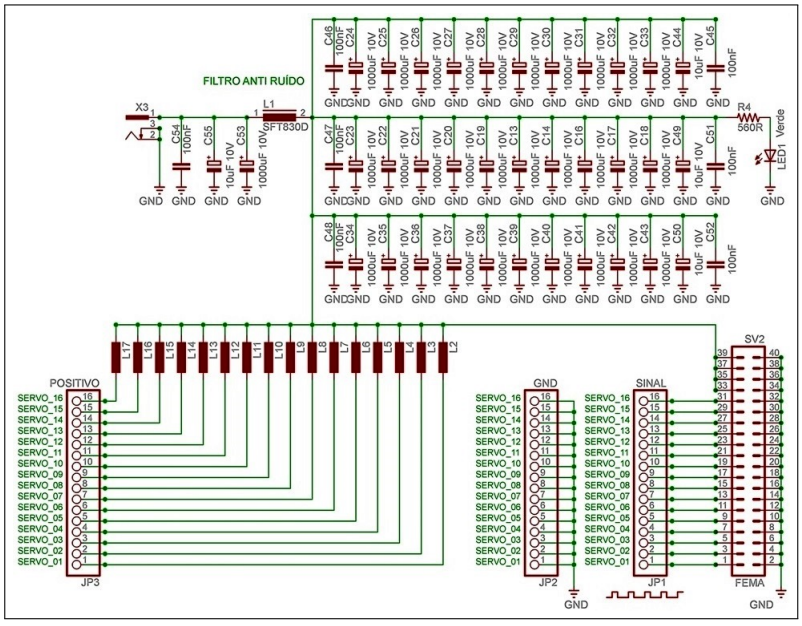

Figura 4.4: Circuito eletrônico da placa de anti ruído.

Respectivamente tem-se na **Figura 4.5** e **4.6**, o leiaute final das peças sugerido em CAD e montagem final da placa eletrônica.

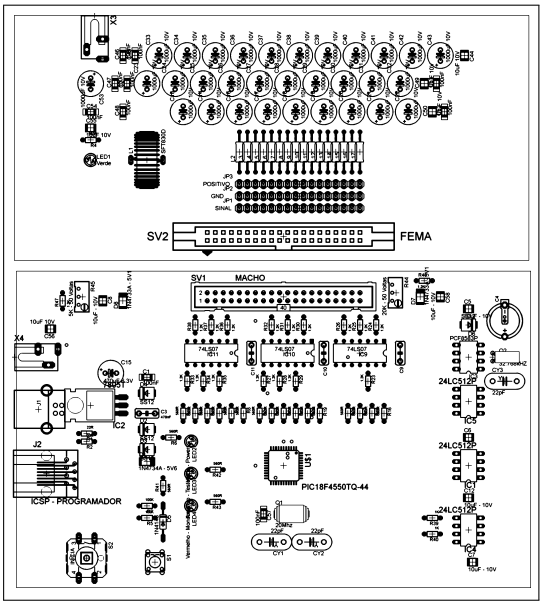

Figura 4.5: Leiaute final das peças sugerido em CAD.

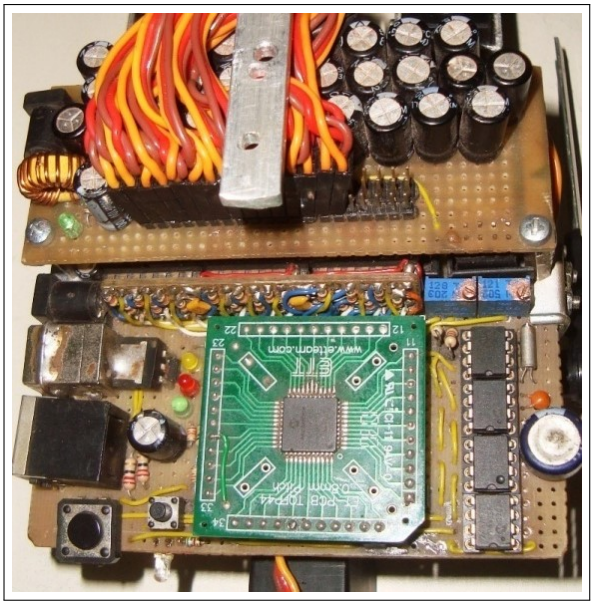

Figura 4.6: Montagem final da placa eletrônica.

A **Figura 4.7** ilustra os detalhes da Instalação Final da Placa Eletrônica e Fiação dos Servos Motores.

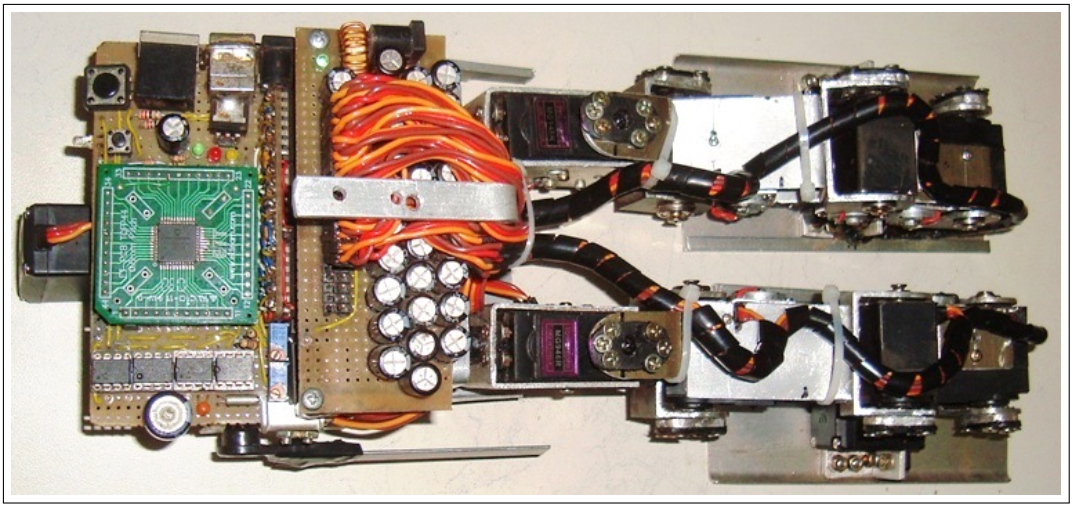

Figura 4.7: Instalação final da placa eletrônica e fiação dos servos motores

Na **Figura 4.8** é demonstrado o projeto final com os detalhes da conexão do sistema bípede ao computador, fonte e programador.

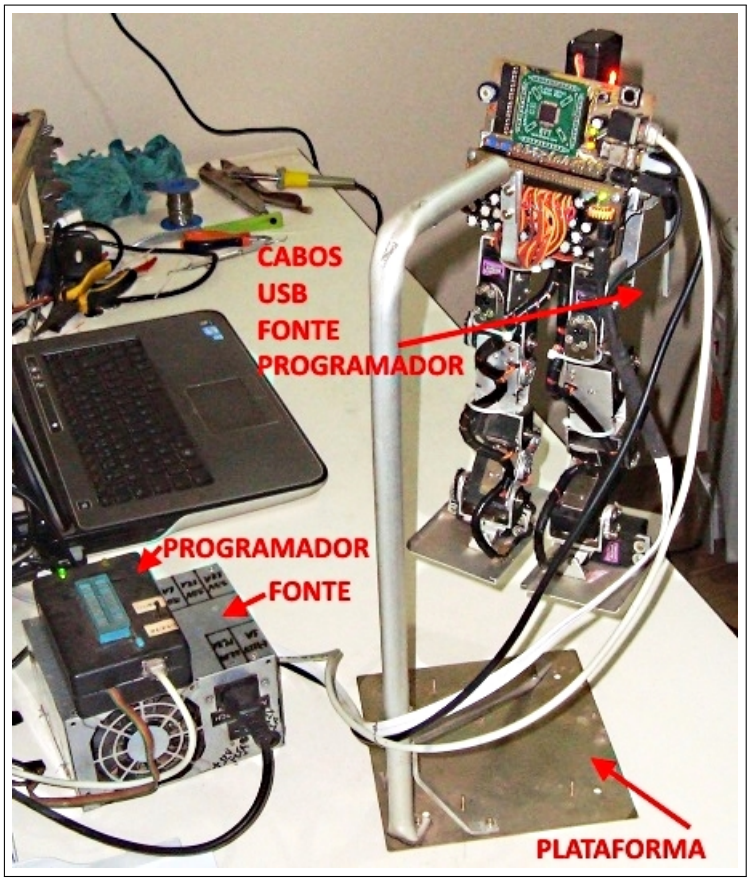

Figura 4.8: Detalhe da conexão do sistema bípede ao computador, fonte e programador

4.2 *Software* Embarcado - *Firmware*

O *software* embarcado é totalmente dedicado a um microcontrolador específico, sendo utilizado neste projeto o "PIC18F4550". Foi definido um microcontrolador de forma genérica como sendo um conjunto de circuitos lógicos integrados em um único *chip* de silício cujas conexões e comportamento podem ser especificados e posteriormente alterados, através do programa em sua memória quando necessário. Sua grande vantagem está no fato de mudar a estrutura do circuito e sua operação, bastando apenas uma mudança no programa e algumas pequenas modificações no circuito físico, quando necessário.

Esse *software* é desenvolvido ou modificado em um computador comum e posteriormente transferido para a memória interna do microcontrolador através dos chamados "gravadores" que possuem circuitos específicos utilizados para esse fim. Por meio destes gravadores, que utilizam programas próprios, é possível transferir programas e dados do computador para o microcontrolador, e viceversa, além de permitir a eliminação do conteúdo de memória do mesmo ou a re-configuração de parâmetros do componente Noleto and Siqueira (2007).

A grande aplicabilidade dos microcontroladores em robótica se deve a sua precisão de cálculos com tempo, baixo custo e fácil reutilização através de regravação nos modelos que possuem memória *flash*.

O programa SERB foi desenvolvido em linguagem C, compilado em um arquivo no formato hexadecimal e enviado por meio de um gravador especial "PICkit-2". Um programador equivalente ao modelo "PICkit-2" precisou ser adaptado por um conector RJ45 para agilizar nas gravações, seguiu as instruções do *datasheet* do fabricante do Microcontrolador "PIC". Conforme demonstra a **Figura 4.9**.

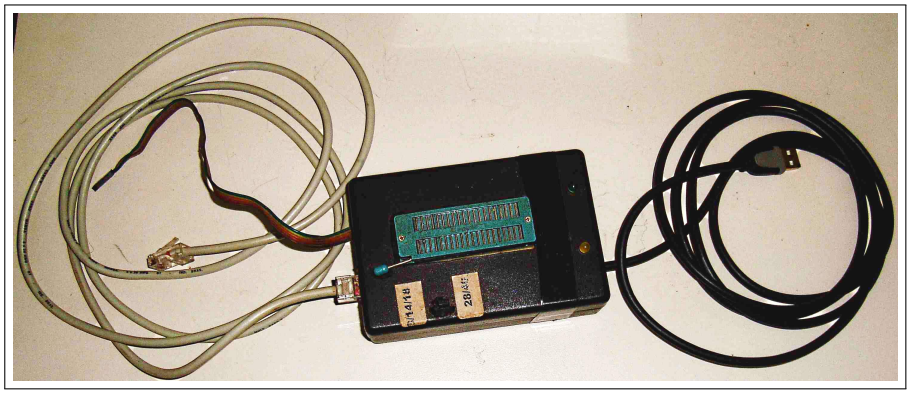

Figura 4.9: Programador "PICKIT-2" utilizado no projeto.

Neste trabalho utilizou-se o microcontrolador "PIC18F4550", da Microchip, onde o diagrama de blocos é apresentado na **Figura 4.10**. As características internas suplementares do microcontrolador são apresentadas na **Tabela 4.1**. O microcontrolador utilizado possui barramento de dados de 8 bits e utiliza a filosofia da arquitetura "RISC" possuindo um total de 75 instruções de controle ou 83 instruções no modo estendido Zanco (2005).

Na **Figura 4.10** é possível observar as unidades *Ports* (PortA, PortB, PortC e PortE) de entrada e saída (I/O), as unidades de *clock* (OSC1 e OSC2) para osciladores externos, as unidades *Timers* (temporizadores), o barramento "BUS", o barramento de dados, a unidade lógica aritmética e a unidade de memória.

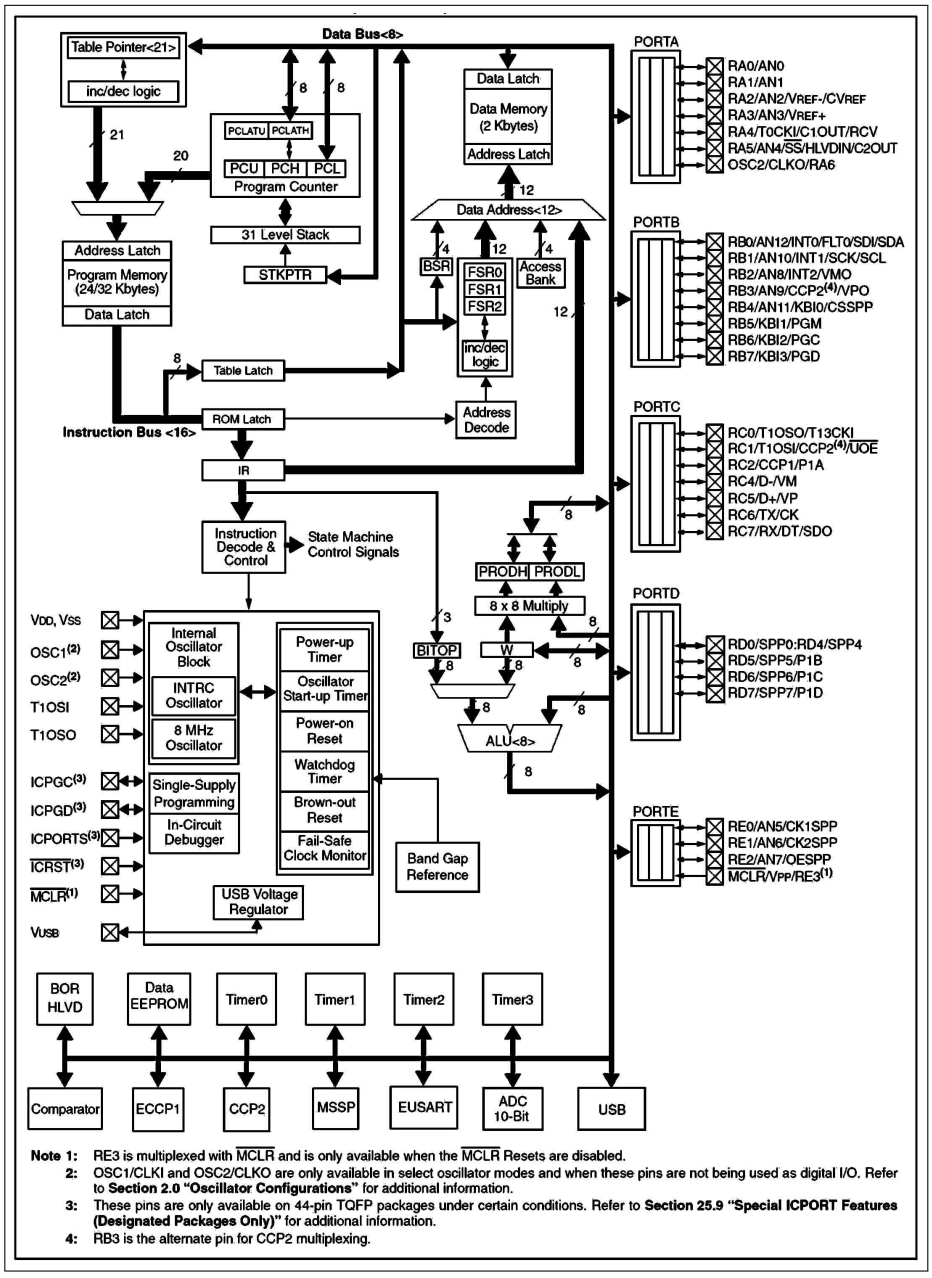

Figura 4.10: Diagrama de bloco do microcontrolador Microchip "PIC18F4550" Fonte: Microchip (2009).

Funcionalidades	**PIC18F4550**
Freqüência de Operação	DC − 48MHz
Memória de Programa (Bytes)	32768
Memória de Programa (Instruções)	16384
Memória de Dados (Bytes)	2048
Memória EEPROM de dados (Bytes)	256
Fontes de Interrupção	20
Portas de E/S	Portas A, B, C, D, E
Temporizadores	4
Captura/Comparação−Módulos PWM	1
Captura/Comparação Avançados − Módulos PWM	1
Comunicações Seriais	MSSP, EUSART
Módulo USB (Universal Serial Bus)	1
Porta Paralela	Sim
Módulo ADC 10bits	13 canais
Comparações	2
Resets (e Delay)	POR, BOR, Instrução RESET, Stack Full, Stack Underflow(PWRT, OST), MCLR , WDT
Detecção Prog. de baixa tensão	Sim
Reset Programável Brown−out	Sim
Conjunto de Instruções	75 instruções, 83 no modo estendido
Apresentação	PDIP − 40 pinos, FN − 44 pinos, TQFP − 44 pinos

Tabela 4.1: Funcionalidades do Microcontrolador "PIC18F4550".
Fonte: Microchip (2009).

Os fatores fundamentais da arquitetura do PIC:

- **Ciclo de Máquina:** Um microcontrolador pode ser entendido como uma máquina que executa operações em ciclos. Todos os sinais necessários para a busca ou execução de uma determinada instrução devem ser gerados dentro de um período de tempo denominado Ciclo de Máquina. Nos microcontroladores PIC, o sinal do *clock* é internamente dividido por quatro, no caso específico do projeto, foi utilizado um *clock* externo de 48MHz, tendo um *clock* interno de 12 MHz, e consequentemente, cada ciclo de máquina dura aproximadamente, devido a dízimas periódica, 83,33 nanosegundos Luz (2003);

- **Pipeline:** É uma técnica de hardware que permite que a CPU realize a busca de uma ou mais instruções além da próxima a ser executada, para aproveitar o máximo do ciclo de máquina. Deste modo é criado um paralelismo que permite a execução de um nova instrução em um período em que esteja executando outra instrução em um ciclo diferente. Para o caso do PIC, existem quatro fases de ciclo de máquina que são chamados de: Q1, Q2, Q3 e Q4. Na sequência o ciclo primeiro é responsável pela extração de uma determinada instrução da memória, o segundo ciclo é responsável pelo carregamento dos registradores da CPU, o terceiro ciclo é responsável pela execução da instrução e o quarto ciclo é responsável por retornar a memória os dados da execução Souza (2011);

- **Pilha:** É o local na memória RAM onde é guardado os endereços de retorno antes de ser executado pela CPU. Por causa do tamanho pequeno da área de pilha, o PIC não aceita funções recursivas Souza (2011);
- **Barramento:** A arquitetura possue três barramentos endereço, dados e controle.
- **Periféricos:** Os microcontroladores, em geral, possuem todos os periféricos necessários num único *chip*, tais como memórias, *timer's*, portas de comunicação, conversores de sinais analógicos para digital, porta USB e outros já interligados por barramento internos Oliveira and Andrade (2006).

4.2.1 *Software* Embarcado Robô Bípede - SERB

O software embarcado foi desenvolvido pelo "MPLAB IDE" Versão 8.50, software da Microchip, em conjunto com o compilador "CCS C *Compiler*" versão 4.068. A plataforma "MPLAB IDE" é ilustrada pela **Figura 4.11**.

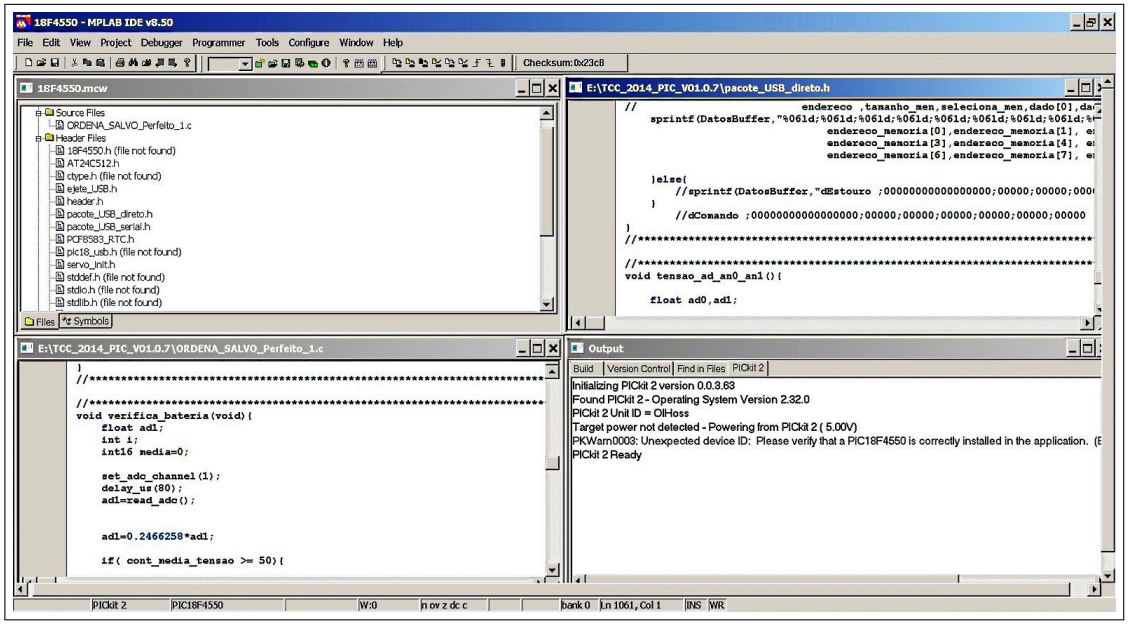

Figura 4.11: MPLAB IDE V8.50.

O *Software* Embarcado Robô Bípede, encontra-se na versão V01.0.7. As demais versões são evoluções parciais das funcionalidades, durante o desenvolvimento do projeto. No **Apêndice A** encontra-se o código principal do projeto o (SERB.c). Para efetuar a

comunicação do PIC com a Porta USB, o compilador "CCS C" disponibiliza algumas bibliotecas. A comunicação pode ser realizada de diversas maneiras:

- **HID:** O Dispositivo USB é reconhecido pelo sistema operacional como um Dispositivo de Interface Humana, não sendo necessária instalação de *driver* para a aplicação. Neste modo a velocidade de comunicação é de 64KB/s.
- **CDC:** Emulação de uma porta COM. comunicação entre o *software* e o *firmware* seja realizada como se fosse uma porta de comunicação serial padrão. É o método mais simples de comunicação bidirecional com velocidade de comunicação de até 115 Kbps.
- **MSB:** Método customizado para dispositivos de armazenamento em massa que permite alta velocidade de comunicação (USB), limitado apenas pela própria velocidade do barramento "USB 2.0"(480Mbps).
- **Genéricos:** Existem *drivers* genéricos descritos pelos próprios fabricantes. No caso da Microchip, existe um pacote de distribuição chamado "MCHPFUSB" contendo uma variedade de *firmwares* demonstrativos de USB. Neste projeto utilizou, as funções do *driver* da Microchip (mpusbapi.dll), porém é feito uma adaptação deste driver em todas as suas funções. O driver foi especialmente recompilado para permitir que as classes do jPicUSB.dll chamem suas funções na aplicação JAVA.

O SERB foi estruturado para funcionar tanto em modo USB quanto em modo CDC Genérico. Porém, as duas modalidades não funcionam simultaneamente. Por isso, é importante haver uma seleção no momento de compilar o código na linha 60 do **Apêndice A**. Devemos escrever (const int ATIVA = USB_SERIAL) para o modo CDC ou (const int ATIVA = USB_DIRETO) para o modo Genérico.

O modo CDC é mais complexo porque exige a instalação de um *driver* que emula a porta serial, acarretando uma comunicação de baixa velocidade de transmissão e reduzindo a eficiência de comunicação entre o *software* do computador e o microcontrolador. Muitas vezes o sistema trava, sendo necessário o reinício do programa, principalmente ao desconectar o cabo USB.

Deste modo, as modalidades USB e CDC foram tratadas separadamente. No caso da programação em "Delphi" em modo CDC não será abordado, em virtude da velocidade baixa de dados, ficando somente as diretivas do compilador.

As bibliotecas utilizadas foram:

- **18f4550.h:** Corresponde o cabeçalho contendo todos os recursos para a programação do PIC. Pertence ao pacote do "CCS C";

- **servo_init.h:** Contém as posições iniciais dos servos motores ao se iniciar o robô, variáveis de sinalização (*flag*) globais e algumas variáveis inicias globais de controle dos servos.

- **servo.h** Contém a função "CommandServos()", responsável em atualizar os sinais de impulso de cada servo sincronizado pelo "timer1".

- **AT24C512.h:** Responsável pelo gerenciamento da comunicação da memória interna. Pertence ao pacote do "CCS C". Foram realizadas ligeiras modificações para permitir o melhor gerenciamento dos bancos de memória, controle de gravação sequencial otimizado e blocos de gravação em 128 Bytes.

- **PCF8583.h:** Pacote responsável pelo controle do relógio de tempo real externo, presente no pacote "CCS C", embora tenha sido instalado e programado, foi substituído com vantagens pelo "timer0" no microcontrolador, para efetuar o controle temporal do programa.

- **pacote_USB_serial.h:** Pacote responsável por receber as informações da porta serial emulada quando utilizado no modo CDC. Este módulo não teve continuidade. Ele somente recebe os tempos do servos no formato serial e não será abordado. Para utilizar este pacote deve-se incluir os pacotes (usb.h), (usb_cdc.h), (usb_desc_cdc.h) pertencente ao "CCS C" e realizar as configurações necessárias conforme a velocidade da porta serial e do vendedor/produtor da porta USB.

- **pacote_USB_direto.h:** Pacote responsável por receber as informações da porta USB Genérica bem como o gerenciamento dos menus de controle de dados e do servo motor. Para utilizar este pacote deve-se incluir os pacotes (usb.h), (header.h), (pic18_usb.h) pertencente ao "CCS C" e realizar as configurações necessárias: Iniciar o (header.h) com os valores do "vendedor/produtor" (0xD8,0x04, //vendor id 0x04D8 is Microchip) e (0x0B,0x00, //product id : Dispositivos USB personalizado 32K Flashable 10 Channel, 10 Bit A/D USB Microcontroller) da porta USB.

- **ctype.h, stddef.h, stdio.h, stdlib.h, string.h:** Bibliotecas padrões do "CCS C" para manipulação de caracteres, definições de tamanho das variáveis, padrão de entrada/saída, manipulação de strings e utilização de funções como printf(), atoi(), itoa() e outras;

A **Figura 4.12** ilustra o fluxograma do código principal do SERB.

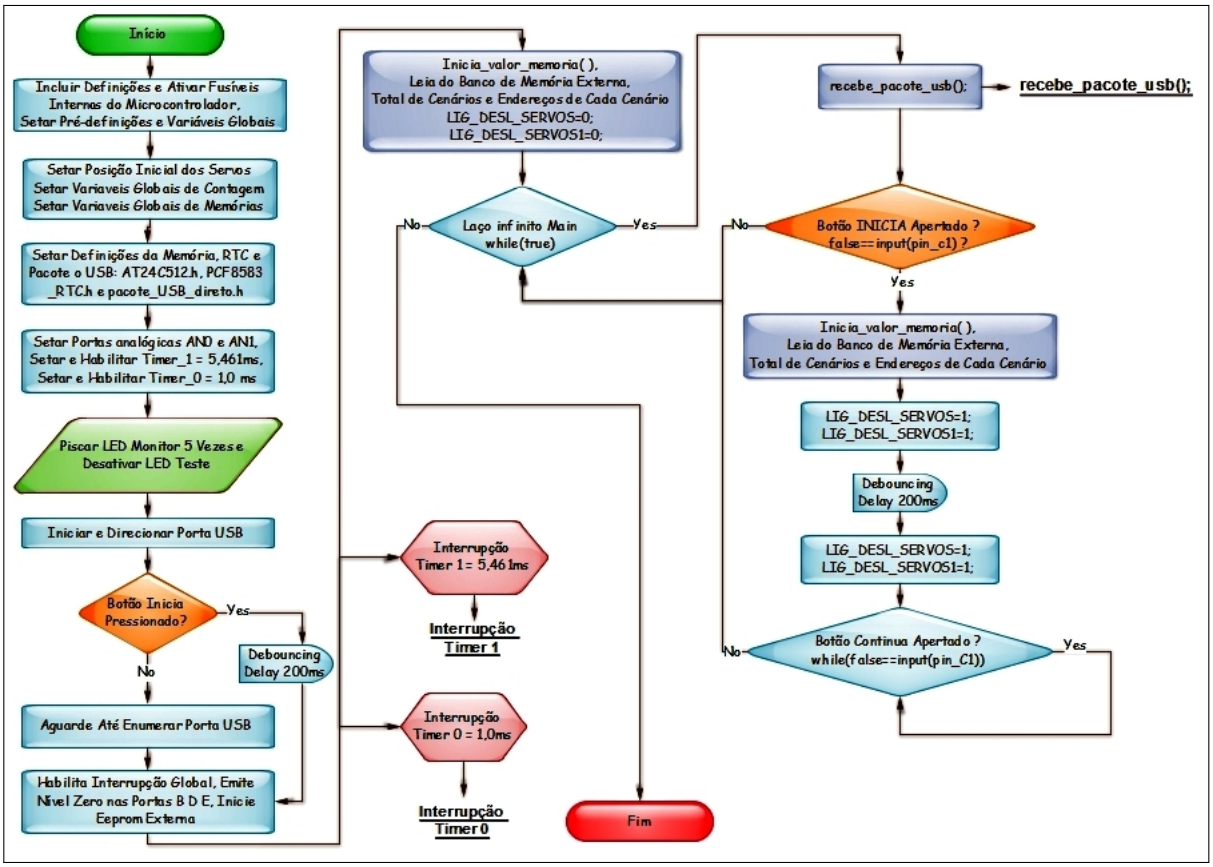

Figura 4.12: Fluxograma - Código principal SERB.

Conforme apresentado no fluxograma, quando o PIC é inicializado, efetua-se a configuração inicial dos fusíveis internos, variáveis, valores iniciais do servo, periféricos internos e externos.

A configuração do "Time1" é crítica, pois ela define o tempo em que o pulso dos servos irá ficar desabilitado e habilitado. Qualquer mudança no tempo de configuração pode gerar um deslocamento excessivo dos servos, ocasionando danos severos.

A configuração do "Timer0" também é crítica, pois define os tempos de cada deslocamento dos servos quando o Robô Bípede está funcionando via banco de memória externa. Depois da configuração inicial o LED monitor pisca 5 vezes para indicar a finalização da configuração inicial. Posteriormente, o Microcontrolador inicia a função Main (função Principal).

Continuamente a função "*main*" é interrompida tanto pelo "Timer1" quanto pelo "Timer0". Por controlar os servomotores, o "Timer1" possui prioridade sobre o "Timer0". A

Figura 4.13 ilustra o fluxograma do "Timer1" para gerar o sinal do desligamento do "PWM" necessário no funcionamento dos servomotores.

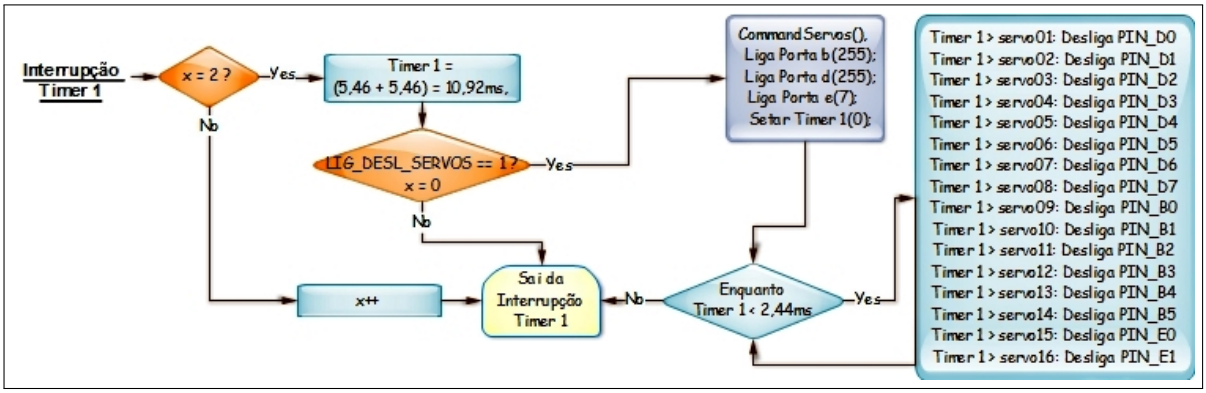

Figura 4.13: Fluxograma - Código do *Timer 1*.

O fluxograma do "Timer0" é ilustrado na **Figura 4.14**. Este *timer* chama a função (rotina_1_ms()), que tem o objetivo de gerar uma taxa de atualização a cada 1 milissegundos, executando o arquivo que se encontra gravado na memória "EEPROM" externa, caso a variável de sinalização esteja habilitada para execução.

Um controle de nível de tensão da bateria foi incorporado ao sistema de controle definido pela variável cont_geral2, que tem a função de verificar se a tensão da bateria é suficiente para o funcionamento dos servomotores.

O fluxograma do laço da função "*Main*" é apresentado na **Figura 4.15**. Observa-se neste fluxograma que primeiramente é efetuado a verificação da enumeração da "USB", que consiste basicamente na detecção de um dispositivo conectado ao barramento USB por um "*Host*" (Equipamento Hospedeiro).

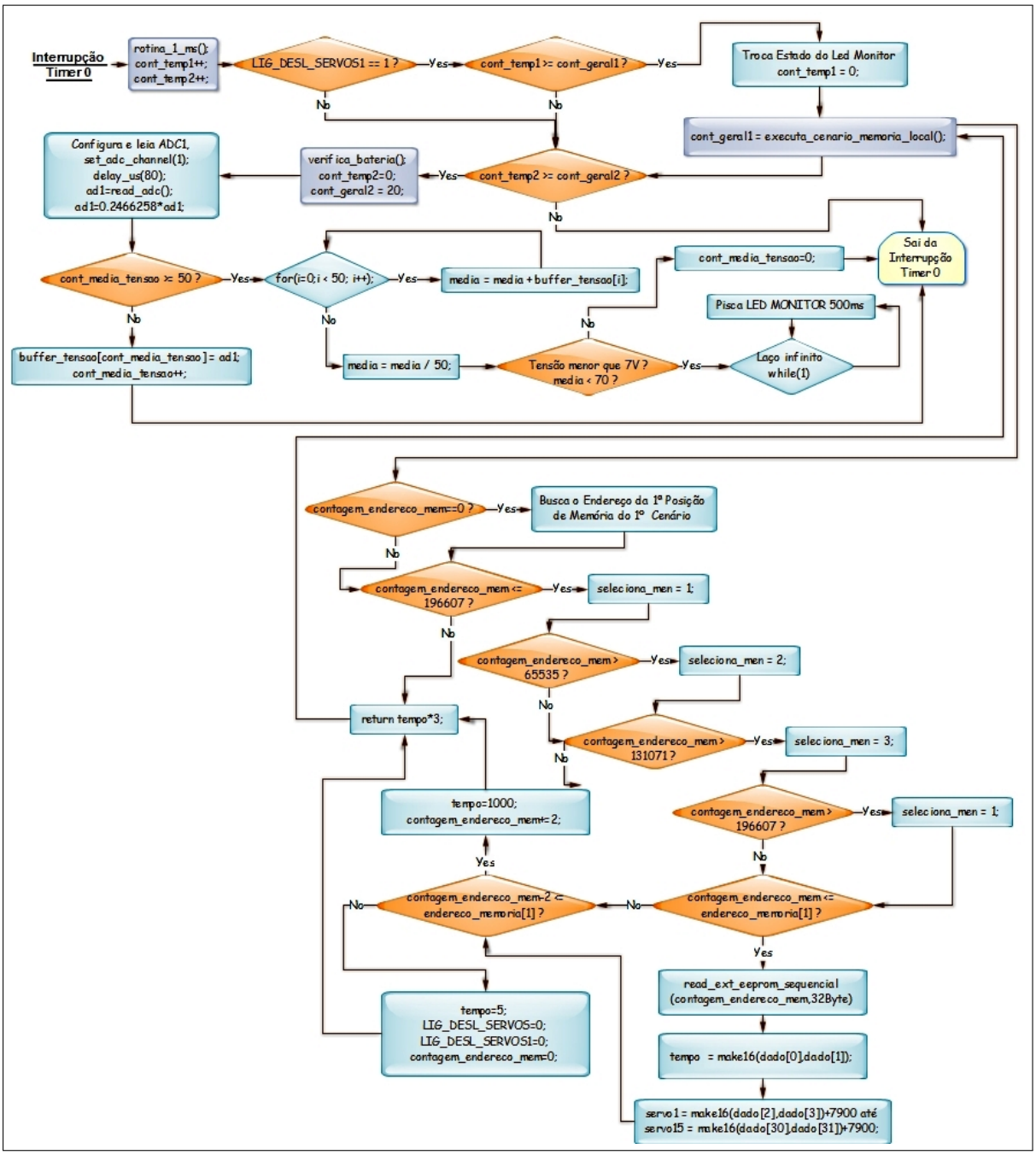

Figura 4.14: Fluxograma - Código do *Timer 0*.

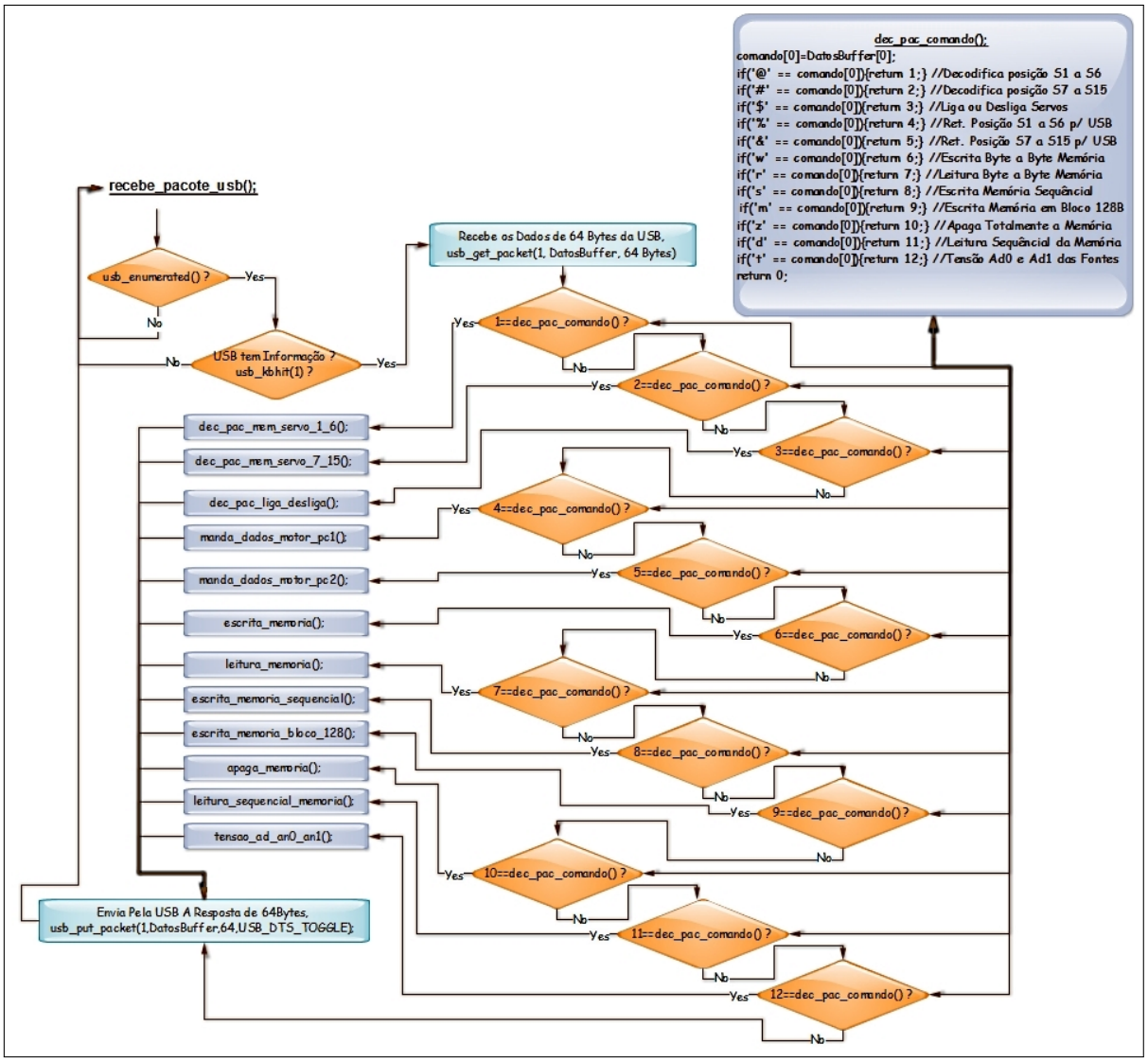

Figura 4.15: Fluxograma - Código do recebe pacote USB.

A informação necessária para o "*Host*" é definida nos descritores de dispositivos chamados. Foi elaborado um programa em Java com a finalidade de ser a interface entre o computador e o microcontrolador utilizando uma porta USB. Este programa permitiu a leitura do posicionamento dos servomotores e o envio dos comandos para o acionamento dos mesmos.

Todos pacotes da dados são padronizados no tamanho de 64 Bytes, devido ao protocolo pré-estabelecido pelo "jPicUSB". A **Tabela 4.2** apresenta as funções de tratamento dos pacotes de dados USB efetuado pelo microcontrolador:

Funções de Tratamento dos Pacotes	
int16 dec_pac_comando(void)	
Nesta função é verificado somente o primeiro elemento do pacote, não importando os demais 63 elementos. Os caracteres correspondentes possuem as seguintes funções:	
COMANDO	**DESCRIÇÃO**
@	Habilita a recepção dos valores posicionais dos servos S1 a S6,
#	Habilita a recepção dos valores posicional dos servos S7 a S15,
$	Liga ou Desliga todos os servos motores,
%	Retorna pra o software java a posição dos servos de S1 a S6,
&	Retorna pra o software java a posição dos servos de S7 a S15,
w	Escreva aleatório Byte a Byte na memória eeprom externa,
r	Leitura aleatório Byte a Byte da memória eeprom externa,
s	Escrita Sequencial da memória eeprom externa,
m	Escrita da memória eeprom externa em bloco de 128 Bytes,
z	Apaga completamente a memória eeprom externa,
d	Leitura Sequencial da memória eeprom externa,
t	Retorna para a plicação java os valores da tensão analógicas das fontes de alimentação Ad0 e Ad1.
void dec_pac_mem_servo_1_6(void)	
Recebe e modifica os valores posicionais dos servos S1 a S6. Segue o exemplo do pacote:	
(@Comando ;0000000000000000;01000;02000;03000;04000;05000;06000);	
void dec_pac_mem_servo_7_15(void)	
Recebe e modifica os valores posicionais dos servos S7 a S15. Segue o exemplo do pacote:	
(#Comando ;07000;08000;09000;10000;11000;12000;13000;14000;15000);	
void dec_pac_liga_desliga(void)	
Liga ou desliga todos os servos motores. Segue os exemplos dos pacotes:	
Comando ligar:	

($Comando ;00000000000000000;10000000000000000000000000000000000);
Comando desligar:
($Comando ;00000000000000000;00000000000000000000000000000000000);
void manda_dados_motor_pc1(void)
Retorna para a aplicação java a posição dos servos de S1 a S6. Segue o exemplo do pacote:
(%Comando ;00000000000000000;00000;00000;00000;00000;00000;00000),
Código do retorno para DatosBuffer no "PIC" para o software "JAVA PC":
sprintf(DatosBuffer,"%Comando ;00000000000000000; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld", servo1, servo2, servo3, servo4, servo5, servo6))
void manda_dados_motor_pc2(void)
Retorna para a aplicação java a posição dos servos de S7 a S15. Segue o exemplo do pacote:
(&Comando ;00000;00000;00000;00000;00000;00000;00000;00000;00000),
Código do retorno para DatosBuffer no "PIC" para o software "JAVA PC":
sprintf(DatosBuffer,"%Comando ; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld; %05Ld", servo7, servo8, servo9, servo10, servo11, servo12, servo13, servo14, servo15))
void escrita_memoria(void)
Escreve aleatoriamente byte a byte na memória eeprom externa. Segue o exemplo do pacote:
(wComando ;00000000000000000;00000;00000;00000;00000;00000;00000);
void leitura_memoria(void)
Leitura aleatório byte a byte da memória eeprom externa. Segue o exemplo do pacote:
(rComando ;00000000000000000;00000;00000;00000;00000;00000;00000);
void escrita_memoria_sequencial(void)
Escrita Sequencial da memória eeprom externa. Segue o exemplo do pacote:
(sComando ;00000000000000000;00000;00000;00000;00000;00000;00000);

escrita_memoria_bloco_128(void)	
Escrita da memória "EEPROM" externa em bloco de 128 bytes, dados estão em formato binário Segue o exemplo do pacote e sua subdivisão:	
(mComand |pp |eeeeee|ddddddddddddddddddddddddddddddddddddddddddddddddd|cccc);	
COMANDO	**DESCRIÇÃO**
mComand	Indica o comando de escrita da memória externa em bloco de 128 Bytes.
p	Tamanho de pacote binário 42 bytes ou 43 bytes. Pois 43 + 43 + 42 = 128 Bytes.
e	Endereço do inicial do pacote.
d	Dados binários do pacote.
c	Previsto para aplicação futura de CRC no pacote.
void apaga_memoria(void)	
Apaga completamente a memória eeprom externa. Segue o exemplo do pacote:	
(zComando ;00000;00000;00000;00000;00000;00000;00000;00000;00000);	
void leitura_sequencial_memoria()	
Leitura Sequencial da memória eeprom externa. Segue o exemplo do pacote:	
(dComando ;0000000000000000;00000;00000;00000;00000;00000;00000);	
void tensao_ad_an0_an1()	
Retorna para a plicação java os valores da tensão analógicas das fontes de alimentação Ad0 e Ad1. Segue o exemplo do pacote:	
(tComando ;0000000000000000;00000;00000;00000;00000;00000;00000);	
Código do retorno para DatosBuffer no "PIC" para o software "JAVA PC":	
sprintf(DatosBuffer,"tComando ;AD0: %.3f, AD1: %.3f, ",ad0,ad1);	

Tabela 4.2: Funções de Tratamento dos Pacotes.

Com o sistema SERB definido e concluído, seguiremos para o capítulo 5 no desenvolvimento do *Software* do Robô Bípede.

Capítulo 5

Projeto de Software

O *Software* do Robô Bípede foi desenvolvido em linguagem Java por meio da "IDE NETBEANS versão 7.2.1", juntamente com o kit Java de desenvolvimento padrão para plataformas Java "JDK 1.7.0_25". É importante observar que esta máquina virtual desta plataforma é de 32 bits, mesmo que o sistema operacional seja de 64 bits. A **Figura 5.1** ilustra o *software* "IDE NETBEANS versão 7.2.1" com o projeto aberto.

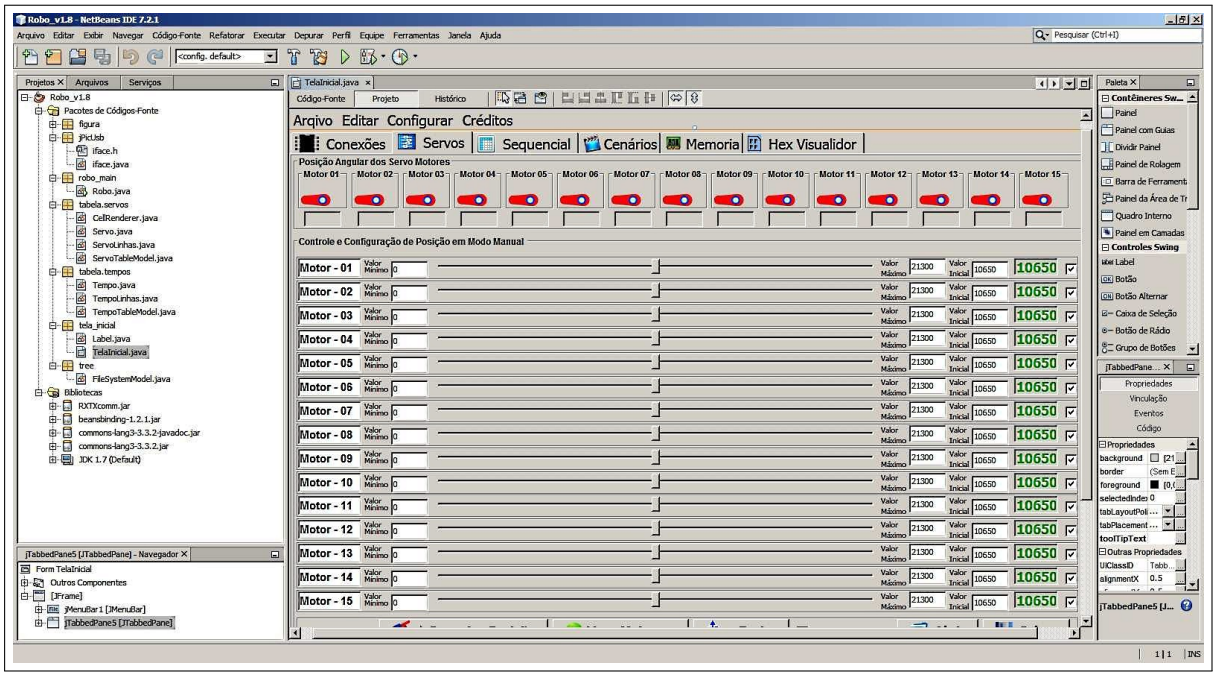

Figura 5.1: Programa "IDE Netbeans 7.2.1" e "JDK 1.7".

Observa-se na **Figura 5.2** a estrutura em árvore do código fonte e a estrutura do

diagrama de pacotes do Java. Foi adicionado ao pacote Java um kit de desenvolvimento padrão para plataformas Java (JDK) em 32 bits para o desenvolvimento do *driver* genérico USB. O JPicUSB.dll é uma biblioteca que implementa todas as funções do API Microchip USB (mpusbapi.dll), e é recompilados para permitir que a classe jPicUSB invoque as suas funções. O pacote jPicUSB foi obtido por Oñativa (2013), e deve-se adicionar ao projeto a biblioteca (jPicUsb/iface.h) ou o pacote (jPicUsb/iface.java). Para rodar o programa deve-se inserir na pasta do binário o (jpicusb.dll). O manual do jPicUsb é de acesso livre e disponível por Oñativa (2009). Devido o tamanho do projeto, será demonstrado o diagrama de classes internamente de cada diagrama de pacote.

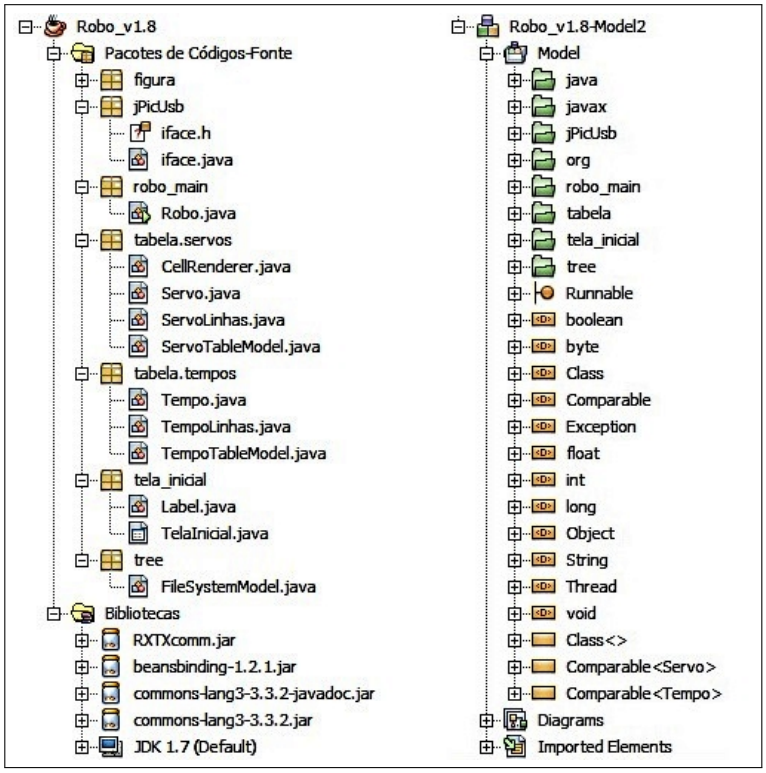

Figura 5.2: Árvore do código fonte e do diagrama de pacotes Java.

Dentre os muitos pacotes e classes que o Java possui, é possível destacar dois grandes pacotes: o pacote Java que possui as classes padrões para o funcionamento do algoritmo; e o pacote Javax que possui pacotes de extensão que fornecem classes e objetos que completam o pacote Java. Também foi adicionando o pacote org que permite a conversão de *array* primitivos, convertendo uma *string* em um *array* de *byte*. Isto permite o envio do pacote ao robô bípede em formato binário. As **Figuras 5.3**, **5.4** e **5.5** ilustram os diagramas de pacotes Java, Javax e org.

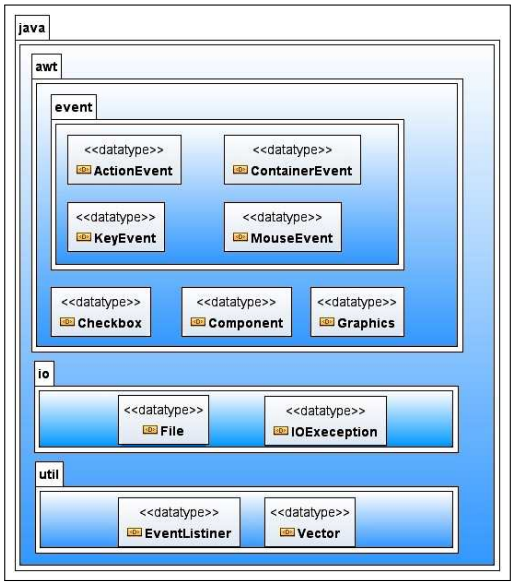

Figura 5.3: Diagrama de pacote do pacote Java.

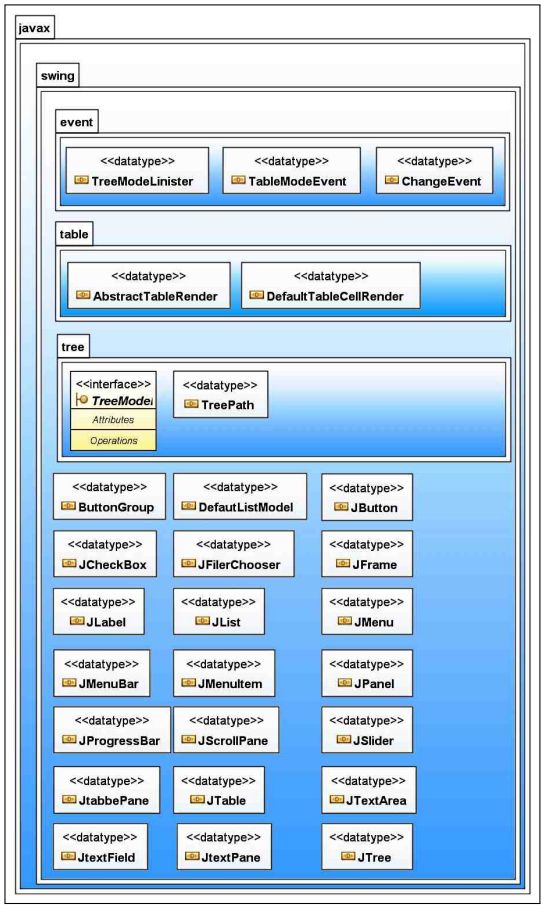

Figura 5.4: Diagrama do pacote Javax.

Figura 5.5: Diagrama de pacote Org.

A **Figura 5.6** apresenta o diagrama de pacote jPicUsb em conjunto com o diagrama de classe *iface*, juntamente com seus atributos, métodos e dependências. Esta classe responsável por toda a comunicação entre o *software* Java e o robô bípede via USB.

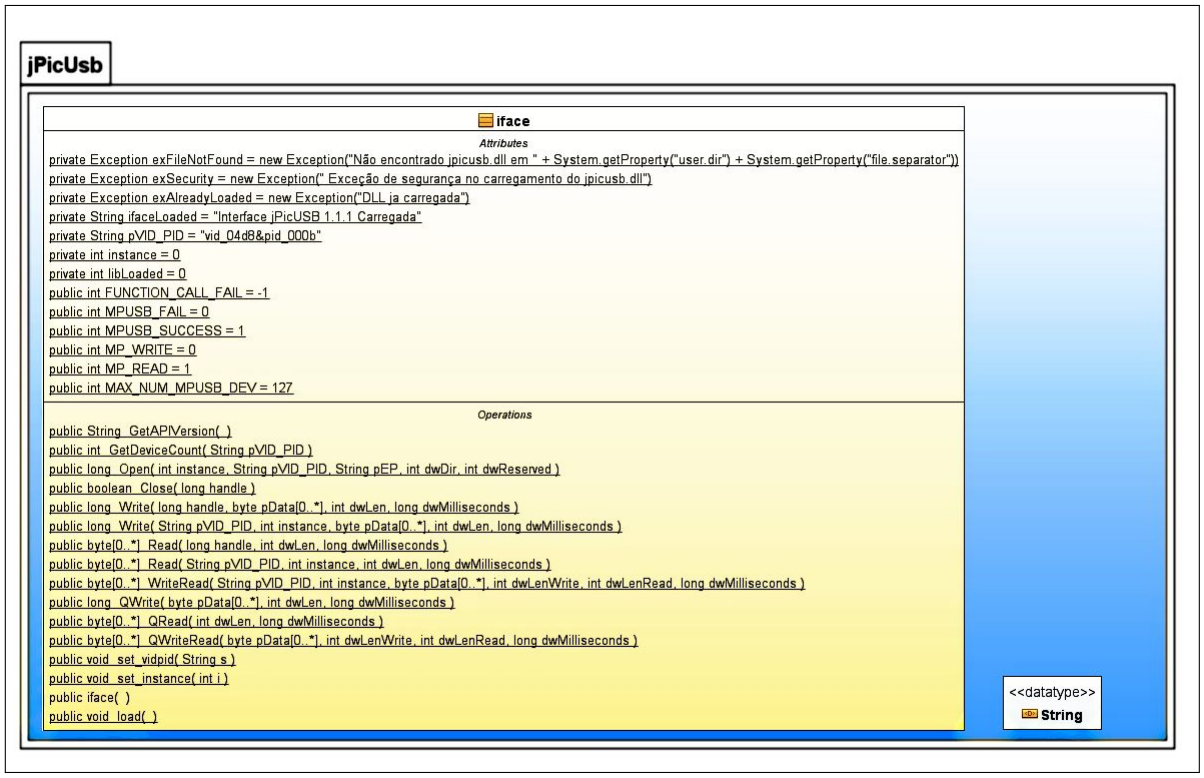

Figura 5.6: Diagrama de pacote jPicUsb com o diagrama de classe iface.

O diagrama de pacote robo_main só contém a função principal, que instancia a sua execução. Sua ilustração é apresentado na **Figura 5.7**.

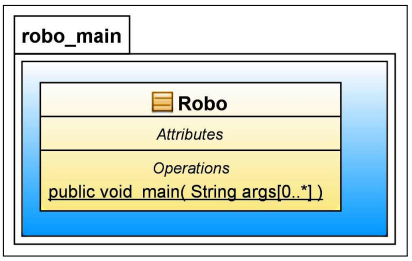

Figura 5.7: Diagrama de pacote robo_main e diagrama de classe do Robô.

O diagrama de pacotes da tabela é uma adaptação do código apresentado por Biscaro (2010). Sua aplicação é importante pois permitiu instanciar e gerenciar tabelas por meio das classes. A **Figura 5.8** mostra a estrutura do pacote tabela.

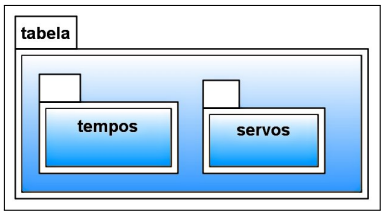

Figura 5.8: Diagrama de pacotes da tabela.

A **Figura 5.9** mostra os diagramas de classes do pacote tempos. A sua função é gerar um modelo de tabela para que possa ser herdado pelas tabelas utilizadas com controle de tempo.

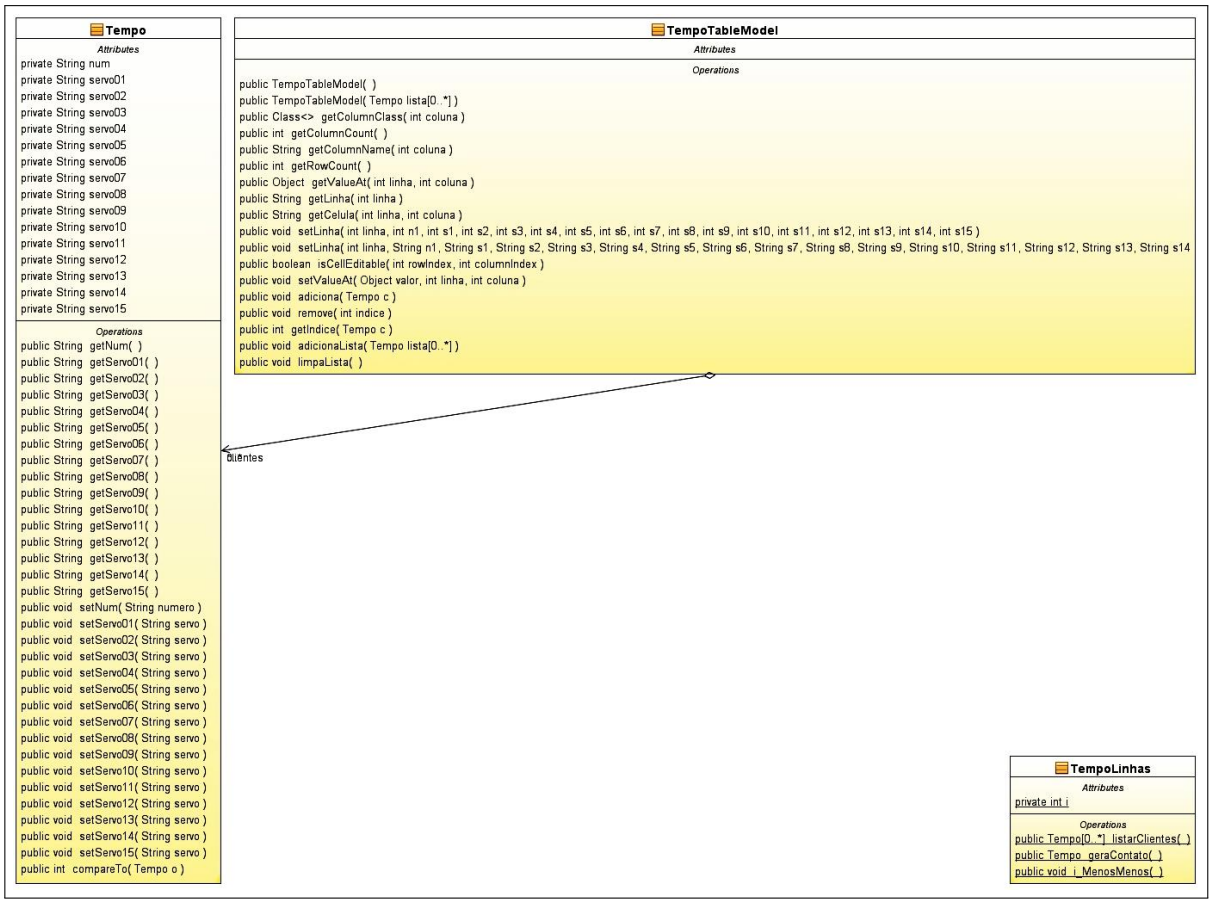

Figura 5.9: Diagramas de classes do pacote tempos.

A **Figura 5.10**, mostra os diagramas de classes do pacote servos. A sua função é gerar um modelo de tabela para que possa ser herdado pelas tabelas utilizadas com controle de servos.

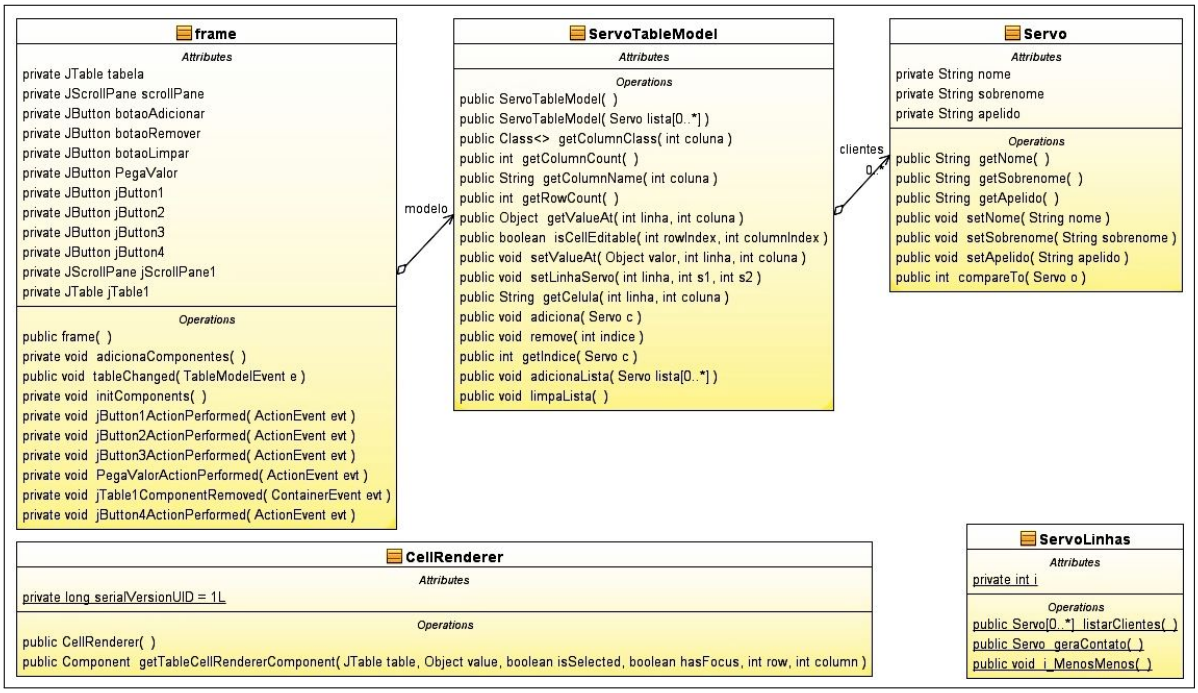

Figura 5.10: Diagramas de classes do pacote servos.

O diagrama de pacote tela_inicial ilustrado na **Figura 5.11** é o pacote mais importante do projeto, sendo responsável por gerar todas as telas de interface gráfica ao usuário, bem como todas as funcionalidades do programa desenvolvido.

Figura 5.11: Diagrama de pacote tela_inicial.

O diagrama de classe do pacote tela_inicial é ilustrado na **Figura 5.12**, juntamente com seus atributos, métodos e dependências. A classe TelaInicial possui quatrocentos e setenta e cinco atributos e cento e dezoito métodos, deste modo tornou-se inviável representar todo conteúdo.

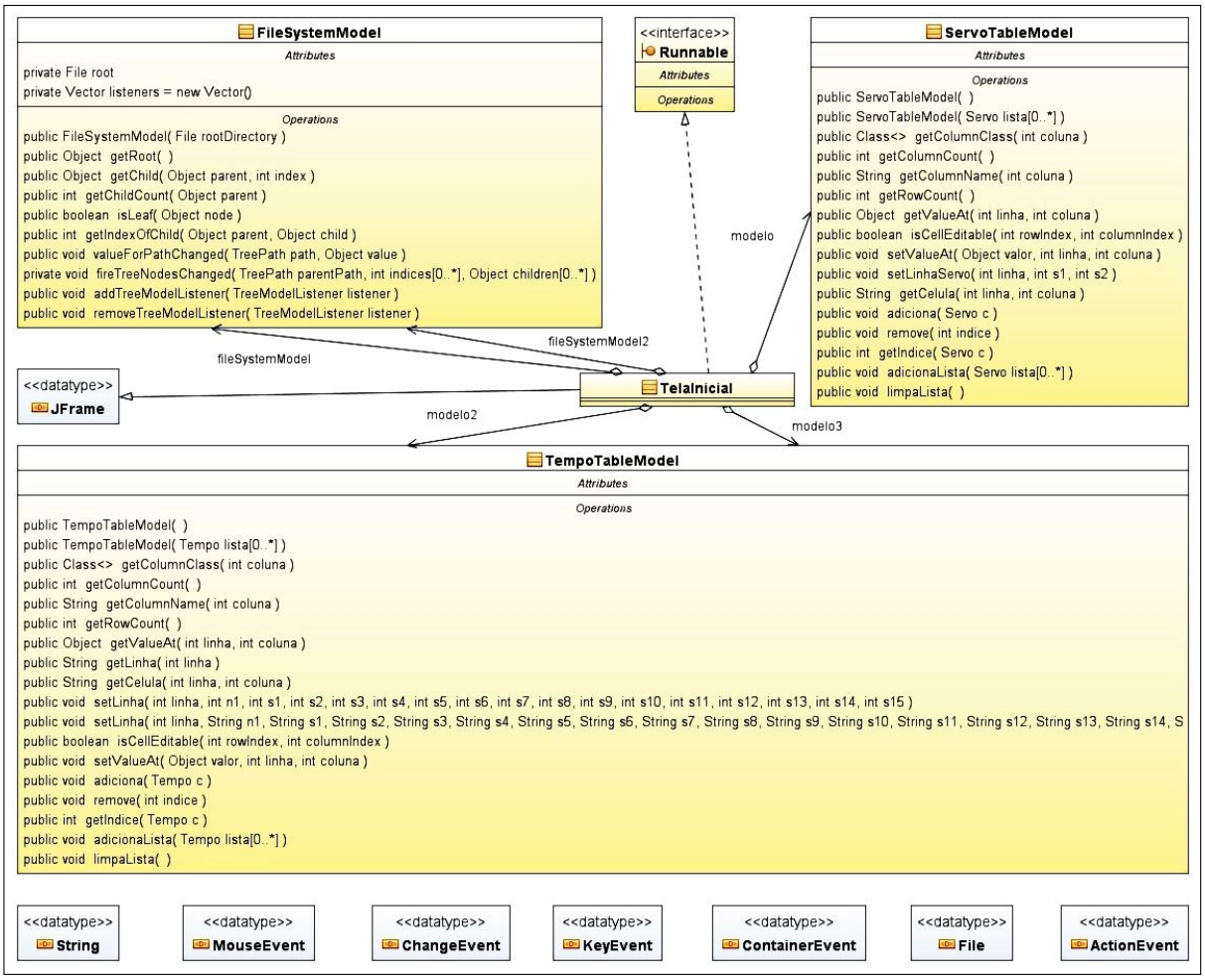

Figura 5.12: Diagramas de classes do pacote tela_inicial.

O diagrama de pacotes de *tree* juntamente com os diagramas de classes é ilustrado na **Figura 5.13**. Este pacote é fornecido pela plataforma Java e sua função é permitir o gerenciamento dos arquivos de sistema.

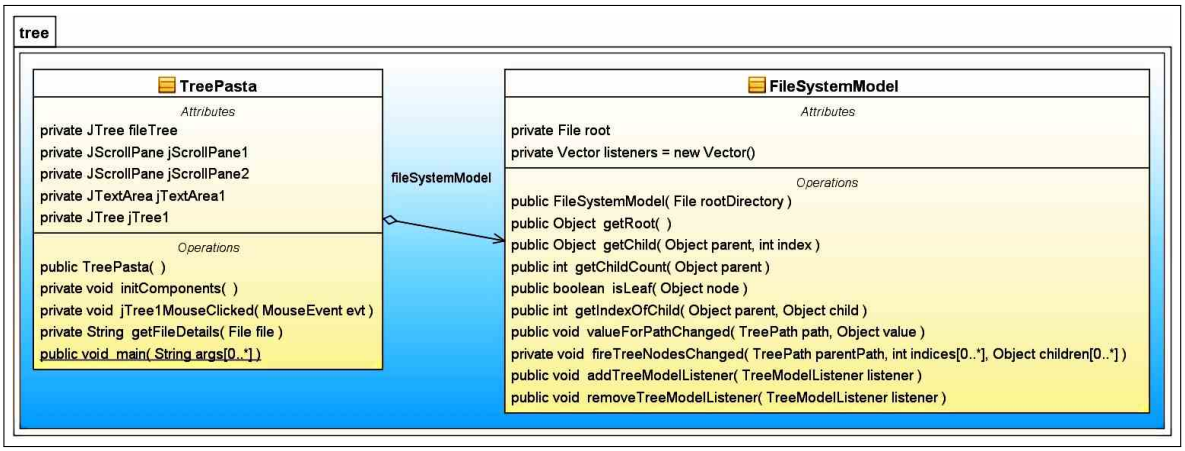

Figura 5.13: Diagrama de pacote *tree* e diagramas de classes.

5.1 Interfaces Gráficas

A interface gráfica do Robô Bípede - PC Controle (RBPC) possui seis telas: Conexões, Servos, Sequencial, Cenários, Memória, Hex Visualizador. Com exceção da tela "Hex Visualizador" todos permitem ativar e enviar comandos ao Robô Bípede. A tela Conexões tem a função de ativar a comunicação com a porta USB, verificar e receber mensagens do robô. A **Figura 5.14** ilustra a tela Conexões, que ao ser inicializada ativa a DLL (jpicusb.dll) por meio do botão "DLL Inicia" para permitir a comunicação com a porta USB.

Foi adicionado um depurador simples à tela Conexões com uma caixa de mensagens e um campo para envio de texto via USB. Nesta interface, alguns caracteres especiais foram utilizados como marcadores para iniciar os comandos de controle, tais como os caracteres '@', 't', '#', '%', '&', etc. Conforme já relatado no Capítulo 4, estes caracteres possuem a função de auxiliar o microcontrolador a identificar o início da linha de comando. É possível também salvar e abrir as informações do depurador de mensagens.

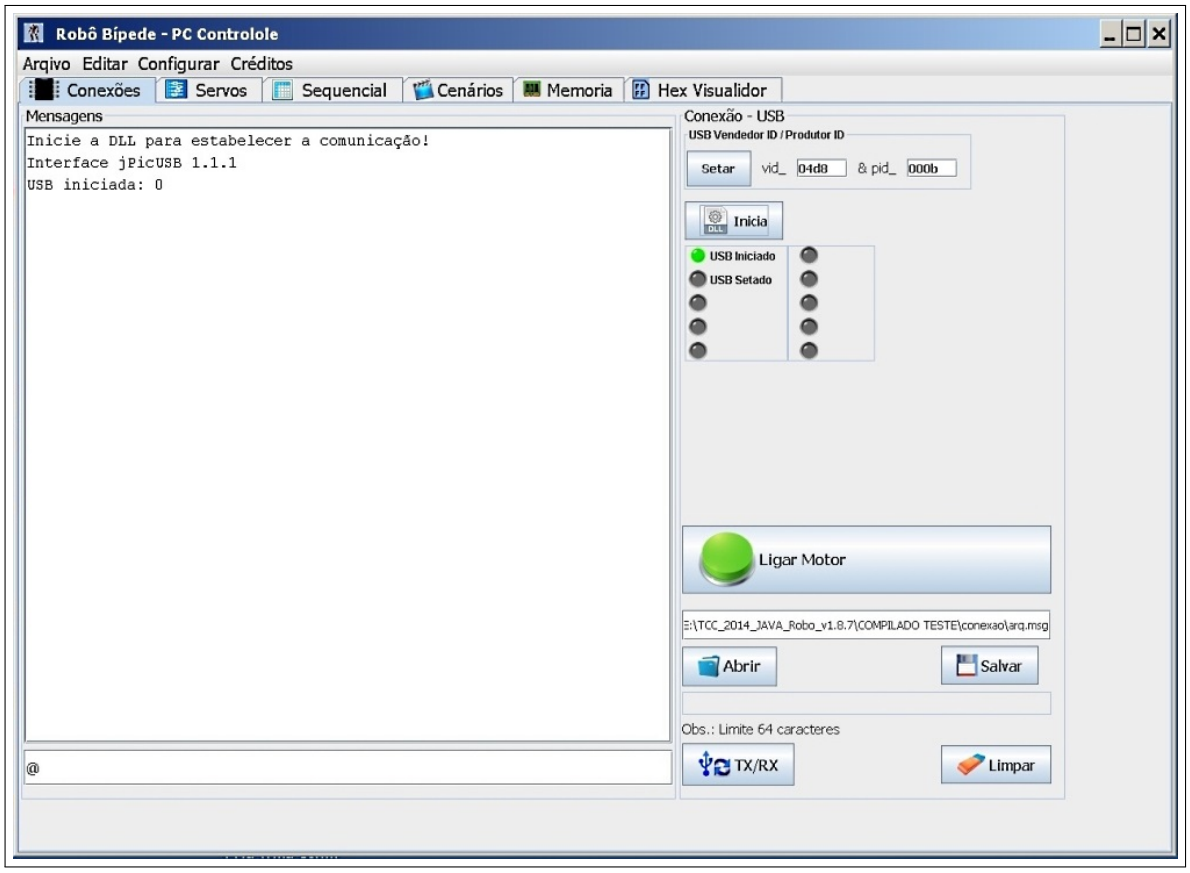

Figura 5.14: Software desenvolvido - Robô Bípede - PC Controle - Tela 1 Conexões.

A **Figura 5.15** ilustra a segunda tela denominada Servos. Nesta tela é possível posicionar os servos individualmente, definir limites máximos e mínimos de movimentos, selecionar quais servos serão modificados, selecionar envio automático ou por botão, importar a posição dos servos, renomear os nomes dos servos, visualizar o ângulo relativo e absoluto de cada servo, salvar e abrir as informações da tela e ligar e desligar os motores.

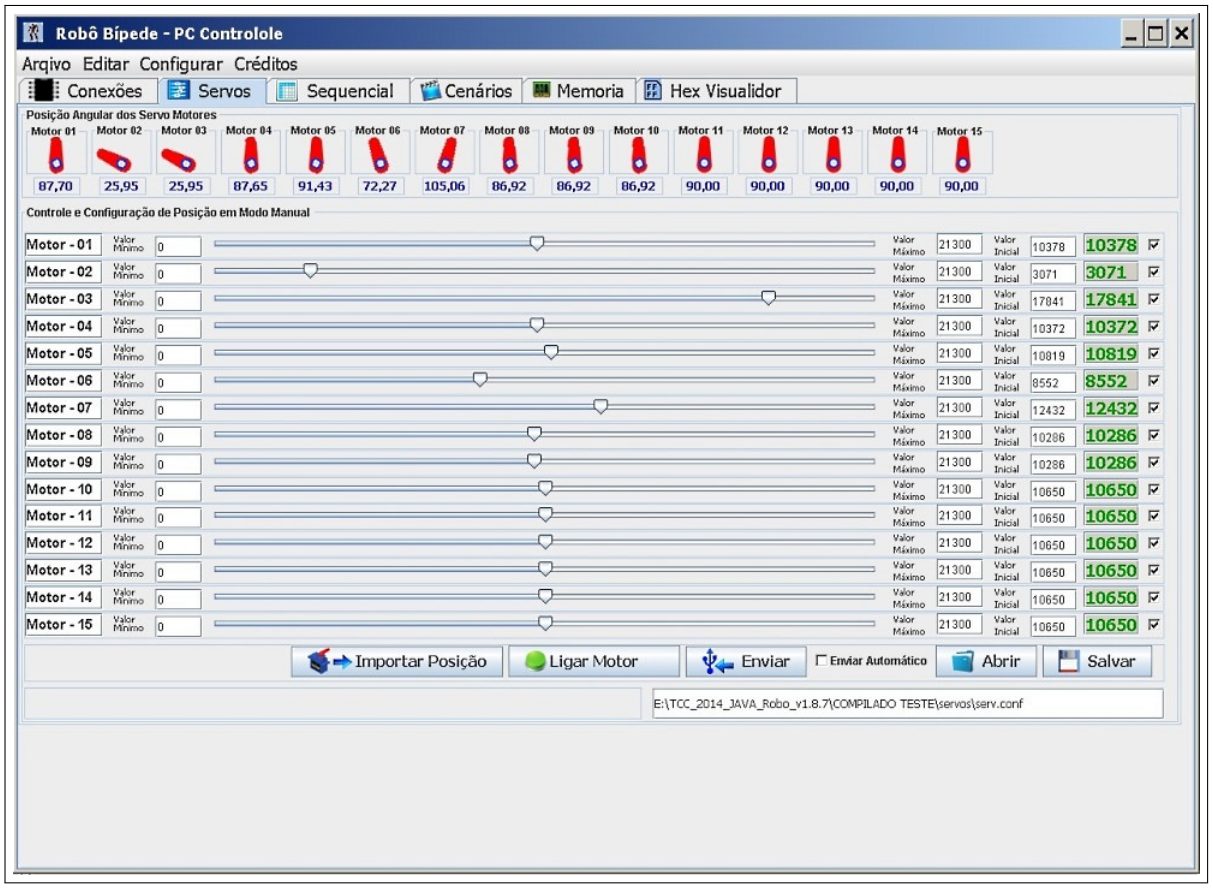

Figura 5.15: Software desenvolvido - Robô Bípede - PC Controle - Tela 2 Servos.

A **Figura 5.16** ilustra a terceira tela denominada Sequencial. Nesta tela definis-se os valores iniciais e finais de cada posição dos servos motores, também o tempo que levará o deslocamento e define os intervalos a serem executados. Deste modo é possível gerar uma rampa linear definindo automaticamente o tempo e frações de movimento. Permite salvar as tabelas para gerar arquivos que irão compor um cenário de movimento.

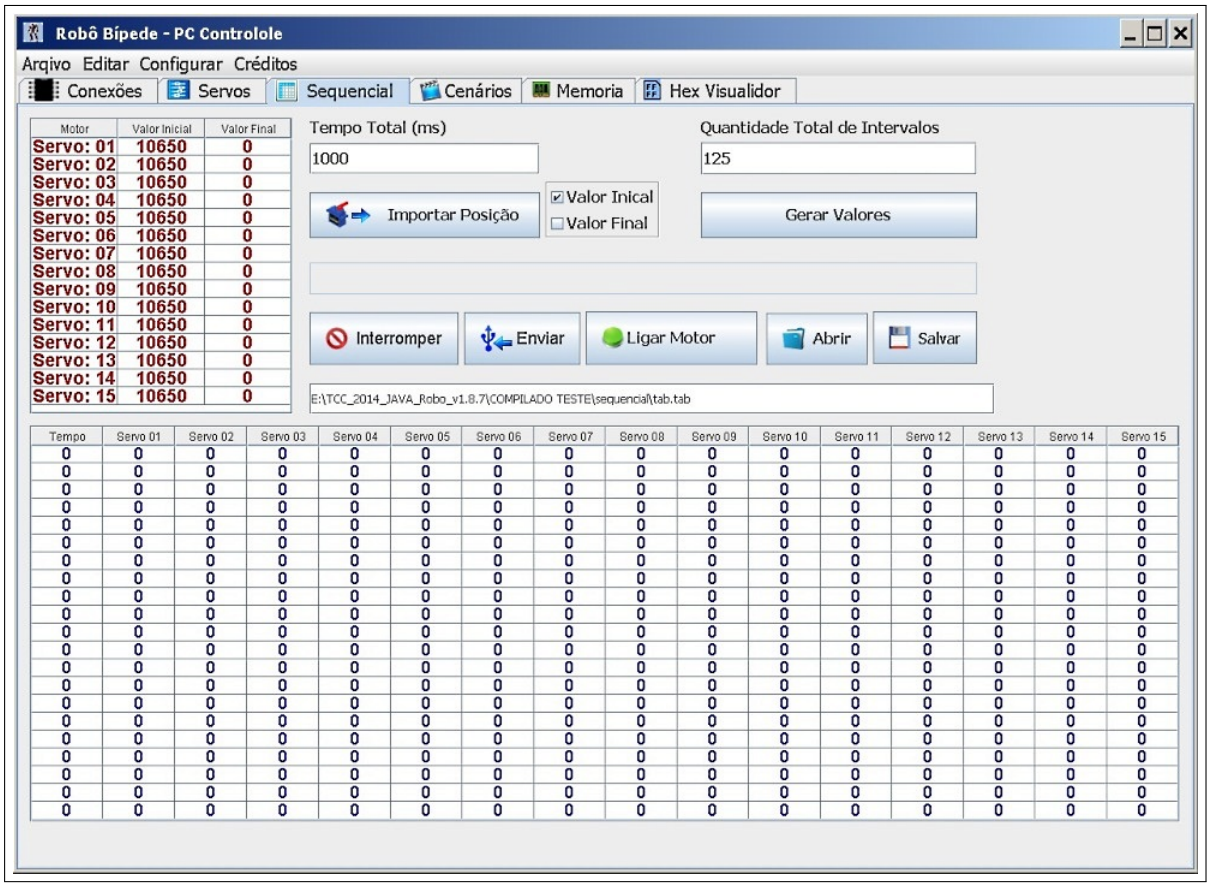

Figura 5.16: Software desenvolvido - Robô Bípede - PC Controle - Tela 3 Sequencial.

A **Figura 5.17** ilustra a quarta tela denominada Cenário. Nesta tela é possível selecionar na árvore de arquivos a sequência de tabelas que irá compor um cenário de movimento.

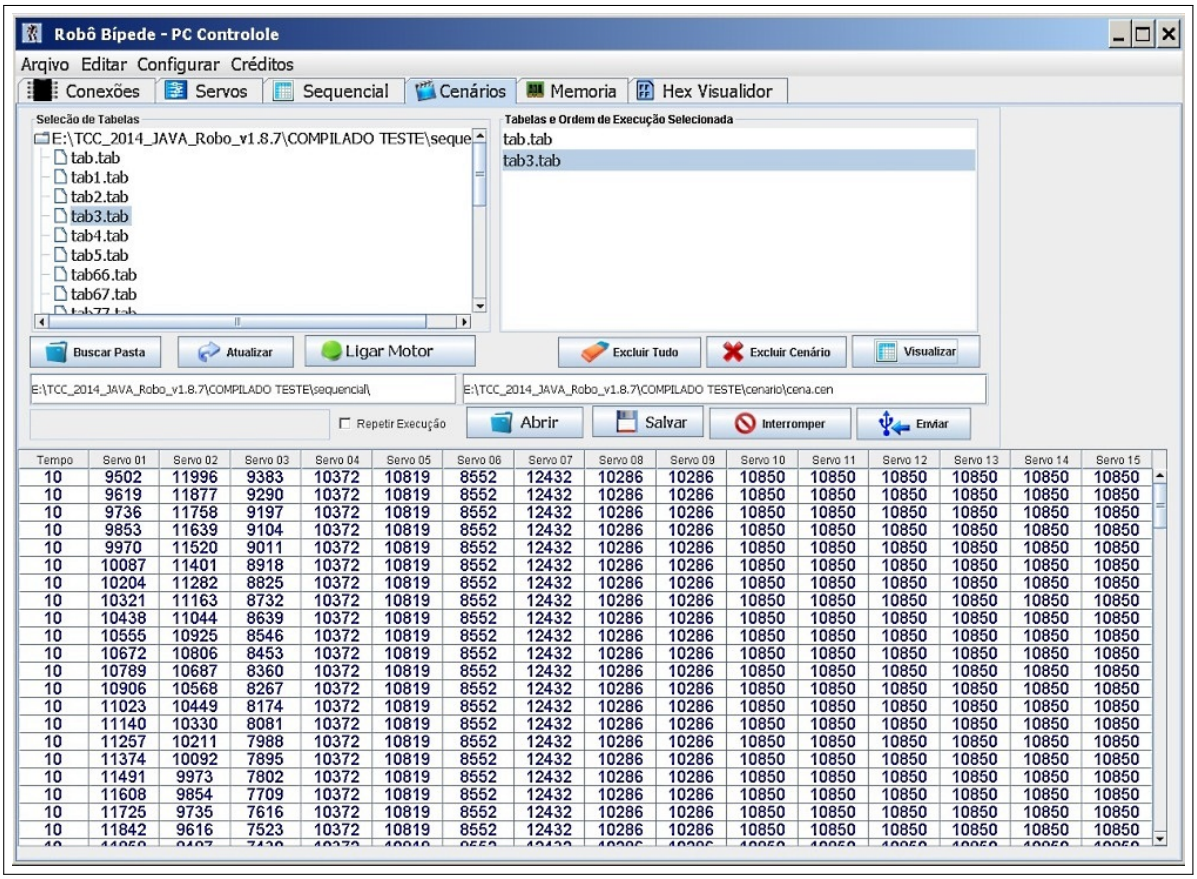

Figura 5.17: Software desenvolvido - Robô Bípede - PC Controle - Tela 4 Cenários.

A tela Memória é responsável por abrir os arquivos cenários em ordem definida pelo usuário, converter os dados para hexadecimal que será enviado ao robô e permitir o controle da memória. A **Figura 5.18** ilustra a quinta tela denominada Memória. Embora seja possível selecionar vários cenários na árvore de arquivo e grava-los na memória externa do robô, ele só irá executar o primeiro cenário.

Devido ao espaço reduzido da memória, o arquivo cenário é convertido em binário e visualizado no formato hexadecimal, para então ser enviado para o robô bípede. Este processo é um pouco lento quando o cenário é grande. Outra característica deste cenário é a possibilidade de visualizar e modificar cada posição física da memória de forma sequencial ou aleatória e, a possibilidade de apagar toda a memória. Este processo de apagar toda a memória também é um processo lento visto que existe uma latência para gravação de 5 milissegundos em média a cada grupo de 128 bytes mais o tempo de comando do microcontrolador em modo I2C.

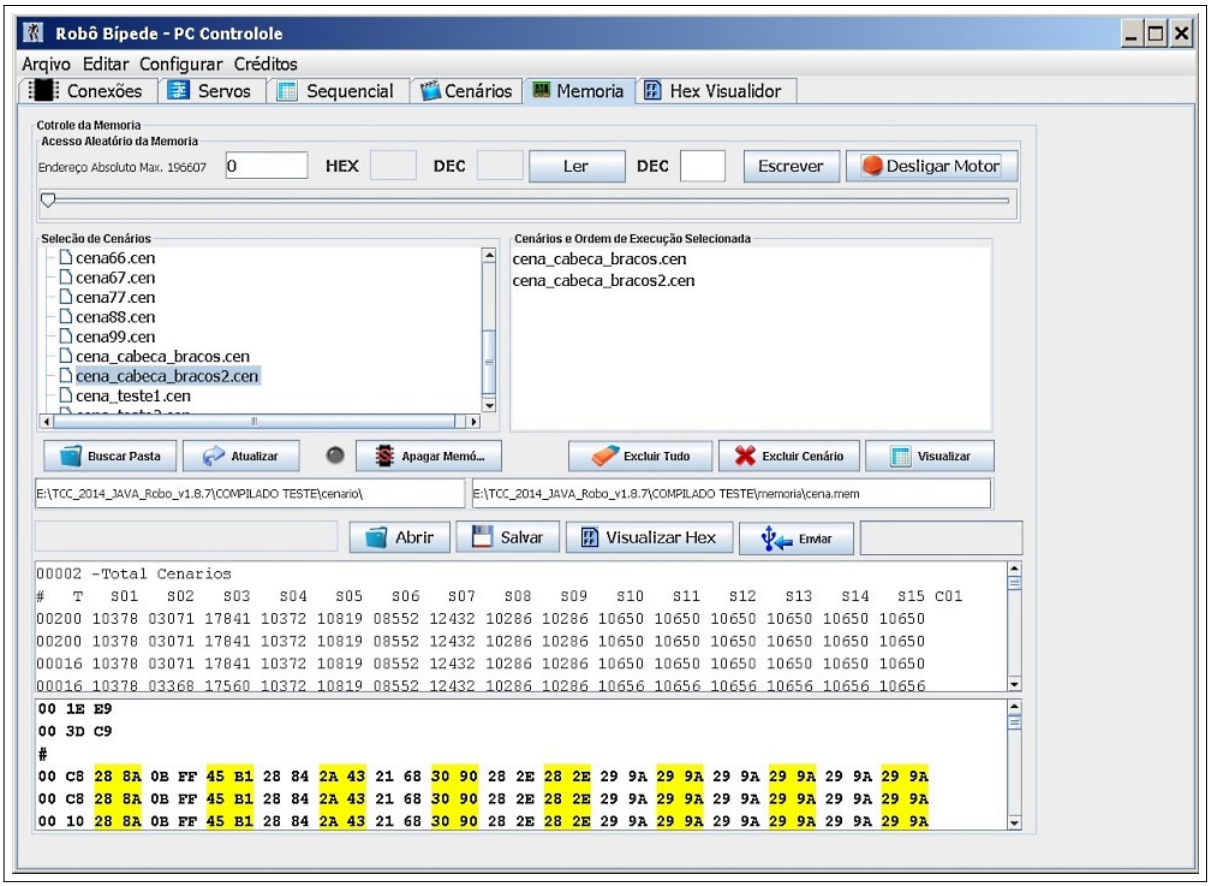

Figura 5.18: Software desenvolvido - Robô Bípede - PC Controle - Tela 5 Memória.

A **Figura 5.19** ilustra a sexta tela Hex Visualizador. Esta tela é utilizada para melhor visualizar os dados da tela Memória quando for apertado o botão “Visualizar Hex”.

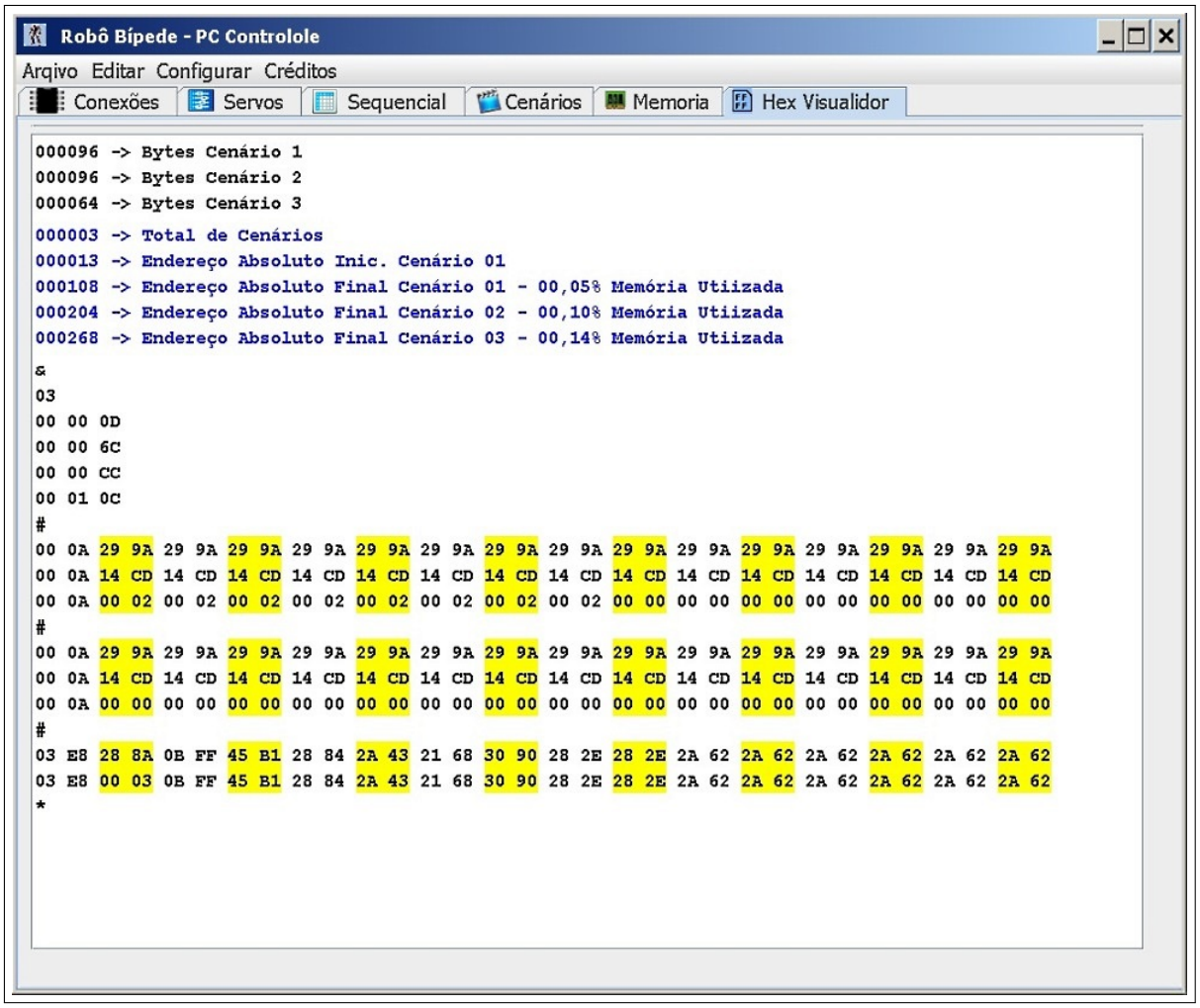

Figura 5.19: Software desenvolvido - Robô Bípede - PC Controle - Tela 6 Hex visualizador.

Finalizada a discussão sobre o sistema de *hardware*, *software* e mecânico, pode-se prosseguir ao capítulo 6 que definirá os movimentos de marcha, os trabalhos futuros e discutirá os resultados.

Capítulo 6

Testes

Os testes basicamente serão definidos pelo diagrama de tempo de acionamento dos motores para os movimentos iniciais do Robô Bípede, relatado no capitulo 3, conforme abordou Silva (2001). O diagrama serviu como base para separar os quadros de vídeo específicos do projeto do robô "PRIMER V5" de Yamaguchi (2013), trabalho intitulado "Robô para uma marcha natural tal como o ser humano".

Os quadros de vídeos referenciados pelo esquema do ciclo de marcha servirá como posição de referência para os pontos estáticos do movimento para o Robô Bípede e verificação do equilíbrio. Estes pontos serão interligados por um gerador linear de intervalo predefinido, garantindo que não se perca a estabilidade e o movimento fluído. Caso o intervalo não seja não seja satisfatório novos intervalos deverão ser adicionados para garantir os movimentos. Este processo é formado por três etapas:

- Posição inicial até o inicio do ciclo de marcha,
- Definição do movimento completo da marcha,
- Processo final da marcha até o ponto estático final em que o motor será desligado em uma posição segura, evitando quedas.

As análises são determinadas pela verificação dos comandos angulares e de tempo enviados aos servos motores, por meio de gráficos e fotos dos posicionamentos, bem como a avaliação do peso do sistema, movimento de marcha e problemas ocorridos durante os testes.

Para poder realizar os testes de movimentos iniciais, foi necessário construir uma plataforma onde o Robô Bípede pudesse permanecer suspenso, conforme ilustra a **Figura 6.1**. Por meio desta ferramenta, foi possível estudar os movimentos do robô durante os testes.

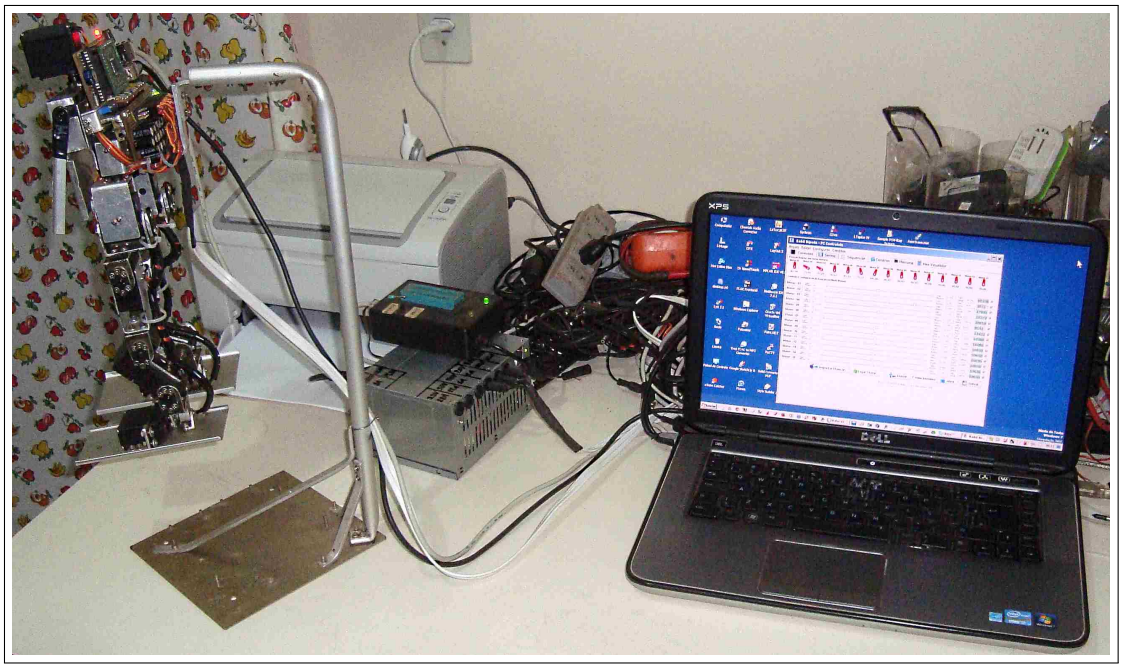

Figura 6.1: Bancada de testes com Robô Bípede fixado por plataforma.

Saber o peso final do robô foi de extrema importância na hora de testar as capacidades dos servo-motores, conforme ilustra a **Figura 6.2**.

O peso total atingido foi de 1.585 gramas. Na prática, o Robô Rípede não suporta muitos minutos em posições críticas, pois os servos irão esquentar e perder um pouco a força, tendo como consequência uma ligeira modificação do movimento. Este problema também é verificado quando a bateria vai descarregando.

Outro problema verificado durante o teste foi o fato dos pés do robô serem de alumínio. Este material fez com que o robô escorregasse muito no momento em que estivesse andando. Para corrigir este problema foi instalado uma borracha de câmara de moto, conforme ilustra a **Figura 6.3**.

Para gerar os movimentos do robô bípede foi preciso estudar os movimentos de um outro robô, o "PRIMER V5". Este robô também possui o formato bípede e é famoso por andar de bicicleta e imitar movimentos humanos com muita semelhança.

As **Figuras 6.4** e **6.5** mostram um comparativo do movimento de marcha em vista lateral do PRIME V5 e o robô bípede desenvolvido, respectivamente.

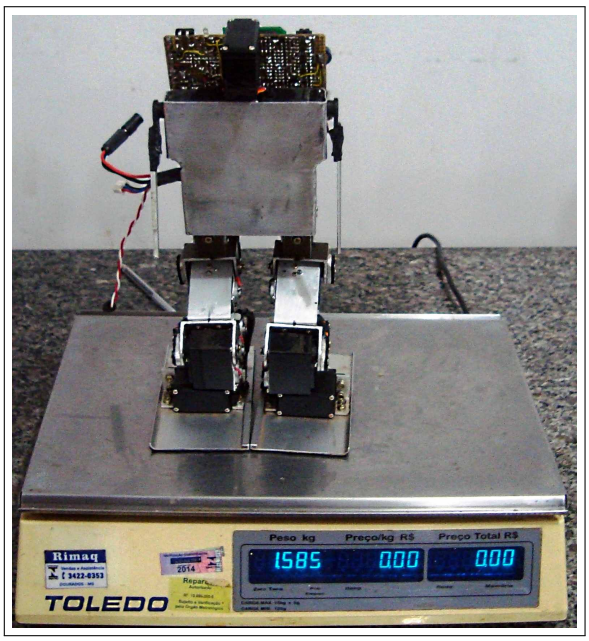

Figura 6.2: Peso total do Robô Bípede com bateria interna.

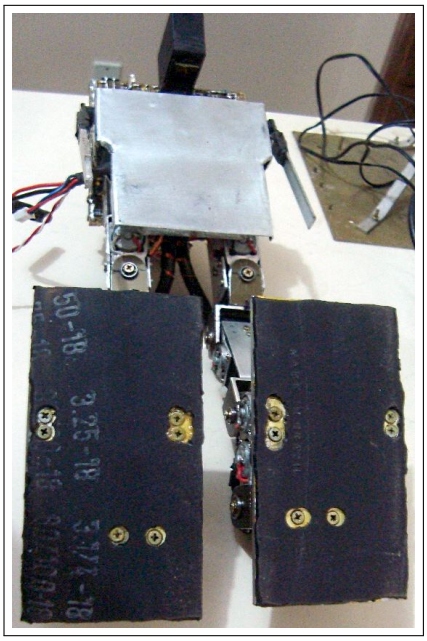

Figura 6.3: Instalação da borracha nos pés.

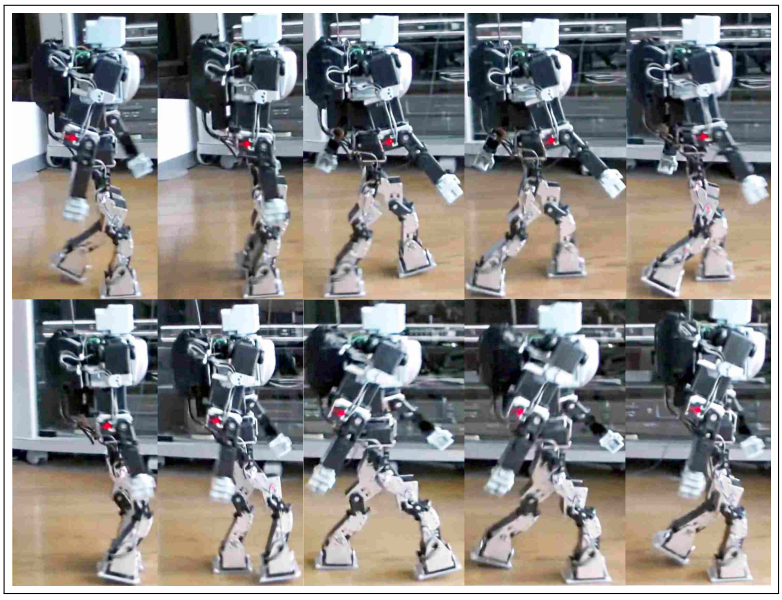

Figura 6.4: Vista lateral marcha do "PRIMER V5", imagens compostas de vídeo. Fonte: Yamaguchi (2013).

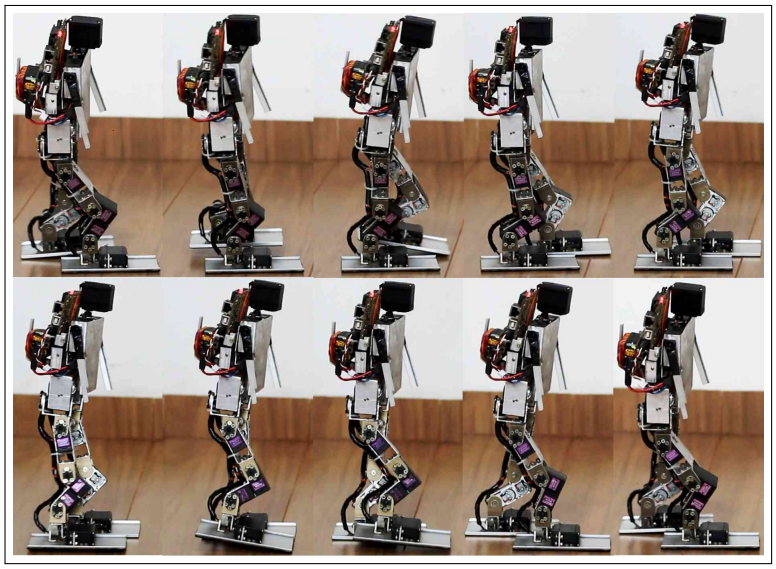

Figura 6.5: Vista lateral marcha do Robô Bípede.

As **Figuras 6.6** e **6.7**, mostram um comparativo do movimento de marcha em vista frontal do PRIME V5 e o robô bípede desenvolvido respectivamente.

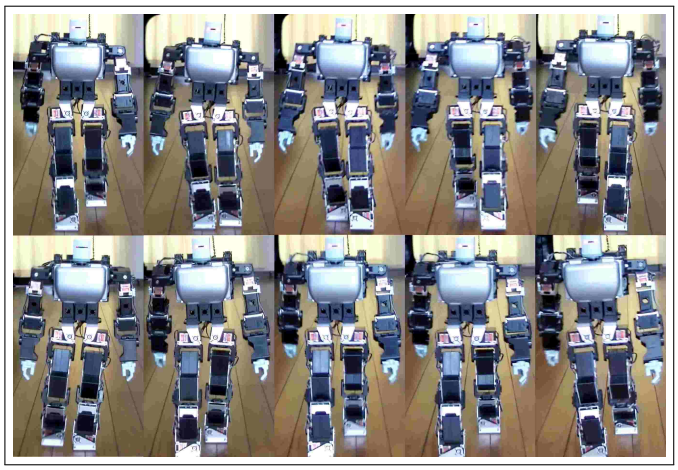

Figura 6.6: Vista frontal marcha do "PRIMER V5", imagens compostas de vídeo. Fonte: Yamaguchi (2013).

A **Figura 6.8** mostra o gráfico do movimento andar do robô bípede.

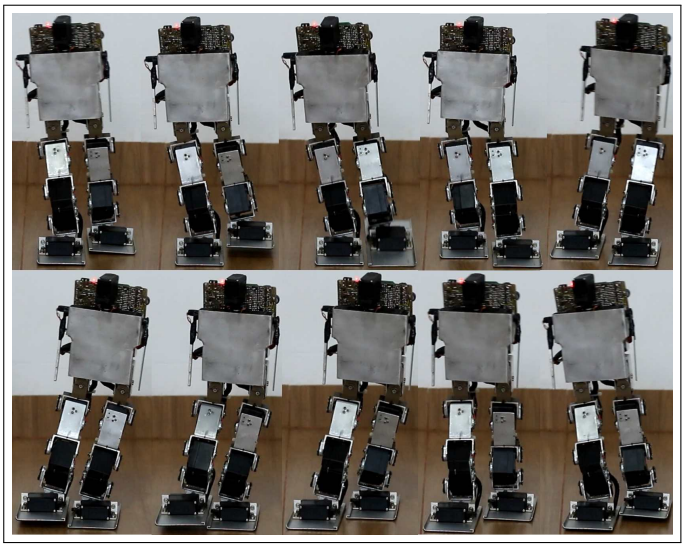

Figura 6.7: Vista frontal marcha do Robô Bípede.

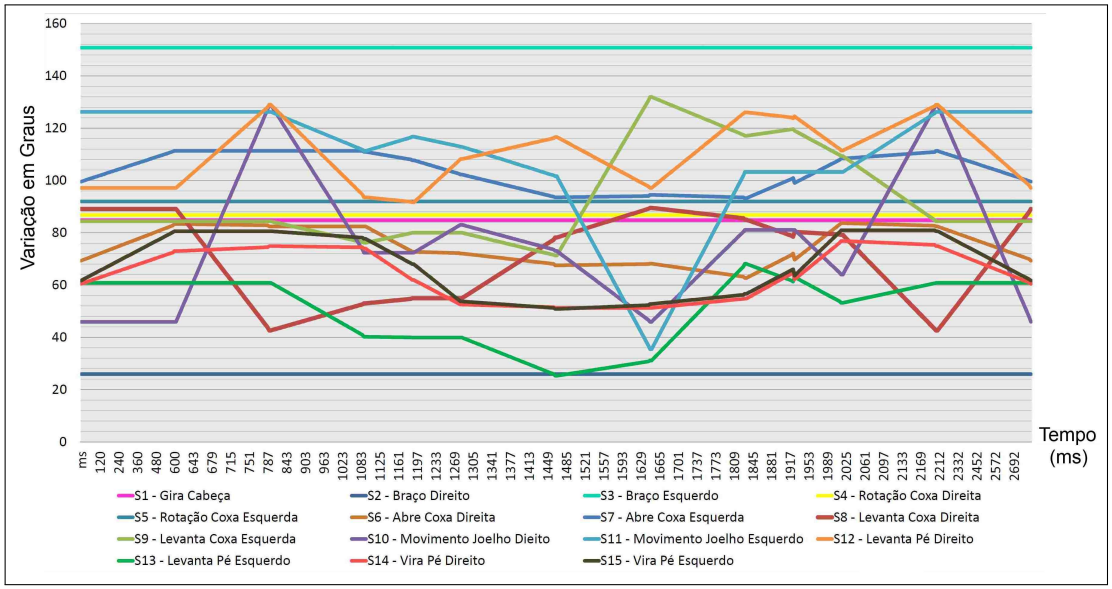

Figura 6.8: Gráfico do movimento do Robô Bípede.

O eixo vertical no gráfico da **Figura 6.8** corresponde aos ângulos absolutos dos servos, deste modo não correspondem ao valor angular ao eixo em um determinado membro do Robô.

O servo varia de 0 graus a 180 graus, este intervalo reduzido deve ser bem aproveitado. Verifica-se que o ponto mediano de 90 graus do servo nem sempre é o 90 graus do membro, um exemplo disto é o eixo do joelho quando o robô está em pé. Os gráficos mostram o posicionamento do sinal em que o processador gerou, por isto haverá um pequeno desvio mecânico correspondente.

Lembrando que existe uma pequena diferença entre o sinal enviado para o servo e seu posicionamento mecânico em virtude dos pequenos atrasos do processador, erros de *feedback* eletrônico, força mecânica exercida e jogo nas juntas, que irá compor um erro final.

Com a tecnologia dos servos utilizados, não é possível resolver os erros de posicionamentos, pois não possuem um *encoder* absoluto medindo precisamente o posicionamento. Assim tornou-se problemático estabilizar o Robô quando uma força mecânica em um eixo é muito grande, pois muda o posicionamento do servo. Esta instabilidade ocorre quando há aquecimento e quando há uma variação da tensão ocasionada pela alteração da carga da bateria.

Estas imprecisões muitas vezes levam ao desequilíbrio em um teste de sucesso, executado em um outro momento. Observa-se ainda uma oscilação nos momentos de parada brusca ou no esforço do calcanhar durante o agachamento, por exemplo.

No eixo horizontal temos o tempo em milissegundo para a execução das variações angulares. Importante observar que tanto o ponto inicial e o final do gráfico correspondem a posição estática em pé. Todos os servos retornam para a posição original no final da execução. O gráfico mostra apenas um único ciclo de marcha.

O gráfico da **Figura 6.9** apresenta os motores de "S6" até "S15", correspondentes ao movimento de andar do robô e demarca o processo da marcha. Caso seja necessário realizar mais passos, então será preciso repetir os valores do movimento de marcha seguidamente, pois o processo consiste em um ciclo. Inclusive seria possível fazer o passo de marcha invertido, se executar o comando reverso.

Os servos "S1-Cabeça", "S2-Braço Direito", "S3-Braço Esquerdo", "S4-Rotação Coxa Direita", "S5-Rotação Coxa Esquerda" não executaram nenhum movimento nos testes de marcha, deste modo não serão avaliados na **Figura 6.9** para maior legibilidade.

Um problema a salientar é o fato dos movimentos não serem semelhantes a um sistema senoidal, pois utilizou-se preenchimento dos intervalos dos pontos de forma linear. Tais sequências de passos podem ocasionar movimentos de desequilíbrio por representar uma velocidade constante próximos aos pontos de inversão dos movimentos. No sistema senoidal estes pontos teriam uma redução gradual próxima a inversão do movimento.

No eixo vertical temos os valores e subdivisões do pulso enviado ao servo que corresponderá aos valores angulares. Os pulsos são subdivididos de valores entre 0 e 21.300. O valor mediano de 10.650, corresponderá ao valor de 90 graus dos servos, quanto o 0 também será 0 graus e o 21.300 corresponde a 180 graus.

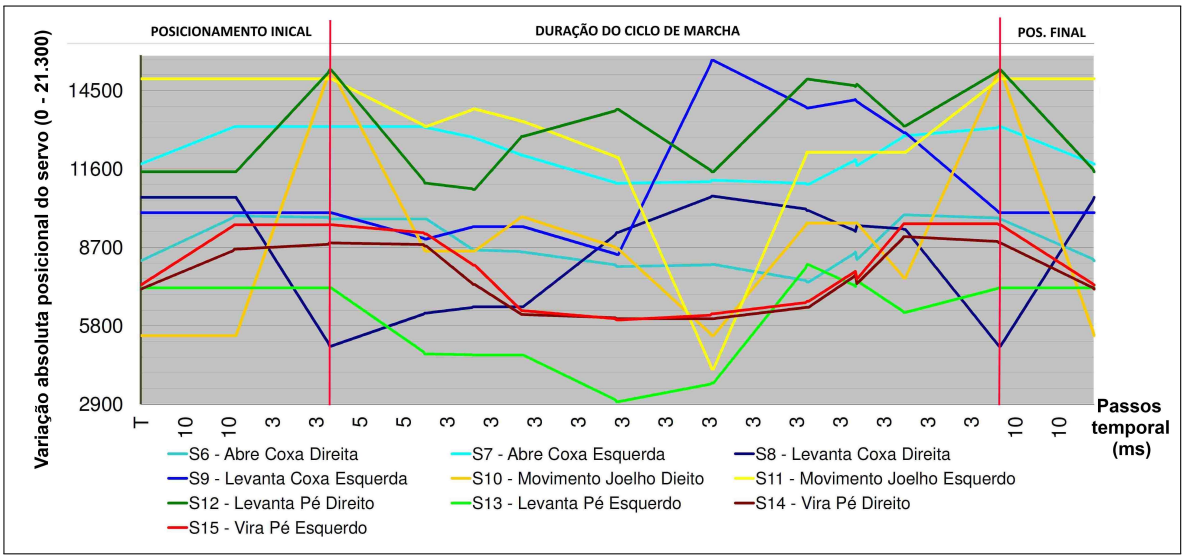

Figura 6.9: Gráfico do movimento andar do Robô Bípede.

Vale a pena ressaltar que alguns servos possuem os movimentos espelhados com relação ao eixo do membro como os calcanhares, joelhos, levanta coxas, vira coxas e braços. Este fato pode gerar uma confusão no momento de acionar estes servos, pois os valores numéricos são opostos entre o lado esquerdo e direito. Um exemplo de oposição de movimentos, são os servos "S14-Vira Pé Direito" e "S15-Vira Pé Esquerdo". Podemos ver no gráfico que as duas linhas de S14 e S15 tiveram um desenho muito parecido e próximas.

Quanto a linha horizontal do gráfico, temos os valores de tempo correspondente em cada variação gradual dos movimentos em milissegundos entre dois pontos. Quanto menor o tempo dessa granulação melhor será a fluidez do movimento, contudo será gasto muito mais memória e maior será o fluxo de dados. Se for escolhido muito mais do que 100 subdivisões entre dois pontos próximos, isto tenderá a deixar o sistema lento. Na prática deve-se testar diversas divisões até conseguir um movimento aceitável, a fim de reduzir gastos com memória e troca de dados. O número de divisões ou passos influenciará no desempenho da velocidade do servo, quanto menor for as subdivisões maior será a velocidade.

O gráfico é subdivido em três partes conforme relatado anteriormente. No gráfico da **Figura 6.9** estas divisões estão demarcadas pelas linhas verticais em vermelho. Primei-

ramente é necessário manter um período de funcionamento do Bípede em pé parado para permitir o manuseio inicial. Logo em seguida o movimento irá para o início do processo de marcha encerrando a posição inicial.

Na segunda parte temos o ciclo de marcha. Para os testes práticos foram testados cinco ciclos sucessivos antes de atingir a posição final. Verifica-se ainda que cada ponta formada no gráfico corresponde aos pontos estáticos escolhidos em um momento dos quadros de movimento.

No terceiro e último estágio verificou a transição do fim da marcha para ao posição final em pé. É necessário manter os servos acionados por um tempo para que seja segurado, caso contrário o Robô poderá cair devido ao seu desligamento. No modo prático foi adicionado uma rotina final de agachamento para que fique em pé mesmo que não esteja sendo segurado.

O tamanho dos pés do Robô Bípede limitou o movimento de caminhada. Devido as reduzidas dimensões das pernas não foi possível manter os pés consideravelmente elevados. As imprecisões dos servos ocasionavam enroscadas com os menores obstáculos e até mesmo entre os pés, algumas vezes.

6.1 Trabalhos Futuros

A alteração iniciais será inserção do módulo "ZigBee", que consiste em um sistema sem fio *wireless* com base no protocolo "IEEE 802.15.4" e transmite dados serial a uma distância considerável. Para o modelo "XBee PRO S1" adquirido, podendo alcançar em campo aberto até 1500 metros. Segue na **Figura 6.10** a imagem dos módulos.

No sistema sem fio, basicamente seria utilizado um sistema de controle pelo computador pessoal por meio de um outro microcontrolador intermediando o *software* existente e o "ZigBee". Deste modo evitaria as mudanças *software*, bastando o microcontrolador converter os dados da USB para serial e assim enviar ao robô. Quanto ao robô, iria agrupar aos dados da serial em um pacote de 64 Bytes e encaminharia ao decodificador já existente do pacote USB, reduzindo assim as alterações do código do Robô.

A utilização de um rádio de areomodelismo em conjunto com outro micro controlador e um outro "ZigBee", permite realizar comandos simples de movimentos, por meio da porta traseira para treinamento do rádio, conforme mostrado na **Figura 6.11**.

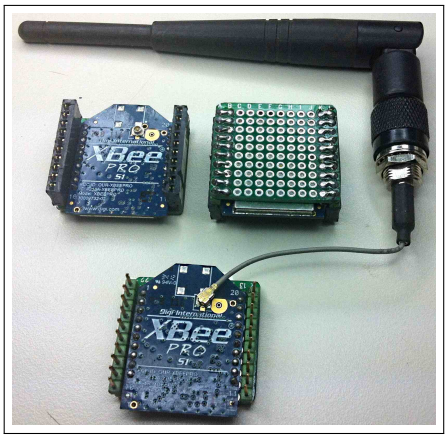

Figura 6.10: ZigBee adquirido.

Figura 6.11: Rádio aeromodelismo e o ponto de treino.

Para trabalhar com movimentos dinâmicos em um nível mais avançado propõem:

- Redução do tamanho dos pés,
- Inserção de módulos como sensor ultrassônico para detectar objetos,
- Giroscópios para o equilíbrio,
- Utilizar microprocessadores de 32 bits,
- Tela de cristal líquido para gerar expressão do rosto do robô,
- Completar os braços.

Importante ainda destacar que este sistema de controle dos servomotores poderia ser utilizado para outras configurações ou tipos de Robôs.

Capítulo 7

Conclusão

A construção do protótipo de robô bípede se mostrou viável apesar das limitações de orçamento e do alto nível de complexidade nas áreas da mecânica, eletrônica e da programação. Para o desenvolvimento do projeto foi necessária a aquisição de diversas peças importadas como alternativa para minimizar os custos, tais como: o microcontrolador, servomotores, osciladores e memórias, entre outros. A necessidade destas aquisições dificultou o andamento do projeto pelo atraso decorrente do tramite aduaneiro, o cálculo e recolhimento dos tributos, bem como a liberação das peças e materiais pelos órgãos competentes.

Durante o desenvolvimento do projeto, os desafios propiciaram o aperfeiçoamento das técnicas para a elaboração de placas e circuitos eletrônicos, fiação, sistemas contra interferências e ruídos. Também favoreceu o desenvolvimento de uma fonte de 7,0 Volts e 15 Amperes com proteção de sobretensão e de corrente a partir de uma fonte de computador convencional, que atendesse as necessidades específicas.

De uma forma abrangente, a determinação da oscilação do centro de massa do robô durante sua caminhada foi realizada de modo empírico, ou seja, por tentativa e erro. A simplificação de montagem, permitiu facilitar a execução do projeto do protótipo, ao evitar a inclusão de sensor gravitacional e giroscópio, que possuem alto custo no mercado nacional, além do alto nível de complexidade para sua programação e processamento via microcontrolador PIC.

Outro fator que simplificou a montagem foi a conversão prévia dos dados a serem embarcados no microcontrolador em hexadecimal realizada pelo microcomputador em Java. A conversão prévia dos valores dos dados reduziu o número de memórias externas. O projeto pode ser aprimorado com a adição do sistema sem fio como o "ZigBee" que permitirá uma melhor mobilidade durante o controle, além de adicionar controle via rádio.

A utilização de equações matemáticas específicos do movimentos bípedes e simuladores de computação gráfica, irá melhorar a naturalidade dos movimentos. Para um nível

mais avançado o projeto pode ser estendido para o de um humanóide completo com controle dinâmico dos movimentos. Para isto, é recomendado um microprocessador com maior capacidade e a utilização de sensores como os módulos ultrassônico e giroscópio. A utilização de sistema de som e tela de cristal líquido para o rosto, tornariam-no em um Robô mais social.

Apêndice A

Firmware - Código Principal

Segue o código principal A.1 em linguegem “C” do robô bípede:

```
1
2  //SERB.c
3  //Software Embarcado Robô Bípede V01.0.7
4  //Autor: Esmael Dias Prado, 2015, zz9abi@gmail.com
5
6  #include <18F4550.h>    //Inclusão das definições do microcontrolador utilizado.
7  #device adc=10                  //Define a entrada analógia de 10bit's.
8
9  //Configuração dos fuses interno do PIC18F4553.
10 #fuses HSPLL        //Cristal ou ressonador externo de alta frequência maior que 4MHz
       juntamente
11                     //com o PLLDIV ativado, cristal utilizado 20MHz.
12 #fuses PLL5         //Ativa o PLL com divisor por 5 na frequêcnia do oscilador HSPLL,
       assim 20MHz/5 = 4MHz;
13                                      //o 4Mhz é padrão para o PLL multiplicar a
                                            frequência e gerar 96MHz.
14                                      //Para melhor entendimento veja "Datasheet Clock
                                            Diagram".
15 #fuses USBDIV       //Divide 96MHz/2 gerando 48Mhz necessário, para o funcionaemnto da
       USB.
16 #fuses VREGEN       //Ativa a voltagem regulada da USB.
17 #fuses CPUDIV1      //Divide 96MHz/2 para gerar 48MHz para a CPU (12 MIPS).
18 #fuses PUT          //Ativa o oscilador temporizador "start-up" (OST) "Power Up Timer",
19                     //gera um atraso de 72ms para estabilizar o oscilador na energização.
20 #FUSES BORV46       //Reseta o processador se tensão cair a menos de 4.6V, "Brownout
       reset".
21 #fuses MCLR                      //Ativa o pino /MCLR-RC3 "Master Clear" (Pino para
       RESETAR a CPU).
22
23 #fuses NOWDT             //Desativa o "Watch Dog Timer".
24 #fuses NOLVP             //Desativa a programação de baixa voltagem.
25 #fuses NODEBUG      //Desativa o Debug em modo ICD.
26 #fuses NOFCMEN      //Desativa o monitor contra falha de Relógio (Relógio interno RTC não
        utilizado),
27                     //"Fail-safe clock monitor disabled".
28 #fuses NOPROTECT    //Desativa a proteção de leitura de código de programação.
29 #fuses NOCPD        //Desativa a proteção de leitura da EEPROM.
30 #fuses NOCPB        //Desativa a proteção de leitura do código do bloco de inicialização,
```

```
     "Boot block".
#fuses NOWRT         //Desativa a protecão de escrita na memória de programa.
#fuses NOWRTD        //Desativa a proteção de escrita de dados na EEPROM.
#fuses NOWRTB        //Desativa a proteção de escrita no bloco de inicialização, "Boot
    block".
#fuses NOWRTC        //Desativa a proteção de escrita nos registradores.
#fuses NOEBTRB       //Desativa a proteção do bloco de inicalização "Boot block" em outras
     tabelas de leitura.
#fuses NOEBTR        //Desativa a proteção da memória em outras tabelas de leitura.

#use delay(clock=48000000)

#include <stdio.h>   // para o printf(), atoi(), itoa();
#include <stdlib.h>
#include "servo_init.h"

#include <AT24C512.h>
//#include <PCF8583_RTC.h>

//Tipos de comunicação selecionável
const int USB_DIRETO = 0;
const int USB_SERIAL = 1;

//ABILITA O TIPO DE COMUNICAÇÃO PRETENDIDA
const int ATIVA = USB_DIRETO;

int LIG_DESL_SERVOS = 0;
int LIG_DESL_SERVOS1 = 0;

//Média da leitura de tensao
int8 buffer_tensao[50];
int cont_media_tensao=0;

//*************************************************************************
#IF ATIVA == USB_DIRETO
          #include "pacote_USB_direto.h" //Se utilizar esta biblioteca não utilize a
              biblioteca pacote_USB_serial.h
#ENDIF

#IF ATIVA == USB_SERIAL
          #include "pacote_USB_serial.h" //Se utilizar esta biblioteca não utilize a
              biblioteca pacote_USB_direto.h
#ENDIF
//*************************************************************************

//*************************************************************************

//-------------------------------------------------------------------------
//Timer 1
#define TIMER1OFFSET 65534
//16 bit = 65536
// setup_timer_1(RTCC_INTERNAL|RTCC_DIV_1);  //48MHz/4/1 = 12MHz  = 0,08333us
//-------------------------------------------------------------------------

//Timer 3                                                                  //c = 1
    ms
#define TIMER3OFFSET 53550

```

```
83  int8 x=0;                                                    //Conta varias
        interrupcoes CCP1 sem pulso
84  //boolean led1=0;
85  int16 recebe_tempo_t1=0;
86  int16 recebe_tempo_t3=0;
87  int16 cont_temp1=0;
88  int16 cont_temp2=0;
89
90  #include "servo.h"
91
92  //*****************************************************************************
93  //Mandar p/ a variável global a inicialização dos dados, para executar os cenários
94  void inicia_valor_memoria(){
95          int   i;
96          int   cont=0;
97          int8  dado[128];
98          int32 tamanho_endereco;
99
100         for(i=0; i<= 30; i++){
101                 endereco_memoria[i]=0;
102         }
103
104         //Marcando a variável Global com o valor do total de cenários
105         //Este valor sempre estará na poição zero e tem o tamanho de um Byte (0-255)
106         //BYTE read_ext_eeprom(long int address, int dispositivo)
107         //Não foi previsto que o total de enderecamento de cenário ultrapasse o banco 1
                de
108         //memoria pois seria incapaz de caber os cenários nos demais 2 bancos de memoria.
109         //E para economizar memoria está definido 128 Bytes de edereçamentos
110         //Totalizando um total de (128/3) - 1 = 41 cenários possíveis
111         //Mais devido a limitações de memoria do pic será limitande uns 30 cenarios
112
113         total_cenario = read_ext_eeprom(0, 1);  //função retorna um byte, total de
                cenários
114
115         if(total_cenario!=0){
116                 tamanho_endereco = 3 * (total_cenario + 1);
117
118                 //Leitura dos endereção de cada cenário são composto de 3 Bytes p/ cada
                        endereçamento,
119                 //para isso será buscado a partir do endereço (1 até ((3 x Byte) x (
                        total_cenario + 1))
120                 //Ex:
121                 //End. 0x0000 = Total de cenário
122                 //End. 0x0001 = Byte alto do endereço inicial do cenário 1
123                 //End. 0x0002 = Byte do meio do endereço inicial do cenário 1
124                 //End. 0x0003 = Byte mais baixo do endreço inicial do cenario 1
125                 //
126                 //End. 0x0004 = Byte alto do endereço final do cenário 1
127                 //End. 0x0005 = Byte do meio do endereço final do cenário 1
128                 //End. 0x0006 = Byte mais baixo do endreço final do cenario 1
129                 //
130                 //End. 0x0007 = Byte alto do endereço final do cenário 2
131                 //End. 0x0008 = Byte do meio do endereço final do cenário 2
132                 //End. 0x0009 = Byte mais baixo do endreço final do cenario 2
133                 //... assim por diante
134                 //read_ext_eeprom_sequencial(long int address, BYTE data[], int
                        tamanho_data, int dispositivo)
```

```
135                 //Lendo a memória extern a a partir do endereço 1, no banco 1 de memoria
136                 read_ext_eeprom_sequencial(1, dado, tamanho_endereco, 1);
137
138                 // tamanho_endereco = (0 até tamanho_endereco-1)
139                 for (i=0;i<tamanho_endereco; i+=3){
140                         endereco_memoria[cont] = make32(dado[i],dado[i+1],dado[i+2]);
141                         cont++;
142                 }
143         }
144 }
145 //*****************************************************************************
146
147 //*****************************************************************************
148 int16 executa_cenario_memoria_local(void){
149
150         int16 tempo=1;
151         //t   i;
152         int   seleciona_men;
153         int   tamanho_endereco=32;
154         int8  dado[33];
155         int1  flag = 0;
156         int16 temp;
157
158         //total_cenario
159         if(contagem_endereco_mem==0){
160                 contagem_endereco_mem = endereco_memoria[0];
161         }
162
163         //Estouro de memoria >196.607
164         if( contagem_endereco_mem <= 196607 ){            //Se o endereço for menor ou
                igual a 196607 entra, endereço válido.
165                 seleciona_men = 1;                                                  //
                    Selecione o banco 1 de memória
166                 if(contagem_endereco_mem>65535){                  //Se for maior que
                    entre (0 e 65535), então banco 2 apartir de 65536;
167                         seleciona_men = 2;                                          //
                            Selecione o banco 2 de memória
168                         if(contagem_endereco_mem>131071){         //Se for maior que
                            entre (65536 e 131071 ), então banco 3 apartir de 131072;
169                                 seleciona_men = 3;                                  //
                                    Selecione o banco 3 de memória
170                                 if(contagem_endereco_mem>196607){ //Se for maior que
                                    entre (131071 e 196607), então banco 4 apartir de
                                    196608;
171                                         seleciona_men = 1;                //Selecione o
                                            banco 1 de memória pois o banco 4 não exite.
                                            há um erro.
172                                 }
173                         }
174                 }
175
176                 //read_ext_eeprom_sequencial(long int address, BYTE data[], int
                    tamanho_data, int dispositivo)
177                 if( contagem_endereco_mem <= endereco_memoria[1] ){
178
179                         read_ext_eeprom_sequencial(contagem_endereco_mem, dado,
                            tamanho_endereco, seleciona_men);
180                         contagem_endereco_mem = contagem_endereco_mem+32;
```

```
181
182                         tempo  = make16(dado[0],dado[1]);
183
184                         //Motor 1 = limite min 130 + 7900 = 8030,   Limite Max = 21100 +
                                7900 = 29000
185                         temp = make16(dado[2],dado[3]);
186                         if(temp<130){temp=130;}
187                         if(temp>20920){temp=20920;}
188                         servo1 = temp+7900;
189
190                         //Motor 2 = limite min 130 + 7900 = 8030,   Limite Max = 20360 +
                                7900 = 28260
191                         temp = make16(dado[4],dado[5]);
192                         if(temp<130){temp=130;}
193                         if(temp>20360){temp=20360;}
194                         servo2 = temp+7900;
195
196                         //Motor 3 = limite min 130 + 7900 = 8030,   Limite Max = 20360 +
                                7900 = 28260
197                         temp = make16(dado[6],dado[7]);
198                         if(temp<130){temp=130;}
199                         if(temp>20360){temp=20360;}
200                         servo3 = temp+7900;
201
202                         //Motor 4 = limite min 600 + 7900 = 8500,   Limite Max = 20900 +
                                7900 = 28800
203                         temp = make16(dado[8],dado[9]);
204                         if(temp<600){temp=600;}
205                         if(temp>20900){temp=20900;}
206                         servo4 = temp+7900;
207
208                         //Motor 5 = limite min 600 + 7900 = 8500,   Limite Max = 20900 +
                                7900 = 28800
209                         temp = make16(dado[10],dado[11]);
210                         if(temp<600){temp=600;}
211                         if(temp>20900){temp=20900;}
212                         servo5 = temp+7900;
213
214                         //Motor 6 = limite min 1127 + 7900 = 8500,   Limite Max = 18450 +
                                 7900 = 26350
215                         temp = make16(dado[12],dado[13]);
216                         if(temp<1127){temp=1127;}
217                         if(temp>18450){temp=18450;}
218                         servo6 = temp+7900;
219
220                         //Motor 7
221                         temp = make16(dado[14],dado[15]);
222                         servo7 = temp+7900;
223
224                         //Motor 8
225                         temp = make16(dado[16],dado[17]);
226                         servo8 = temp+7900;
227
228                         //Motor 9
229                         temp = make16(dado[18],dado[19]);
230                         servo9 = temp+7900;
231
232                         //Motor 10
```

```
233                         temp = make16(dado[20],dado[21]);
234                         servo10 = temp+7900;
235
236                         //Motor 11
237                         temp = make16(dado[22],dado[23]);
238                         servo11 = temp+7900;
239
240                         //Motor 12
241                         temp = make16(dado[24],dado[25]);
242                         servo12 = temp+7900;
243
244                         //Motor 13
245                         temp = make16(dado[26],dado[27]);
246                         servo13 = temp+7900;
247
248                         //Motor 14
249                         temp = make16(dado[28],dado[29]);
250                         servo14 = temp+7900;
251
252                         //Motor 15
253                         temp = make16(dado[30],dado[31]);
254                         servo15 = temp+7900;
255
256                 }else{
257                         if( contagem_endereco_mem-2 <= endereco_memoria[1] ){
258                                 tempo=1000;
259                                 contagem_endereco_mem+=2;
260                         }else{
261                                 tempo=5;
262                                 LIG_DESL_SERVOS=0;
263                                 LIG_DESL_SERVOS1=0;
264                                 contagem_endereco_mem=0;
265                         }
266                 }
267
268         }else{
269                 //estoutrou o limite da memoria
270                 //faz nada retorna tempo = 1;
271         }
272         return tempo*3;    //x3 original
273 }
274 //***************************************************************************
275
276 //***************************************************************************
277 void verifica_bateria(void){
278         float ad1;
279         int i;
280         int16 media=0;
281
282     set_adc_channel(1);
283         delay_us(80);
284         ad1=read_adc();
285         ad1=0.2466258*ad1;
286
287         if( cont_media_tensao >= 50){
288
289                 for(i=0;i < 50; i++){
290                         media = media + buffer_tensao[i];
```

```
291                 }
292                 media = media / 50;
293
294                 if( media < 70){
295                         while(1){
296                                 output_high(pin_a2);
297                                 delay_ms(500);
298                                 output_low(pin_a2);
299                                 delay_ms(500);
300                         }
301                 }
302                 cont_media_tensao=0;
303         }else{
304
305                 buffer_tensao[cont_media_tensao] = ad1;
306                 cont_media_tensao++;
307         }
308 }
309 //***************************************************************************
310
311 //***************************************************************************
312 void rotina_1_ms(void){
313
314         cont_temp1++;
315         cont_temp2++;
316
317         if(LIG_DESL_SERVOS1==1){
318                 if(cont_temp1>=cont_geral1){
319
320                         led1 = !led1;
321                         output_bit(pin_a2,led1);
322                         cont_temp1=0;
323
324                         cont_geral1 = executa_cenario_memoria_local();
325                 }
326         }
327
328         //executa a cada 20ms
329         if(cont_temp2>=cont_geral2){
330         verifica_bateria();
331                 cont_temp2=0;
332                 cont_geral2 = 20;
333         }
334 }
335 //***************************************************************************
336
337 //***************************************************************************
338 void main(){
339     int16 w=0;
340
341    //Configura as porta analogicas Pino P19/RA0/AN0 e P20/RA1/AN1
342    setup_ADC_ports (AN0_TO_AN1_ANALOG);
343    setup_adc(ADC_CLOCK_INTERNAL );
344
345    setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
346    set_timer1(TIMER1OFFSET);
347    setup_timer_0(RTCC_INTERNAL|RTCC_8_BIT|RTCC_DIV_64);
348    set_timer0(TIMER3OFFSET);
```

```
349
350     enable_interrupts(INT_TIMER1);
351     enable_interrupts(INT_TIMER0);
352
353     for(w=0;w<5;w++){
354                 output_high(pin_a2);
355                 //output_low(pin_a3);
356                 delay_ms(30);
357                 output_low(pin_a2);
358                 //output_high(pin_a3);
359                 delay_ms(180);
360     }
361     output_low(pin_a3);
362
363     #IF ATIVA == USB_DIRETO
364          //Inicialização da porta usb direta-----------------------------------------
365              usb_init(); //inicializamos el USB
366                  usb_task(); //Se encarga de mantener el  sentido de la comunicación,
                        llama a usb_detach() yusb_attach() cuando se necesita
367
368                  //Se o botão estiver apertado salte a enumeração da porata USB p/
                        Funcionar sem cabo.
369                  if(false!=input(pin_c1))                          //Verifique se o Botão
                        foi precionado nivel '0'.
370                  {
371                          usb_wait_for_enumeration();       // Esperamos hasta que el
                                PicUSB sea configurado por el host
372                  }else{
373                          delay_ms(2000);
374                  }
375                  //
                        ------------------------------------------------------------------------------

376     #ENDIF
377
378     #IF ATIVA == USB_SERIAL
379                  //Inicialização da porta serial
                        ------------------------------------------------
380                  usb_cdc_init();
381                  usb_init();
382                  //
                        ------------------------------------------------------------------------------

383     #ENDIF
384
385     enable_interrupts(global); // Habilitamos todas las interrupciones
386
387     output_b(0);
388     output_d(0);
389     output_e(0);
390     init_ext_eeprom();
391     inicia_valor_memoria();
392     LIG_DESL_SERVOS=0;
393     LIG_DESL_SERVOS1=0;
394
395   while(true)
396   {
397         #IF ATIVA == USB_DIRETO
```

```
398                 recebe_pacote_usb();
399         #ENDIF
400
401         #IF ATIVA == USB_SERIAL
402                 gets_usb_serial();
403         #ENDIF
404
405         if(false==input(pin_c1))                        //Verifique se o Botão foi
                precionado nivel '0'.
406         {
407                 inicia_valor_memoria();           //Inicializa o cabeçalho da memória
                    total de cenário e endereços
408
409                 LIG_DESL_SERVOS=1;                //Liga Servos
410                 LIG_DESL_SERVOS1=1;
411                 delay_us(200);                                  //Tempo para Debouncing
                    , para evitar ruídos de chave.
412                 while(false==input(pin_C1)){}     //Espere o botão ser solto.
413         }
414  }//while
415 }
416 //***************************************************************************
417
418 //***************************************************************************
419 #int_TIMER1
420 void TIMER1_isr()
421 {
422         if(x == 2){     // Define o tempo desligado, Ciclo para x: 1=10.8ms, 2=16.4ms,
                3=22.8ms, 4=26ms
423       if( LIG_DESL_SERVOS == 1 ){
424         CommandServos();
425           }
426           x=0;
427     }
428         else{
429       x++;
430     }
431 }
432 //***************************************************************************
433
434 //***************************************************************************
435 #int_TIMER0
436 void TIMER0_isr()
437 {
438           set_timer0(120);
439           rotina_1_ms();
440 }
441 //***************************************************************************
```

Código A.1: Função Principal PIC - Código C

Referências Bibliográficas

Araújo Neto, J. G. d. (2013). *Midiatização da inovação científica: estratégias do Instituto Internacional de Neurociências de Natal/RN pela intervenção do ator cientista (Natal/RN)*. PhD thesis, Universidade do Vale do Rio dos Sinos, Programa de Pós-graduação em Ciências da Comunicação, São Leopoldo-RS. 251 f. (p. 224-227). Disponível em : <`http://biblioteca.asav.org.br/vinculos/000003/00000305.pdf`> Acesso em : 6 Outubro. 2015.

Asimov, I. (1969). *Eu, Robô*. Ediouro, 2 edition. [tradução de Luiz Horácio da Matta].

Banakara, B. (2013). Analysis by simulation of the over-voltages in PWM-inverter fed induction motors. *International Journal of Electrical Energy*, 1(1):29–33.

Biscaro, M. (2010). Tablemodel. Disponível On-line em 15 Maio de 2015 em <`http://www.guj.com.br/java/149034-duvidas-ao-fazer-uma-consulta`>.

Boylestad, R. L. and Nashelsky, L. (2004). *Dispositivos eletrônicos e teoria de circuitos*. Prentice-Hall do Brasil, 8 edition. 672 págs. ISBN: 85-87918-22-2.

Citengine, E. (2015). Iron 'ElectriRx' Man: Overground Stepping in an Exoskeleton Combined with Noninvasive Spinal Cord Stimulation after Paralysis. Disponível On-line em 30 Setembro de 2015 na URL `http://emb.citengine.com/event/embc-2015/paper-details?pdID=4403`.

Demasi, D. (2012). Modelagem dinâmica e de controle de um mecanismo de três graus de liberdade para aplicação em um robô hexápode. Dissertação (mestrado), Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET, Rio de Janeiro-RJ.

Dorf, Richard C e BISHOP, R. H. (2001). *Sistemas de controle Modernos*. LTC - Livros Técnicos e Científicos S.A., Rio de Janeiro, RJ, 8 edition.

Duffy, B. R. (2003). Anthropomorphism and the social robot. *Robotics and Autonomous Systems*, 42:177–190. DOI: 10.1016/S0921-8890(02)00374-3.

Fitzsimmons, N. A., Lebedev, M. A., Peikon, I. D., and Nicolelis, M. A. (2009). Extracting kinematic parameters for monkey bipedal walking from cortical neuronal ensemble activity. *Frontiers in integrative neuroscience*, 3(3):1–19.

Futaba (2015). Digital fet servos. Disponível On-line em 03 Outubro de 2015 na URL <`http://www.futaba-rc.com/servos/digitalservos.pdf`>.

Gerasimenko, Y. P., Lu, D. C., Modaber, M., Zdunowski, S., Gad, P., Sayenko, D. G., Morikawa, E., Haakana, P., Ferguson, A. R., Roy, R. R., et al. (2015). Noninvasive reactivation of motor descending control after paralysis. *Journal of neurotrauma*, 32:1–13. `http://online.liebertpub.com/doi/pdf/10.1089/neu.2015.4008`.

Hirose, M. and Ogawa, K. (2007). Honda humanoid robots development. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 365(1850):11–19.

Honda, M. C. L. (2007). Honda-asimo-technical information-the honda humanoid robot. Disponível On-line em 03 Setembro de 2015 na URL <`http://asimo.honda.com/downloads/pdf/asimo-technical-information.pdf`>.

Honda, M. C. L. (2015). Why Create a Humanoid Robot? Disponível On-line em 30 Setembro de 2015 na URL `http://asimo.honda.com/asimo-history/`.

Iida, F., Minekawa, Y., Rummel, J., and Seyfarth, A. (2009). Toward a human-like biped robot with compliant legs. *Robotics and Autonomous Systems*, 57(2):139 – 144. Selected papers from 9th International Conference on Intelligent Autonomous Systems (IAS-9)9th International Conference on Intelligent Autonomous Systems. doi: http://dx.doi.org/10.1016/j.robot.2007.12.001.

Katic, Dusko, M. V. (2002). Intelligent soft-computing paradigms for humanoid robots. *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, 3:2533–2538. 0-7803-7398-7.

LimoncelloDigital (2011). Comparativa: Kits robots bipedos comerciales. Disponível On-line em 04 Outubro de 2015 na URL <`http://www.limoncellodigital.com/2011/06/comparativa-kits-robots-bipedos.html`>.

Luz, R. P. D. D. (2003). CONSTRUÇÃO DE UM BRAÇO MECATRÔNICO E SOFTWARE PARA CONTROLE DO MESMO. Trabalho de Conclusão de Curso.

McCOMB, G. (2001). *THE ROBOT BUILDER'S BONANZA*. McGraw-Hill, 2 edition. [99 Inexpensive Robotics Projects].

Microchip (2009). Microchip PIC18F2455/2550/4455/4550 Data Sheet. Disponível On-line em 30 Setembro de 2015 na URL <http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf>.

MikroElektronika (2015). El mundo de los microcontroladores-microcontroladores pic-programación en c con ejemplos. Disponível On-line em 03 Outubro de 2015 na URL <http://www.mikroe.com/chapters/view/79/capitulo-1-el-mundo-de-los-microcontroladores/>.

Noleto, T. H. B. and Siqueira, J. A. (2007). Aferição da pressão arterial utilizando Wireless. Trabalho de Conclusão de Curso.

Oñativa, G. I. (2009). jpicusb: Clase java para comunicación usb con pics usando api de microchip. Universidad Nacional de Tucumán - Facultad de Ciencias Exactas - Ingeniería en Computación. Disponível On-line em 15 Setembro de 2015 em <http://www.edutecne.utn.edu.ar/microcontrol_congr/microcontrol_congr\discretionary{-}{}{}trabajos.html>.

Oñativa, G. I. (2013). jpicusb 1.1.1. Disponível On-line em 15 Abril de 2015 em <http://sourceforge.net/projects/jpicusb/>.

Oliveira, A. S. d. and Andrade, F. S. d. (2006). *SISTEMAS EMBARCADOS*. Érica Ltda, 1 edition. [Hardware e Firmware na Prática].

PyroElectro.com (2012). Servo motor control: The servo motor. Disponível On-line em 03 Outubro de 2015 na URL <http://www.pyroelectro.com/tutorials/servo_motor/servomotor.html>.

Rocha, T. L. (2011). percepção do professor acerca do uso das mídias e da tecnologia na prática pedagógica. *Cadernos da FUCAMP*, 10(13):1–10. http://www.fucamp.edu.br/editora/index.php/cadernos/article/view/159/168.

Romano, V. F. (2002). *ROBÓTICA INDUSTRIAL - Aplicação na Indústria de Manufatura e de Processos*. Edgard Blucher Ltda, São Paulo, SP, 1ª Edição.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., and Fujimura, K. (2002). The intelligent ASIMO: system overview and integration. *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, 3:2478–2483. DOI: 10.1109/IRDS.2002.1041641, ISBN:0-7803-7398-7.

ScienceDaily (2015). Completely paralyzed man voluntarily moves his legs, scientists report. Disponível On-line em 30 Setembro de 2015 na URL http://www.sciencedaily.com/releases/2015/09/150901204831.htm.

Silva, A. L. V. d. (2008). Arquitetura compacta para projeto de robôs móveis visando aplicações multipropósitos. Dissertação (mestrado), USP-Escola de Engenharia de São Carlos, São Carlos, SP. Disponível on-line em 10 Outubro de 2015 em <`http://www.teses.usp.br/teses/disponiveis/18/18153/tde-17032009-115712/publico/Andre.pdf`>.

Silva, F. M. T. P. D. (2001). *Análise Dinâmica e Controlo de Sistemas Robóticos de Locomoção Bípede*. PhD thesis, Faculdade de Engenharia da Universidade do Porto.

Souza, J. S. (2011). Movimentar um braço robótico, utilizando câmeras para a captura de movimentos. Trabalho de Conclusão de Curso.

TheOldRobotsWebSite (2009). Bioloid humanoid small robots. Disponível On-line em 04 Outubro de 2015 na URL <`http://www.theoldrobots.com/SmallRobot10.html`>.

Toscano, G. S. (2011). Análise cinemática e geração de trajetórias estáveis para um robô bípede antropomórfico. Dissertação (mestrado), Universidade Federal de Santa Catarina, Florianópolis, SC. Disponível on-line em 10 Outubro de 2015 em <`http://repositorio.ufsc.br/xmlui/handle/123456789/95693`>.

Villate, J. E. (2012). *Física 2. Eletricidade e Magnetismo*. Faculdade de Engenharia Universidade do Porto, 3 edition. 221 págs. Disponível On-line em Outubro de 2015 no site: <`http://repositorio-aberto.up.pt/handle/10216/19525`>.

Westervelt, E., Grizzle, J., Chevallereau, C., Choi, J., and Morris, B. (2007). *Feedback Control of Dynamic Bipedal Robot Locomotion (Automation and Control Engineering)*, volume 28 of *Automation and Control Engineering*. Tokyo: CRC Press. Electrical Engineering, 528 Pages - 172 B/W Illustration, Language: English.

Yamaguchi, M. (2013). Robô para uma marcha natural tal como o ser humano. A.I.2001. Dr.Guero. Disponível On-line em 20 Abril de 2015 em <`http://ai2001.ifdef.jp/`>, Linguagem: Japonês.

Zanco, W. D. S. (2005). *Microcontroladores PIC16F628A/648A*. Érica Ltda, 1 edition. [Uma Abordagem Prática e Objetiva].