INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Telecomando de um robô Hexápode

ANDRÉ MIGUEL LOPES TRIGÓ
outubro de 2017

POLITÉCNICO
DO PORTO

TELECOMANDO DE UM ROBÔ HEXÁPODE

André Miguel Lopes Trigó

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2017

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: André Miguel Lopes Trigó, Nº 1100394, 1100394@isep.ipp.pt
Orientação científica: Ramiro Barbosa, rsb@isep.ipp.pt
Co-Orientação científica: Manuel Silva, mss@isep.ipp.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
16 de outubro de 2017

À minha família e à Sandra

Agradecimentos

Ao longo deste percurso enriquecedor e extremamente exigente que foi o Mestrado de Engenharia Eletrotécnica e Computadores existiram fatores determinantes para o sucesso e por consequência conclusão do curso que culmina com este trabalho.

Da parte do Instituto de Engenharia e do Porto houve os meus orientadores Ramiro Barbosa e Manuel Santos Silva que ajudaram em tudo que esteve ao seu alcance assim como foram os principais motores para o desenrolar deste trabalho. Os meus colegas de turma Pedro Almeida e Nuno Guerra que partilharam seu conhecimento assim como ajuda e companheirismo.

Da parte da WEG aos meus colegas de trabalho da manutenção e engenharia industrial que ajudaram a obter mais conhecimento e autonomia.

Por fim à minha família e à minha namorada Sandra Martins pelo apoio incansável e motivação para conseguir seguir em frente e ter força para terminar este curso.

Resumo

Nos dias de hoje, como é conhecido muitas partes do nosso planeta são inacessíveis para qualquer mecanismo de locomoção que utilize rodas. Os obstáculos naturais como rochas, solo irregular, encostas ou até mesmo montanhas tornam o seu acesso difícil. No entanto para este tipo de terreno as pernas são perfeitamente adequadas para as superar com mais ou menos dificuldade e dessa forma é seguro dizer que são mais eficazes que rodas. Os mecanismos dotados de pernas permitem trepar e até evitar desníveis permanecendo equilibrados.

Os sistemas artificiais de locomoção são estruturas mecânicas com várias pernas que por sua vez são constituídas por elos ligados por eixos lineares ou rotacionais. Estes sistemas imitam sistemas biológicos e tal como eles apresentam vantagens face a veículos convencionais. Todavia estes tipos de sistemas apresentam fenómenos cinemáticos e dinâmicos muito complexos o que torna difícil sua análise e seu controlo.

O trabalho proposto "Telecomando de um Robô Hexápode" vai permitir numa primeira fase explorar e desenvolver os modelos cinemáticos e dinâmicos inerentes aos hexápodes. Numa segunda fase é feita a elaboração desse modelo cinemático num hexápode constituído por controlo de um microcontrolador, servos para a locomoção de cada perna, e um telecomando para enviar os comandos da locomoção.

Grande parte deste trabalho vai ser o dimensionamento dessa locomoção assim como o seu controlo e a programação das unidades de microprocessadores e a comunicação do hexápode com o telecomando.

Palavras-Chave

Róbotica, Locomoção, Cinemática, Hexápode, Telecomando, Microprocessador.

Abstract

Today it is known that many parts of our planet is inaccessible to any locomotion mechanism using wheels. The natural obstacles such as rocks, uneven ground, slopes or even mountains makes its difficult access. However for this type of ground legs are perfectly suitable for overcoming with more or less difficulty and they are more effective than wheels.

The mechanisms with legs allow climbing and to avoid gaps while remaining balanced. Artificial locomotion systems are mechanical structures with several legs which in turn are made up of links connected by linear or rotational axes. These systems mimic biological systems and they have advantages over conventional vehicles. However these types of systems have very complex kinematic and dynamic phenomena which makes it difficult to analyze and control.

The proposed work "Remote Control of a Hexapod Robot" will allow in a first phase of work to explore and develop the kinematic and dynamic models inherent to hexapods. In a second phase is made the elaboration of this kinematic model in a hexapod consisting of a microcontroller, servos for the locomotion of each leg, and a remote control to send the locomotion commands.

Much of this work will be the design of such locomotion as well as the control and programming of microprocessor units and communication of hexapod with the remote control.

Keywords

Robotics, Locomotion, Kinematics, Hexapod, Remote Control, Microprocessor.

Índice

AGRADECIMENTOS ..... **I**

RESUMO ..... **III**

ABSTRACT ..... **V**

ÍNDICE ..... **VII**

ÍNDICE DE FIGURAS ..... **IX**

ÍNDICE DE TABELAS ..... **XI**

ACRÓNIMOS ..... **XIII**

1. INTRODUÇÃO **1**

1.1. CONTEXTUALIZAÇÃO 2
1.2. OBJECTIVOS 3
1.3. CALENDARIZAÇÃO 4
1.4. ORGANIZAÇÃO DO RELATÓRIO 4

2. ESTADO DE ARTE **5**

2.1. ROBÔS BÍPEDES 5
2.2. ROBÔS QUADRÚPEDES 7
2.3. ROBÔS HEXÁPODES 8
2.4. ROBÔS OCTÓPODES 9

3. ANÁLISE DE LOCOMOÇÃO ROBÓTICA HEXÁPODE **11**

3.1. INTRODUÇÃO 12
3.2. MECANISMOS DE HEXÁPODES 15
3.3. CINEMÁTICA 16
3.4. PLANEAMENTO DE TRAJETÓRIAS 22
3.5. TIPOS DE LOCOMOÇÃO 25
3.6. DINÂMICA 28

4. ESCOLHA DE TECNOLOGIA **31**

4.1. DETALHES PARA REQUISITOS 31
4.2. ESCOLHA DE COMPONENTES 32

5. ARQUITETURA GERAL **47**

5.1. LIGAÇÕES DO ROBÔ 48
5.2. LIGAÇÕES DO COMANDO 50
5.3. INICIALIZAÇÕES DO ROBÔ 51
5.4. INICIALIZAÇÕES DO COMANDO 55

6. TESTES REALIZADOS **59**

6.1. ROBÔ 59

6.2. COMANDO 64
7. IMPLEMENTAÇÃO DO ROBÔ 71
7.1. DESENVOLVIMENTO DO MENU DO COMANDO 71
7.2. PARAMETRIZAÇÃO E SIMULAÇÃO DO ROBÔ 74
7.3. DESENVOLVIMENTO DOS PADRÕES DE LOCOMOÇÃO 86
7.4. TESTES E RESULTADOS 97
8. CONCLUSÕES 105
REFERÊNCIAS DOCUMENTAIS 107
ANEXO A. CÁLCULO DE CINEMÁTICA EM MATLAB 111
ANEXO B. CÓDIGO ARDUINO DO COMANDO 113
ANEXO C. CÓDIGO PYTHON DO RASPBERRY PI 125

Índice de Figuras

Figura 1 Robô Mantis [1] 2
Figura 2 Robô TOPIO 3.0 [2] 6
Figura 3 Robô AIBO ERS-7M2 [3] 7
Figura 4 Robô LAURON V [4] 8
Figura 5 Robô Mondo Spider [5] 10
Figura 6 Dois tipos de robôs Hexápodes [18] 15
Figura 7 Tipos de graus de liberdade num Hexápode [18] 16
Figura 8 Sistema de coordenadas de um braço [19] 17
Figura 9 Configuração para α 20
Figura 10 Movimentação ciclóide [20] 23
Figura 11 Numeração e sentido das pernas do hexápode 26
Figura 12 Diagrama da marcha trípode 27
Figura 13 Diagrama da marcha metacronal 27
Figura 14 Diagrama da marcha *ripple* 28
Figura 15 Estrutura Phoenix [24] 33
Figura 16 Estrutura PhantomX MK-III [25] 33
Figura 17 Estrutura AH3-R [26] 34
Figura 18 Raspberry Pi [27] 36
Figura 19 Arduino Mega2560 [28] 36
Figura 20 ARM STM32F103 [29] 37
Figura 21 Servo MG966R [30] 39
Figura 22 Servo HS-645MG [31] 39
Figura 23 Servo Ax-12 [32] 40
Figura 24 Controlador PCA9685 [33] 41
Figura 25 Controlador 32-Channel Servo Controller Board V2 [34] 42
Figura 26 Controlador Mini Maestro 24-Channel USB Servo [35] 42
Figura 27 Módulo sem fios nRF24L01 [36] 44
Figura 28 Módulo sem fios CC1101 [37] 44
Figura 29 Módulo sem fios XBee Pro [38] 45
Figura 30 Diagrama geral do projeto 48
Figura 31 Ligações Rapsberry Pi 49
Figura 32 Aspeto do Robô após montagem 50
Figura 33 Ligações do Arduino Mega 51
Figura 34 Raspbian 52

Figura 35 Impulsos PWM para servo [32] 54
Figura 36 IDE Arduino 55
Figura 37 LCD Inhaos 56
Figura 38 *Shield* da Funduino 57
Figura 39 Código de Python para XBee 60
Figura 40 Teste entre Rpi e XCTU 61
Figura 41 Posição e numeração de cada pata do robô 62
Figura 42 ServoMed em todos os servos visto de frente 63
Figura 43 ServoMed em todos os servos visto de cima 63
Figura 44 Código de Arduino para XBee 65
Figura 45 Teste entre Arduino e XCTU 65
Figura 46 Código de Arduino do LCD 67
Figura 47 Teste de Arduino no LCD 67
Figura 48 Código Arduino dos botões e analógico 68
Figura 49 *Shield* Arduino 69
Figura 50 Comando do Hexápode 72
Figura 51 Fluxograma do menu do comando 73
Figura 52 Pata do Hexápode e seus pontos zeros das juntas α e β 76
Figura 53 Disposição das patas e posição zero para a junta γ 77
Figura 54 Área de trabalho de uma pata 78
Figura 55 Ciclóide na fase de transferência em locomoção tripé 81
Figura 56 Ângulos γ, α e β ao longo de um período em locomoção tripé 82
Figura 57 Ciclóide na fase de transferência em locomoção ondulatória 83
Figura 58 Ângulos γ, α e β ao longo de um período em locomoção ondulatória 84
Figura 59 Ciclóide na fase de transferência em locomoção *ripple* 84
Figura 60 Ângulos γ, α e β ao longo de um período em locomoção *ripple* 85
Figura 61 Fluxograma do robô 87
Figura 62 Posição *Home* vista de cima 88
Figura 63 Posição *Home* vista de frente 89
Figura 64 Ângulos γ, α e β ao longo de cinco períodos em locomoção tripé 91
Figura 65 Diagrama da locomoção tripé 92
Figura 66 Ângulos γ, α e β ao longo de cinco períodos em locomoção ondulatória 93
Figura 67 Diagrama da locomoção ondulatória 94
Figura 68 Ângulos γ, α e β ao longo de cinco períodos em locomoção *ripple* 95
Figura 69 Diagrama da locomoção *ripple* 96
Figura 70 Locomoção tripé 97
Figura 71 Locomoção ondulatório 98
Figura 72 Locomoção *ripple* 99

Índice de Tabelas

Tabela 1 Calendarização do trabalho 4
Tabela 2 Comparativos entre estruturas 34
Tabela 3 Comparativos entre microcontroladores 38
Tabela 4 Comparativos entre servomotores 40
Tabela 5 Comparativos entre controladores de servos 43
Tabela 6 Comparativos entre módulos sem fios 45
Tabela 7 Parâmetros das juntas e elos 75
Tabela 8 Limites mecânicos das juntas 77
Tabela 9 Resultados da primeira configuração 100
Tabela 10 Resultados da segunda configuração 101
Tabela 11 Resultados da terceira configuração 102

Acrónimos

ARM	-	*Advanced RISC Machine*
BLDC	-	*Brushless Direct Current*
CAN	-	*Controller Area Network*
CNC	-	*Computer Numerical Control*
DOF	-	*Degree of Freedom*
FSK	-	*Frequency-shift keying*
FIFO	-	*First-in First-out*
GFSK	-	*Gaussian frequency-shift keying*
GPIO	-	*General-purpose input/output*
GPU	-	*Graphics processing unit*
HDMI	-	*High-Definition Multimedia Interface*
I2C	-	*Inter-Integrated Circuit*
I/O	-	*Input Output*
IDE	-	*Integrated Development Environment*
IEEE	-	*Institute of Eletrical Eletronics Engineers*
LCD	-	*Liquid-Crystal Display*
MSK	-	*Minimum-Shift Keying*
PCB	-	*Printed Circuit Board*

PID - *Proportional Integral Derivative*

PWM - *Pulse Width Modulation*

RAM - *Random Access Memory*

RISC - *Reduced Instruction Set Computing*

ROM - *Read Only Memory*

RTC - *Real Time Clock*

RTOS - *Real-time operating system*

SPI - *Serial Peripheral Interface*

SWD - *Serial Wire Debug*

USART - *Universal Synchronous Asynchronous Receiver Transmitter*

USB - *Universal Serial Bus*

WAN - *Wireless Area Network*

1. INTRODUÇÃO

No âmbito da unidade curricular TEDI (Tese | Dissertação) do Mestrado em Engenharia Eletrotécnica e Computadores (MEEC) foi proposto e consequentemente validado um projeto, que consiste num Hexápode com controlo comandado via sem fios por unidades XBee, sendo ambas as unidades controladas por microcontroladores.

O principal objetivo deste trabalho é mostrar as vantagens do uso de um controlo numa unidade com um tipo de locomoção complexa, o qual é um problema matemático bastante elaborado que necessita de conhecimento específico.

Outro ponto importante é realçar as qualidades de uma locomoção multiperna em relação ao convencional de rodas em comparação a solos irregulares. Atualmente existem alguns projetos de Hexápodes, mas normalmente os algoritmos de movimentação são um pouco simplificados para evitar a complexidade do envolvimento matemático, tendo este trabalho por objetivo colmatar essas falhas observadas.

1.1. CONTEXTUALIZAÇÃO

Este projeto surgiu da vontade de realizar um trabalho no âmbito da robótica envolvendo microcontroladores, controlo avançado e sistema de comunicação sem fios. O principal motivo deve-se à elevada exigência que requer uma modelação matemática da locomoção de um hexápode e todos os benefícios que esta tem em termos de graus liberdade.

Um hexápode no contexto da robótica é um veículo mecânico que anda sobre seis patas. Um robô pode ficar facilmente estável sobre três ou mais patas, e dessa forma um hexápode tem boa liberdade de movimentação porque é biologicamente inspirado na locomoção de um animal *hexapoda*. Usualmente o projeto das patas variam um pouco, mas normalmente são simétricos e possuem dois ou três graus de liberdades por pata.

De uma forma generalizada a locomoção deste tipo de robô pode ser um tripé alternado, em que vai utilizar três patas para se equilibrar e para se movimentar, pode ser movimento ondulatório que envolve movimento pata a pata ou *ripple* também conhecido por tetrápode em que envolve a movimentação de duas patas em duas fases de transferência.

Existem atualmente diversos robôs inspirados em hexápode e que possuem diversas finalidades, como desviar obstáctulos, subir escadas, carregar cargas, transformarem-se em bolas e movimentarem-se, entre outras tecnologias. Existe inclusive robôs próprios para transportar pessoas, como o exemplo da Figura 1.

Figura 1 Robô Mantis [1]

Seu nome é Mantis e é o maior hexápode construído capaz de transportar uma pessoa, criado por Matt Denton e com três metros de altura e a pesar quase 2 toneladas, sendo uma evolução mecatrónica significativa [1].

1.2. OBJECTIVOS

O principal objetivo deste projeto é o estudo da mecânica de um robô hexápode e desenvolvimento de um algoritmo de locomoção eficaz e com controlo preciso de deslocação e velocidade com auxílio de microcontroladores com boa capacidade de processamento além de comunicação sem fios.

Existiu a necessidade de mostrar as diferentes etapas que permitem uma divisão de tarefas mais simples, como as seguintes:

- Estudo do estado da arte das tecnologias envolventes, controlo, microcontroladores e servos;
- Estudo da mecânica de um hexápode e suas características;
- Análise de requisitos para um hexápode e comparação da tecnologia existente;
- Arquitetura geral do sistema com seus devidos testes;
- Simulações e testes de cálculo da cinemática envolvente;
- Protótipo final do hexápode funcional;
- Conclusões e futuras melhorias.

1.3. CALENDARIZAÇÃO

A calendarização do trabalho desenvolveu-se conforme a Tabela 1 descrita.

Tabela 1 Calendarização do trabalho

Tarefa	Setembro 2016	Outubro 2016	Novembro 2016	Dezembro 2016	Janeiro 2017	Fevereiro 2017	Março 2017	Abril 2017	Maio 2017	Junho 2017	Julho 2017	Agosto 2017
Estudo do estado da arte												
Estudo dos componentes do trabalho												
Estudo da locomoção dos hexápodes												
Estudo dos requisitos												
Teste de inicializações												
Desenvolvimento do protótipo final												
Revisões												
Conclusões												
Elaboração do relatório												

1.4. ORGANIZAÇÃO DO RELATÓRIO

No Capítulo 1 é feito um pequeno enquadramento geral do trabalho assim como o seu objetivo e a respetiva calendarização. No capítulo 2, é demonstrado o Estado da Arte sobre os hexápodes e outros robôs. No Capítulo 3, é analisado em maior detalhe a cinemática e a dinâmica de um hexápode. No Capítulo 4, é reunido os requisitos para este trabalho assim como um comparativo e um estudo de cada componente. No Capítulo 5 é realizado a descrição da arquitetura geral do trabalho e as principais inicializações e noções usadas. No Capítulo 6 é elaborado uma série de testes para verificar se todos componentes estão funcionais e cumprem o requisitado. No Capítulo 7 é apresentado a implementação do robô e descrito todas as suas funcionalidades. No último capítulo, o 8, são reunidas as principais conclusões e perspetivados futuros desenvolvimentos.

2. ESTADO DE ARTE

Existem diversas formas de configurações de robôs multi-pernas, cada qual possuindo suas vantagens e desvantagens. Este capítulo pretende tanto analisar do ponto de vista da robótica de locomoção robôs bípedes, quadrúpedes, hexápodes e octópodes.

2.1. ROBÔS BÍPEDES

Robôs bípedes ou robôs humanóides são os quais o corpo aproxima-se mais à anatomia humana, normalmente tem um *design* para específicas funcionalidades e finalidades, como interagir com o meio ambiente ou utilizar ferramentas assim como realizar tarefas normalmente atribuídas a humanos, ou até para fins experimentais como o estudo da locomoção bípede e melhorar a qualidade de próteses. Em geral estes robôs possuem uma cabeça, um tronco, dois braços e duas pernas, no entanto existem alguns robôs humanóides que só têm partes do corpo da cintura para cima ou até apenas a cara para imitar expressões humanas. Como exemplo existe o robô TOPIO [2] desenvolvido

desde 2005 pela TOSY, existem três versões deste robô e é utilizado para jogar ping-pong, a versão 3 é demonstrado na Figura 2.

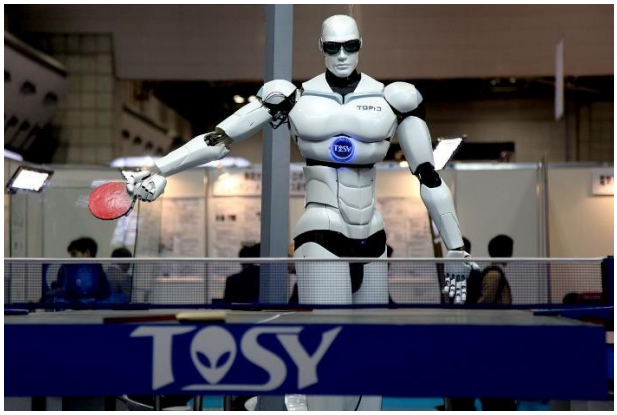


Figura 2 Robô TOPIO 3.0 [2]

Os atuadores que são responsáveis pela locomoção são usados para replicar a função de músculos e juntas. Os mais utilizados são os atuadores pneumáticos, hidráulicos e elétricos. Os sensores mais utilizados são os ultrassónicos, indutivos e até infra-vermelhos.

Os Robôs com duas pernas, ou bípedes, utilizam apenas o seu par de pernas para moverem-se, realizando sua locomoção em relação ao chão e mantendo seu equilíbrio constantemente de modo a evitar sua queda. Segundo Phuong et al sua modelação é baseada em um pêndulo invertido em três dimensões, onde seu sistema de controlo é realizado baseado na solução da cinemática inversa, determinada pelo método da geometria dos sólidos, de modo que seu padrão de locomoção é gerado por um sistema de controlo de trajetória referido ao *Zero Moment Point* (ZMP), permitindo assegurar a locomoção dinâmica do robô bípede.

A dificuldade encontrada por Phuong et al nesses robôs está na complexidade do controlo de estabilidade perante sua locomoção. A modelação foi feita considerando um robô bípede com 10 graus de liberdade (DOF) e foi implementada no robô CIMEC-1. Como possui um menor número de pernas este apresenta o menor consumo energético.

2.2. ROBÔS QUADRÚPEDES

Robôs quadrúpedes são os que se assemelham em termos de locomoção aos animais terrestres de quatro patas, muitos dos nossos mamíferos quadrúpedes são vertebrados. Em geral estes possuem quatro patas com três DOF cada. Existem diversos robôs conhecidos deste género sendo um deles o AIBO demonstrado na Figura 3 que foi criado em 1998 e se assemelha a um cão e é usado para brinquedo e estudos.

Figura 3 Robô AIBO ERS-7M2 [4]

Quando comparado aos robôs com seis e oito pernas os quadrúpedes desfrutam de uma estrutura mais simples. Porém o desempenho dos quadrúpedes é afetado diretamente quando há uma falha elétrica ou mecânica de uma de suas pernas durante o período de locomoção.

Os motores usados usualmente são os servomotores, também existe atuadores por bomba hidráulica que são cilindros. O controlo usado é muito dependente da dinâmica pois requer grande estabilidade no corpo para manter este tipo de robô em movimento.

Em relação às locomoções existem principalmente duas, que são a marcha rastejante e a marcha de trote ou galopante, ambas se baseiam na presença da estabilidade do corpo

no momento do levantar as patas. No caso da marcha rastejante existe estabilidade proporcionada por três patas formando um triângulo, na marcha de trote são duas patas que equilibram o corpo enquanto outras duas impulsionam para a frente o corpo.

2.3. ROBÔS HEXÁPODES

Os robôs hexápodes são um veículo mecânico que se movimenta em seis pernas, visto que um robô pode ser estável em três ou mais patas um robô hexápode vai ter mais flexibilidade e maior liberdade de movimento, outra vantagem é no facto de se uma pata se tornar inoperável este pode na mesma se mover. Além disso nem todas as pernas do robô são necessárias para estabilidade e podem ser usadas as patas livres para chegar a algum outro sítio ou para suportar carga. Um conhecido robô deste género, que nasceu no início da década de 90, é LAURON e foi desenvolvido na Alemanha na FZI e demonstrado na Figura 4, sendo criadas até aos dias de hoje 5 gerações deste robô. A finalidade deste modelo é para estudar e realizar movimentação em terreno desnível, isto devido à sua grande flexibilidade e ao seu sistema de controlo. Este também tem instalado uma câmara panorâmica que permite ao robô ter a capacidade de reconhecer o seu ambiente em redor.

Figura 4 Robô LAURON V [5]

Os atuadores mais encontrados nestes robôs são os servomotores porque permitem um controlo mais preciso, esse controlo pode ser variado desde PID até outros tipos, os sensores variam desde óticos, ultrassónicos e até mesmo para reconhecimento do espaço, como câmaras.

Sorin e Nitulescu estudaram justamente a análise da estabilidade estática de robôs hexápodes quer em queda livre, onde concluíram que esta análise é útil pois permite investigar o que acontece com o robô em terreno irregular ou queda por acidentes que podem acontecer devido à perda de contacto em superfícies escorregadias, inoperância do sistema atuador ou por falha na alimentação, quer nos instantes de transição, onde descreve a variação do polígono de apoio durante a sequência de marcha, mostrando a importância do correto sincronismo das pernas afim de satisfazer a condição de estabilidade.

As locomoções dos hexápodes são principalmente a tripé que envolve deslocação com três patas que formam um triângulo e as restantes três apoiam o peso, a metacronal ou ondulatória que é um movimento pata a pata em que descreve uma onda, e por fim a *ripple* ou tetrápode. A vantagem dos hexápodes é que permitem também as locomoções quadrúpedes.

2.4. ROBÔS OCTÓPODES

Estes robôs são os que se assemelham aos seres vivos de oito patas e têm uma locomoção muito parecida aos hexápodes. Um robô similar a um octópode é o Mondo Spider, observável na Figura 5. Este foi desenhado e desenvolvido durante 2005 e 2006 por uma equipa de Vancouver, sendo a fonte principal de energia um motor de 18 kW que alimenta os *drives* para as oito patas [6].

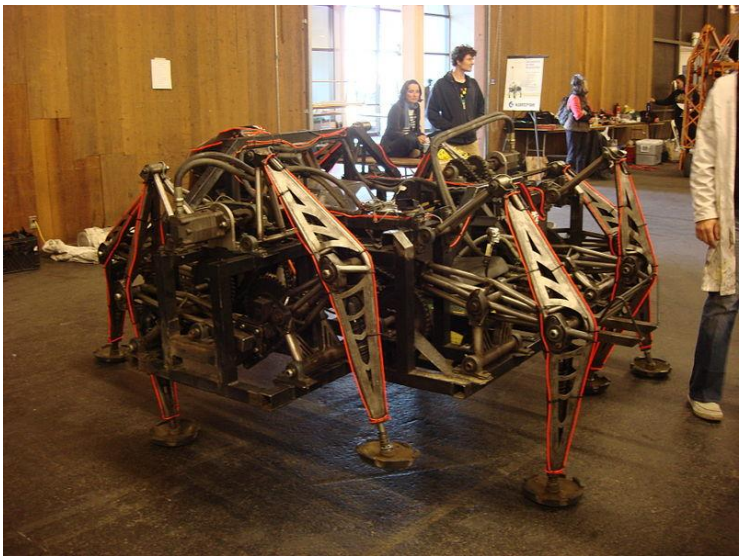


Figura 5 Robô Mondo Spider [8]

Os atuadores para este género de robô variam desde motores hidráulicos até motores elétricos e até mesmo motores por explosão, neste caso especifico são usados todos estes mencionados.

No entanto os octópodes careciam de desenvolvimento porque apresentam as seguintes lacunas:

- Elevado custo para atuadores e alimentação dos mesmos por possuir mais membros e mais DOF;
- Maior complexidade de controlo e planeamento de trajetórias;
- Mais peso e consequentemente maiores problemas de dinâmica.

Analisando todas as arquiteturas, e variando a sua velocidade frontal, observa-se indiscutivelmente que quanto menor for o número de pernas do robô maior é a complexidade em realizar a locomoção. Conclui-se também que há certos valores de velocidade frontal onde cada estrutura se destaca. Portanto, os octópodes ainda necessitam de maiores avanços tecnológicos de forma a possuírem um maior reconhecimento e utilização.

3. ANÁLISE DE LOCOMOÇÃO ROBÓTICA HEXÁPODE

O mecanismo primordial em que a robótica se sustenta consiste na modelação do seu sistema. É fundamental ter o conhecimento matemático do sistema, pois é através dele que os fenómenos físicos reais são reconstruídos e considerados no sistema. Desta forma, um robô, quer seja fixo ou móvel, possui um conjunto de equações que representam as suas ações temporais. Restritamente aos robôs móveis, as equações relacionadas com a questão da mobilidade do robô são afetadas diretamente, variando-se consoante ao tipo de locomoção (rodas, lagartas, pernas).

O processo de modelação de um sistema robótico segue uma sequência ordenada de tarefas, são elas:

- definir um nível de abstração do modelo robótico real para um modelo básico;
- desenvolver o estudo cinemático;

- desenvolver o estudo das locomoções;
- desenvolver o estudo dinâmico.

Após todos os critérios acima serem realizados, o robô está apto a ser implementado num ambiente real. A seguir serão abordados cada um dos itens definidos acima.

3.1. INTRODUÇÃO

Os robôs multi-pernas demonstram uma vantagem clara sobre os veículos a rodas quando o terreno é irregular porque não precisam de constante suporte sobre o terreno. Nestes mesmos robôs, especialmente Hexápodes que é o caso deste trabalho, os veículos mecânicos que se movimentam sobre seis pernas, têm atraído uma atenção considerável nas décadas recentes. Suas vantagens são:

- São fáceis de manter estáticos e estáveis sobre três ou mais pernas;
- Têm grande flexibilidade de movimentação;
- Têm a forma mais eficiente para movimentação estável, no entanto mais que seis não aumenta a velocidade;
- Mostra robustez no caso de perda de pernas;
- Estes permitem usar uma, duas ou três pernas para realizar trabalho e operações complexas.

O problema mais estudado para robôs multi-pernas é determinar a sequência de levantar e pousar uma pata. Do ponto de vista de estabilidade, a locomoção do robô pode ser classificada em locomoção dinâmica, como correr e saltar, e locomoção estável como andar [10].

A locomoção estável estática tem a restrição que o corpo em movimento tem que se encontrar sempre estável a todo momento. A projeção vertical da locomoção do centro

da gravidade do robô tem que estar dentro das convexões dos polígonos de suporte de cada pata.

Um movimento estável é dependente da forma do corpo e das pernas. A movimentação dos hexápodes tem sido investigada como função da forma e características da estrutura do robô. Em 1985, Kaneko endereçou a movimentação retangular de um hexápode com liberdades desconetadas onde a movimentação foi gerada por um grau de liberdade (DOF). Em 1988, Lee realizou um sistema de controlo para andar omnidirecional para um robô hexápode retangular com suspensão adaptativa. Uma movimentação circular foi estudada para um hexápode de camadas, seu nome Ambler, na Universidade de Carnegie Mellon com pernas rotativas ligadas ao mesmo eixo vertical em seis diferentes alturas. Hirose em 1992 e 1998 e Gurocak em 1998 desenvolveram outros dois hexápodes com o corpo a consistir em duas camadas, cada uma ligada a três pernas. A movimentação relativa da camada realizou a deslocação omnidirecional numa forma simples, mas limitando a capacidade de andar por causa de falhas nas pernas. Two Lees em 2001 estudou a movimentação de um robô em especial, o qual a composição do corpo era de três partes conectadas por elos de revolução. A sua locomoção flexível permitiu superar terrenos complexos, a sua configuração, no entanto foi demasiado complexa para um sistema de controlo. RHex, introduzido por Uluc em 2001, é outro hexápode com pernas de semicírculos com uma locomoção de alternada de tripé. Os hexápodes mais populares podem ser divididos em duas categorias, retangular e hexagonal. Hexápodes retangulares têm um corpo retangular com dois grupos de três pernas distribuídas simetricamente nos dois lados. Hexápodes hexagonais têm um corpo circular ou hexagonal com as pernas distribuídas uniformemente. As deslocações de robôs retangulares com seis pernas motivaram um número de pesquisas teóricas e experimentações que hoje em dia chegou a uma extensão de maturidade. Em 1998 Lee mostrou para hexápodes retangulares a margem de estabilidade longitudinal, que é definido como a distância mais curta da projeção vertical do centro de gravidade para os limites do padrão de suporte do plano horizontal, isto para movimentação em linha reta e andar de caranguejo [11].

Song & Choi em 1990 definiram o fator β como uma fração de um ciclo em que a perna está em fase de suporte e provaram que a locomoção da onda é estável entre as deslocações periódicas e regulares para hexápodes retangulares quando $¾ < \beta < 1$. Ambas as deslocações tripé e o problema de rodar sobre um ponto num terreno plano

foi investigado e testado para um hexápode retangular com três DOF nas pernas. O chamado de locomoção quadrúpede foi demonstrado como tolerante a falhas. Umas séries de locomoção tolerantes a falhas foram analisados por Yang. O seu foco era manter a estabilidade em caso de falhas que provocassem uma perna de suportar o robô. Em 1975, Kugushev e Jaroshevsky propuseram uma locomoção livre que se adaptasse ao terreno e que não fosse periódica.

McGhee e outros pesquisadores continuaram a estudar locomoções livres de robôs hexápodes retangulares. Ao mesmo tempo, os robôs hexápodes hexagonais foram estudados com a inspiração da família dos insetos, e demonstram melhor desempenho para alguns aspetos do que os robôs retangulares. Kamikawa em 2004, confirmou a capacidade de subir e descer uma ladeira com a locomoção tripé através da construção de uma superfície lisa virtual que se aproxima ao solo exato. Yoneda em 1997 aumentou os resultados de Song & Choi em 1990, desenvolvendo uma marcha de onda variável no tempo para robôs hexagonais, na qual a velocidade, fator de utilização e ângulo de caranguejo são alteradas de acordo com as condições do terreno. A. Preumon em 1991 demonstrou que os hexápodes hexagonais podem facilmente orientar em todas as direções e que têm margem de estabilidade mais longo, mas não deu uma análise teórica detalhada. Takahashi em 2000 descobriu que os robôs hexagonais podem girar e mover em todas as direções ao mesmo tempo melhor do que os retangulares, comparando a margem de estabilidade e deslocação em marcha onda, mas sem resultados experimentais. Chu e Pang em 2002 compararam a marcha de tolerância a falhas e a marcha 4 + 2 para dois tipos de hexápodes do mesmo tamanho. Eles provaram que, teoricamente, os robôs hexápodes hexagonais têm uma estabilidade superior, passo e capacidade em comparação com robôs retangulares. Também vale a pena mencionar aqui um trabalho realizado por Gonzale de Santos, o qual otimizou a estrutura de hexápodes retangulares e descobriu que o alargamento do comprimento médio das pernas de robôs retangulares ajuda na economia de energia. Este resultado pode ser visto como uma transição dos robôs retangulares de seis pernas para os hexagonais.

3.2. MECANISMOS DE HEXÁPODES

Tipicamente os robôs hexápodes podem ser classificados em retangulares e hexagonais. Hexápodes retangulares são inspirados em insetos e têm seis pernas distribuídas simetricamente ao longo de dois lados, cada um possuindo três pernas. Hexápodes hexagonais tem seis pernas distribuídas assimetricamente à volta do corpo, que pode ser hexagonal ou circular, como se verifica na Figura 6 [10].

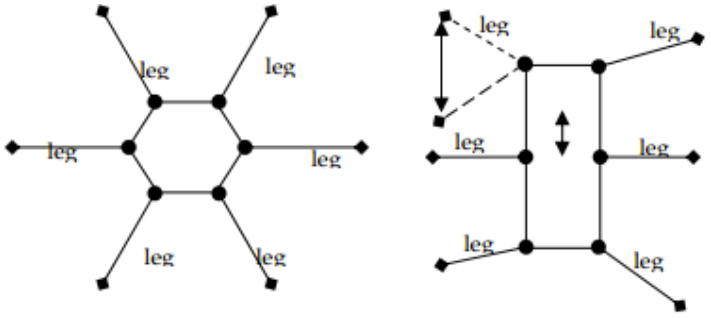


Figura 6 Dois tipos de robôs Hexápodes [10]

Tipicamente, as pernas individualmente variam de dois para seis graus de liberdade. Uma pata de um inseto tem normalmente quatro segmentos: coxa, fémur, tíbia e tarsus. Muito do comprimento de um inseto é contribuído por 2 longos e iguais segmentos. Os pontos de junção entre estes segmentos permitem dobrar ambos e ficarem exatamente um ao longo do outro. O tamanho da tíbia é correlacionado com a do fémur, essa correlação tem um fator que varia de 0,78 até 0,97. A coxa não tem correlação direta com o fémur ou tíbia. Similarmente, a coxa e a parte inferior da perna dos mamíferos e humanos são quase iguais.

A estrutura abordada será como a de 3 junções e 3 ângulos de liberdade da Figura 7 (do extremo direito), o centro demonstra uma para 4 junções e respetivos ângulos e a do extremo esquerdo uma de 2 junções e seus ângulos.

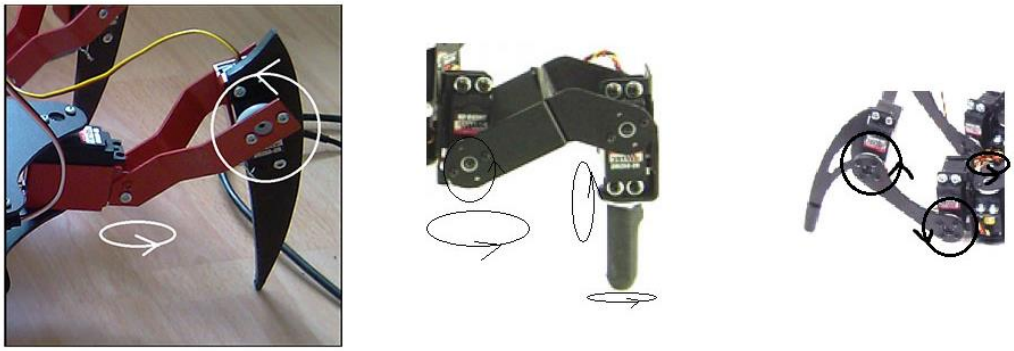


Figura 7 Tipos de graus de liberdade num Hexápode [10]

3.3. CINEMÁTICA

A mecânica é a área da física que estuda o movimento dos corpos, objetos ou partículas. Já a cinemática trata-se de um conjunto dentro da mecânica, que apenas considera o movimento do corpo, desprezando as forças que originam tal movimento. A cinemática é utilizada na robótica para a obtenção da posição e orientação desejada dos diversos componentes integrantes de um robô. Para descrever o movimento de um robô multi-pernas o estudo cinemático deve ser dividido em duas etapas, ou instantes, chamados de fase de transferência (*swing phase*) e fase de suporte (*stance phase*). A primeira etapa do estudo, *swing phase*, analisa o movimento durante a fase de transferência da perna. Nesta etapa considera-se a anca como referencial inercial fixo ao corpo do robô e os pés em movimento consoante a trajetória planeada pelo utilizador.

A Figura 8 representa os elos e as juntas da perna do robô. Embora seja possível extrair as coordenadas de posição dos pés através da convenção de Denavit-Hartenberg, achou-se viável efetuar os cálculos tendo por base as relações trigonométricas, uma vez que se trata de um sistema com 3 DOF [13].

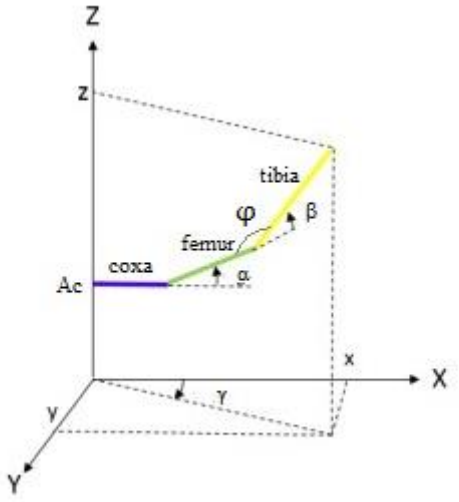


Figura 8 Sistema de coordenadas de um braço [13]

Das equações seguintes é possível concluir que a cinemática direta de uma perna do hexápode é:

$$\begin{aligned} x &= (coxa + femur.cos\,\alpha + tibia.\cos(\alpha+\beta)).\cos\gamma \\ y &= (coxa + femur.\cos\alpha + tibia.\cos(\alpha+\beta)).\sin\gamma \\ z &= Ac + femur.\sin\alpha + tibia.\sin(\alpha+\beta) \end{aligned} \quad (1)$$

- *Coxa* é o comprimento do elo que representa a coxa da perna do hexápode;
- *Fémur* é o comprimento do elo que representa o fémur da perna do hexápode;
- *Tíbia* é o comprimento do elo que representa a tíbia da perna do hexápode;
- A_C representa a altura do corpo do robô ao solo;
- γ é o valor angular da junta rotacional que representa a anca do hexápode;
- α é o valor angular da junta rotacional que representa o joelho do hexápode;
- β é o valor angular da junta rotacional que representa o tornozelo do hexápode.

Todas as pernas do robô são simétricas, logo todas possuem as mesmas expressões da cinemática direta e inversa. Ainda na fase de transferência realiza-se o estudo da cinemática inversa. Pela mesma Figura 8 extrai-se γ, pois:

$$\tan\gamma = \frac{y}{x} \tag{2}$$

Logo:

$$\gamma = \text{atan}_2{}^{-1}\frac{y}{x} \tag{3}$$

Onde $atan_2$ é a função que retorna o valor correto do par ordenado (*x*,*y*), pois caso o valor da tangente seja positivo não se pode distinguir se o ângulo é do primeiro ou terceiro quadrante e se for negativo, se é do segundo ou quarto quadrante; assim, $atan_2$ pode ser exibido por:

$$atan_2(x,y) = \begin{cases} \tan^{-1}\frac{y}{x}, & x > 0 \\ \tan^{-1}\frac{y}{x} + \pi, & y \geq 0, x < 0 \\ \tan^{-1}\frac{y}{x} - \pi, & y < 0, x < 0 \\ +\frac{\pi}{2}, & y > 0, x = 0 \\ -\frac{\pi}{2}, & y < 0, x = 0 \\ indeterminado & y = 0, x = 0 \end{cases} \tag{4}$$

Para encontrar a expressão matemática que representa β é necessário primeiramente encontrar a hipotenusa do triângulo retângulo formado conforme a Figura 9.

Identificando-se o triângulo retângulo com catetos $\sqrt{x^2 + y^2} - coxa$ e $z - Ac$ encontra-se a sua hipotenusa ρ pelo teorema de Pitágoras:

$$\rho = \sqrt{\left(\sqrt{x^2 + y^2} - coxa\right)^2 + (z - Ac)^2} \quad (5)$$

Baseando-se na lei dos cossenos, a relação entre os elos fémur, tíbia e a hipotenusa ρ com o ângulo φ é dada por:

$$\rho^2 = femur^2 + tibia^2 - 2.femur.tibia.\cos\varphi \quad (6)$$

O ângulo φ é complementar, isto é:

$$\varphi = 180° - \beta \quad (7)$$

Substituindo φ na equação (5), simplificando a equação e aplicando a identidade trigonométrica do $\cos(180° - \beta)$ tem-se:

$$\left(\sqrt{x^2 + y^2}\right)^2 + (z - Ac)^2 = femur^2 + tibia^2 - 2.femur.tibia[\cos 180°.\cos\beta + \sin 180°.\sin\beta] \quad (8)$$

$$x^2 + y^2 - 2.coxa.\sqrt{x^2 + y^2} + coxa^2 + (z - Ac)^2 = femur^2 + tibia^2 + 2.femur.tibia.\cos\beta \quad (9)$$

$$\cos\beta = \frac{x^2 + y^2 - 2.coxa.\sqrt{x^2 + y^2} + (z - Ac)^2 + coxa^2 - femur^2 - tibia^2}{2.femur.tibia} \quad (10)$$

Portanto,

$$\beta = \cos^{-1} \frac{x^2 + y^2 - 2.coxa.\sqrt{x^2 + y^2} + (z - Ac)^2 + coxa^2 - femur^2 - tibia^2}{2.femur.tibia} \quad (11)$$

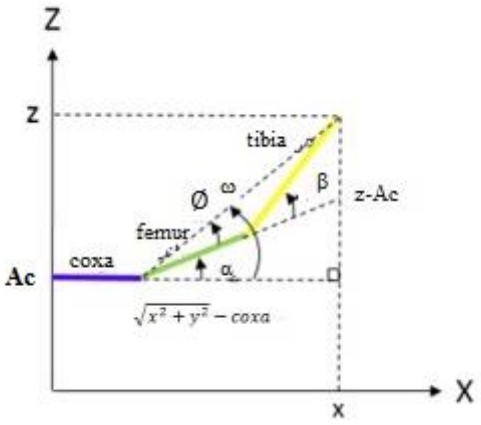


Figura 9 Configuração para α [13]

Após ser encontrada a expressão para β é possível abstrair α através da análise da Figura 9.

O ângulo α é o resultado da diferença entre:

$$\alpha = \omega - \emptyset \quad (12)$$

Sendo

$$\omega = atan_2\left(\frac{z - Ac}{\sqrt{x^2 + y^2} - coxa}\right) \quad (13)$$

$$\emptyset = atan_2\left(\frac{tibia.\sin\beta}{femur - tibia.\cos\beta}\right) \quad (14)$$

Logo,

$$\alpha = atan_2\left(\frac{z - Ac}{\sqrt{x^2 + y^2} - coxa}\right) - atan_2\left(\frac{tibia.\sin\beta}{femur + tibia.\cos\beta}\right) \qquad (15)$$

Na cinemática direta não há nenhuma restrição ou problema, originando sempre em única coordenada para o ponto escolhido. Isso já não se verifica na cinemática inversa que possui várias representações para um mesmo ponto, pois há ângulos que se complementam dando origens a redundâncias. Outro problema encontrado na cinemática inversa está no que é denominado de singularidades. Singularidades são definidas como os lugares onde o determinante da matriz jacobiana do modelo geométrico direto se anula, sendo um problema matemático e não físico da perna do robô. A singularidade ocorre quando a perna se encontra totalmente esticada, totalmente retraída ou sobre uma reta que passa verticalmente pela origem (referencial inercial) do sistema. Portanto deve-se evitar que a trajetória da perna passe por esses pontos.

Finalizando a primeira etapa, a análise da fase de transferência, é realizado o estudo cinemático para a segunda etapa, a fase de suporte. A segunda etapa é caracterizada por considerar o pé do robô fixo ao solo, oferecendo sustentação e propulsão para o robô se mover. Assim, o referencial inercial passa a ser os pés do hexápode e a anca, ou quadril, passa a possuir a trajetória planeada pelo utilizador. Nesta fase a perna é vista como um braço manipulador com a sua base fixa na coordenada do instante de colocação dos pés do robô ao solo, e assim, a coordenada *z* desse braço manipulador já expressa a altura do corpo do robô ao solo. As equações da cinemática direta permanecem inalteráveis, porém a cinemática inversa para a fase de suporte possui A_C = 0, isto é:

$$\gamma = atan_2\frac{y}{x} \qquad (16)$$

$$\alpha = atan_2\left(\frac{z - Ac}{\sqrt{x^2 + y^2} - coxa}\right) - atan_2\left(\frac{tibia.\sin\beta}{femur + tibia.\cos\beta}\right) \qquad (17)$$

$$\beta = \cos^{-1} \frac{x^2 + y^2 - 2.coxa.\sqrt{x^2 + y^2} + (z - Ac)^2 + coxa^2 - femur^2 - tibia^2}{2.femur.tibia} \qquad (18)$$

3.4. PLANEAMENTO DE TRAJETÓRIAS

O algoritmo de locomoção determina o movimento do robô, a carga do motor do robô, a dinâmica do consumo de energia, o binário dos redutores separados, etc. A maneira de mover a perna é, portanto, muito importante. Há muita literatura sobre os algoritmos de trajetória de pernas, eles variam de algoritmos genéticos cíclicos a funções de ciclóide simples relativas. A maneira mais fácil para uma geração de trajetória de perna é a partir do estudo do Rhexrobot. As pernas podem girar rapidamente em circulo na fase de transferência para retornar à fase de suporte. Para o habitual sistema de 3 DOF é preciso uma abordagem diferente. O trajeto das pernas é prescrito em ordem para um movimento. Em primeiro lugar, o modelo da cinemática inversa deve ser definido onde os ângulos das juntas são uma função da extremidade da perna. Assumindo que o robô anda na marcha do tripé ou trípode, cada perna faz exatamente o mesmo caminho, apenas um grupo de pernas é atrasado metade de um período no tempo. Um grupo está de pé no chão e empurra as pernas para trás, e os outros estão em posição levantada e avançar (supondo que o robô anda para a frente). Esta abordagem básica pode então ser mais tarde otimizada para andar mais rápido, estável e flexível, o que também influenciará a eficiência do consumo de energia [16].

Um modelo 2 DOF no modelo direto para o planeamento da trajetória cinemática, onde a velocidade do corpo do robô de velocidade constante é ajustada e as pernas seguem uma função cíclica é descrito na Figura 10.

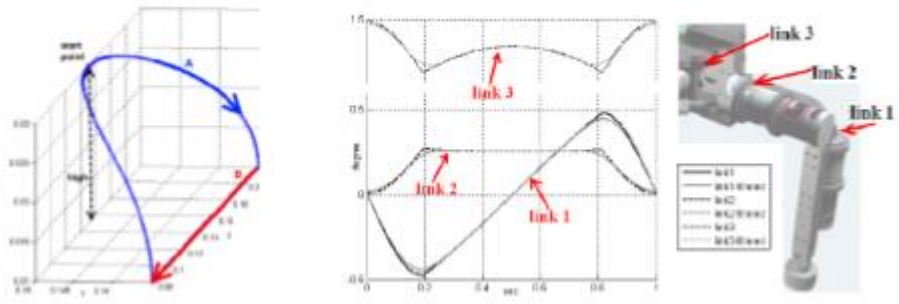


Figura 10 Movimentação ciclóide [14]

As trajetórias das pernas também são estudadas a partir de animais. Ao vincular a biologia e aplicação técnica. Insetos andantes e a maioria dos robôs de seis patas requer o controlo simultâneo de até 18 articulações. Um modelo simples de uma perna de inseto de vara consiste de quatro segmentos funcionais: a coxa, o fêmur, a tíbia e o pé. Quando a articulação do pé e do próprio pé são negligenciadas por simplicidade, uma perna de inseto de vara pode ser modelada como um manipulador com três articulações de dobradiça, resultando em 3 DOF, que normalmente é o mesmo que uma perna de robôs hexápodes. Para a otimização das trajetórias das pernas são encontradas diversas técnicas. Um filtro de resposta de impulso finito (FIR) é um tipo de filtro de tempo discreto. Com o uso deste filtro o movimento fica mais suave e, portanto, também mais eficiente como mostrado na Figura 10.

Segundo Silva, que provou que a ciclóide minimiza os correspondentes binários nas juntas melhorando o seguimento das trajetórias das ancas e dos pés [12], as coordenadas cartesianas para uma ciclóide planar são:

$$\begin{aligned} X_{tr} &= constante \\ Y_{tr} &= V_L\left(t - \frac{T \sin\frac{2\pi t}{T}}{2\pi}\right) \\ Z_{tr} &= \frac{F_c}{2}\left(1 - \cos\frac{2\pi t}{T}\right) \end{aligned} \tag{19}$$

Sendo T o período de um ciclo completo do movimento de uma pata do hexápode, V_L a velocidade de deslocamento da perna e F_c a altura máxima atingida pelas extremidades do pé.

A velocidade da perna representada na equação (19) deve ser relacionada com a velocidade frontal do corpo, de forma que o comprimento percorrido num ciclo completo, tanto por cada perna, quanto pelo corpo, seja igual a L_S, comprimento do passo. A relação entre V_L e V_F é:

$$\begin{aligned} V_F &= \frac{L_S}{T} \\ V_F &= (1-\beta)\ V_L \\ V_L &= \frac{1}{(1-\beta)} V_f \end{aligned} \tag{20}$$

Onde β é o fator de ocupação, não confundir com o ângulo β, pela simplicidade, eficiência e por permitir ajustes em V_L e F_C esta será a trajetória adotada para o planeamento da perna durante a fase de transferência. Porém na fase de suporte as pernas do hexápode devem servir de sustentação e propulsão do corpo, desse modo, considera-se o corpo em movimento retilíneo uniforme deslocando-se ao longo do eixo *y*, para representar tal movimento as coordenadas da anca devem ser:

$$\begin{aligned} X_s &= 0 \\ Y_s &= -V_F t \\ Z_s &= Ac \end{aligned} \tag{21}$$

Mais literatura para otimização da trajetória da perna é encontrada com o uso de algoritmos genéticos. Um algoritmo rápido e convergente adequado para qualquer tempo de aprendizagem. Algoritmos genéticos cíclicos são desenvolvidos com quatro variações, com uma variação que produz uma marcha ótima do tripé que é robusta o

suficiente para se adaptar a mudanças significativas nas capacidades do modelo robô. A otimização também pode ser obtida por técnicas *soft computing*. Estudos adicionais são baseados no planejamento de trajetória ponto a ponto e na forma de fazer adaptações de trajetórias para que o hexápode siga uma certa linha. Mesmo quando a falha conjunta acontece e uma articulação é bloqueada o robô ainda pode andar. Ou adaptar a posição de centro de gravidade do robô em relação à colocação dos pés, a fim de estabilizar o hexápode [15].

3.5. TIPOS DE LOCOMOÇÃO

Um padrão de locomoção é uma sequência de movimentos, tanto dos membros, quanto do corpo, de um robô ao longo de um ciclo completo de locomoção. Cada animal possui um padrão de locomoção distinto e o mesmo animal pode alterar o seu padrão de locomoção devido à necessidade de se alterar a velocidade da marcha. Existem diferentes estilos de marcha, sendo as mais comuns nos hexápodes a trípode, onda metacronal e *ripple*, variando a sua velocidade de deslocamento para lenta, média e rápida, respetivamente. A diferença básica entre essas marchas são o uso de uma, duas ou três pernas simultaneamente na fase de transição. Para implementar os padrões de locomoção num robô hexápode é necessário compreender alguns conceitos [11].

O fator de ocupação, ou *duty factor*, de um robô multi-pernas é a fração temporal em que uma perna permanece na fase de suporte relativamente ao período *T*, em outras palavras:

$$\beta_i = \frac{tempo\ de\ suporte\ da\ perna\ i}{T}, \qquad i = 1{,}2 \dots 6 \tag{22}$$

A fase $\emptyset_i$ de uma perna é a razão entre o instante em que ela entra em contacto com o solo e o período *T*, isto é:

$$\emptyset_i = \frac{instante\ em\ que\ a\ perna\ i\ toca\ no\ solo}{T}, \quad i = 1,2 \dots 6 \tag{23}$$

O tempo de aterragem da perna *i* é medido a partir do toque da perna 1, por conseguinte, tem-se $\emptyset_i$ = 0 para qualquer marcha. Essas variáveis são utilizadas para a construção de um diagrama de marchas, que é um diagrama que indica o instante em que a perna *i* inicia a fase de transferência ao longo do período *T*, com tempo normalizado.

A ordem em que as pernas se dispõem ao longo do corpo do robô possui um padrão: as pernas ímpares situam-se do lado esquerdo enquanto as pernas pares se localizam do lado direito, sendo o sentido de movimento ao longo do eixo +*y* e assinalado com a seta vermelha, como verificado na Figura 11.

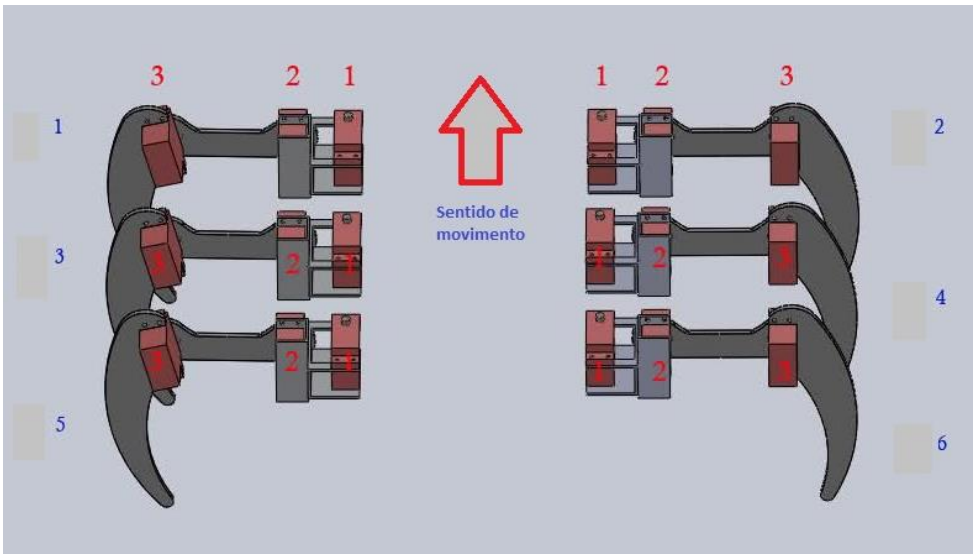


Figura 11 Numeração e sentido das pernas do hexápode

Assim, para a marcha trípode da Figura 12 as pernas 1, 4 e 5 iniciam juntas a fase de transferência, enquanto as pernas 2, 3 e 6 só irão iniciar esta fase após 50% do período, pois elas iniciam juntas a fase de suporte e possuem $\emptyset_i$ de 0,5. Esse tipo de marcha é caracterizada por atingir as maiores velocidades frontais. Seja L_S o comprimento do

passo da perna, ao fim de um ciclo cada perna do hexápode ter-se-á deslocado exatamente L_S, movimentando seu corpo L_S.

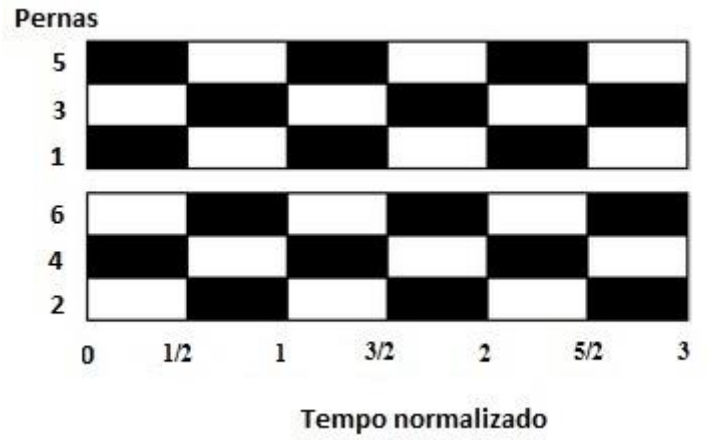


Figura 12 Diagrama da marcha trípode

Movimentando apenas uma perna de cada vez, a marcha onda metacronal alcança uma velocidade máxima inferior ao das duas marchas mencionadas, como se pode ver na Figura 13. Para desenvolver tal marcha deve-se ter $\beta = 5/6$. Portanto o fator de ocupação pode ser utilizado para distinguir se o robô está a caminhar, com $\beta > 1/2$, ou a correr, com $\beta < 1/2$.

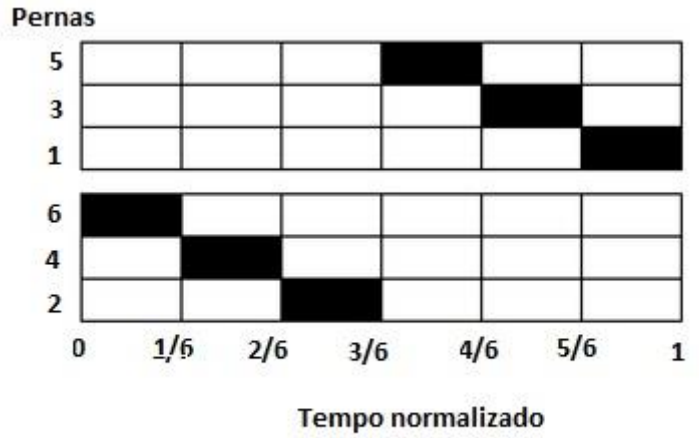


Figura 13 Diagrama da marcha metacronal

Similar à marcha da onda metacronal, a marcha *ripple* alcança uma velocidade máxima semelhante, no entanto é ligeiramente mais rápido devido à deslocação simultânea de duas patas em duas fases da locomoção, como se pode ver na Figura 14. Para desenvolver tal marcha deve-se ter $\beta = 3/4$. Portanto o fator de ocupação pode ser utilizado para distinguir se o robô está a caminhar, com $\beta > 1/3$, ou a correr, com $\beta < 1/3$.

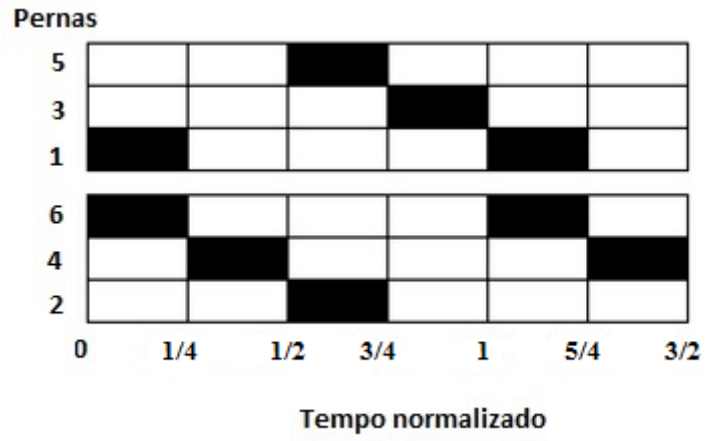


Figura 14 Diagrama da marcha *ripple*

3.6. DINÂMICA

A dinâmica é a parte da mecânica que se dedica ao estudo dos movimentos considerando as forças e/ou binários que o originam. Após realizar os cálculos da cinemática e do planeamento de trajetórias, resultando na evolução das posições, velocidades e acelerações das juntas entre os elos de cada perna do robô, considera-se à posteriori o binário necessário do atuador de junta para que o mesmo possa seguir e manter o movimento. Assim como há cinemática direta e inversa, há também dinâmica direta e inversa.

O problema da obtenção do modelo dinâmico de um robô é um dos aspetos mais complexos no domínio da robótica e é necessário para atingir os seguintes objetivos:

conceção e avaliação do controlo dinâmico do robô, dimensionamento de atuadores, avaliação da estrutura mecânica e simulação de movimento do projeto do robô.

O modelo dinâmico do robô hexápode pode ser obtido através da aplicação de algoritmos de Lagrange-Euler ou Newton-Euler. Mesmo se o método de Newton-Euler for mais eficiente do ponto de vista do processamento do computador, pode-se também utilizar o método Lagrange-Euler, porque estes robôs têm poucos DOF por perna. As equações, no entanto, não vão ser utilizadas no âmbito deste trabalho pois pressupõe-se inicialmente que a parte mecânica não precisa de cálculo prévio.

A dinâmica direta calcula as posições, velocidades e acelerações das juntas do robô a partir das forças/binários produzidas pelos atuadores das juntas. A dinâmica inversa calcula as forças/binários resultantes das posições, velocidades e acelerações das juntas do robô [15].

4. ESCOLHA DE TECNOLOGIA

Para este trabalho foi necessário destacar os principais componentes para o desenvolver. Foram consideradas algumas possibilidades pois existem diversos modos de aplicação para este projeto, no entanto tentou-se a melhor escolha possível tendo em conta as principais limitações.

4.1. DETALHES PARA REQUISITOS

Para que fosse possível começar o trabalho foi preciso definir o *hardware* e a estrutura. Para tal, as seguintes características tiveram que ser cumpridas:

- Estrutura com mobilidade para três DOF com seis pernas;

- Controlador com capacidade no mínimo de 18 servos e respetivos servos adequados para a estrutura a usar;
- Microcontrolador com capacidade considerável de processamento tanto no hexápode como na unidade de comando, assim como modos de comunicação série como SPI, I2C e USART, ADCs e I/O;
- LCD no comando para observar as várias opções;
- Comunicação rápida e fiável sem fios entre o hexápode e o comando.

4.2. ESCOLHA DE COMPONENTES

Nesta secção foi feita uma cuidada pesquisa sobre os diferentes componentes a aplicar no projeto final. Em cada um foi elaborado uma tabela final com os principais pontos fortes e dessa forma foi escolhido o melhor considerado para este caso.

4.2.1. ESTRUTURA

No caso das estruturas foram analisadas as seguintes possibilidades:

- Phoenix

  Esta estrutura comercializada pela Lynxmotion, observável na Figura 15, tem os constituintes para as 6 patas com 3 DOF cada. É feito em alumínio que permite suportar o peso da estrutura toda montada, tem o preço aproximado de 220 €, no entanto no ebay existe modelos a rondar os 70 €.

Figura 15 Estrutura Phoenix [17]

- PhantomX MK-III

Esta estrutura comercializada pela trossenrobotics, demonstrada na Figura 16, tem diversas melhorias sobre o MK-II sendo uma delas ser construído em alumínio, possui 6 patas com 3 DOF cada, existe um pacote com os servos e toda a robótica necessária para começar a funcionar a rondar os 1050 €.

Figura 16 Estrutura PhantomX MK-III [18]

- AH3-R

Esta estrutura comercializada pela Lynxmotion, demonstrada na Figura 17, tem os constituintes para as 6 patas com 3 DOF cada. É feito em alumínio que permite suportar o peso da estrutura toda montada, tem o preço de aproximadamente 820 € com o pacote todo do controlo e servos.

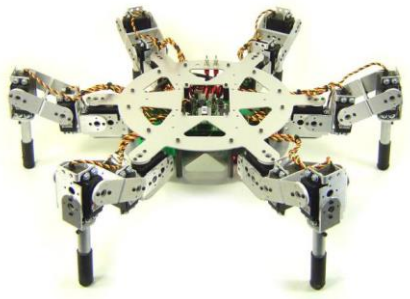


Figura 17 Estrutura AH3-R [19]

Após a análise de cada estrutura foi elaborado a Tabela 2 onde é permitido tirar as principais características e poder fazer a escolha mais equilibrada.

Tabela 2 Comparativos entre estruturas

Nome	Nº Patas	DOF por pata	Material	Extras	Preço
Phoenix	6	3	Alumínio	Nenhum	220 € / 62 €
PhantomX MKIII	6	3	Alumínio	Servos e controlador	1050 €
AH3-R	6	3	Alumínio	Servos e controlador	820 €

A escolha para a estrutura foi a Phoenix, por ter as características idênticas às restantes e possuir um preço consideravelmente inferior.

4.2.2. MICROCONTROLADOR

O microcontrolador é preciso para a manipulação do hexápode e para interligar e coordenar os servos com controlo próprio. Como é conhecido um microcontrolador é bastante utilizado para controlo de diversos dispositivos e afins, sendo composto por um processador, entradas e saídas, memórias, modos de comunicação, entre outras características.

- Raspberry Pi

O Raspberry Pi da Figura 18 é muito mais que um microcontrolador pois é um pequeno computador numa PCB do tamanho de um cartão. Já existem três gerações de Raspberry Pi no entanto foi considerada como opção a primeira geração pois é mais simples e mais acessível.

A geração em causa contém 4 modelos, no entanto foi o modelo B escolhido. Neste modelo contém um processador ARM1176JZF com 700 Mhz de frequência, uma GPU Broadcom IV BCM2837 com 400 Mhz, a RAM é de 512 MB, tem 2 portas USB, entrada Micro SD *card* para correr o sistema operativo, uma HDMI e RCA para vídeo, uma para rede RJ45 10/100 BaseT Ethernet, *jack* 3,5 mm para áudio, 27 pinos de GPIO onde possui ligações série como I2C, USART, SPI e saídas digitais assim como analógicas. É possível programar este GPIO com o uso de bibliotecas próprias desenvolvidas em Python correndo no sistema operativo Linux.

Figura 18 Raspberry Pi [20]

- Arduino Mega 2560

O microcontrolador que equipa a placa de desenvolvimento é um ATmega2560, explícito na Figura 19, a frequência de operação é de 16 Mhz. Para programar é necessário utilizar o *software* do Arduino e a placa já contém um *bootloader* que permite fazer programação sem *hardware* externo. No entanto esta placa tem um ATmega2560 com 256 Kbytes de *Flash* e 8 Kbytes de SRAM (*Static Random Access Memory*). Tem também 54 pinos de I/O dos quais 15 podem ser PWM, 16 pinos analógicos, 4 USARTS, 1 SPI e 1 I2C. Na Figura 19 está sua imagem.

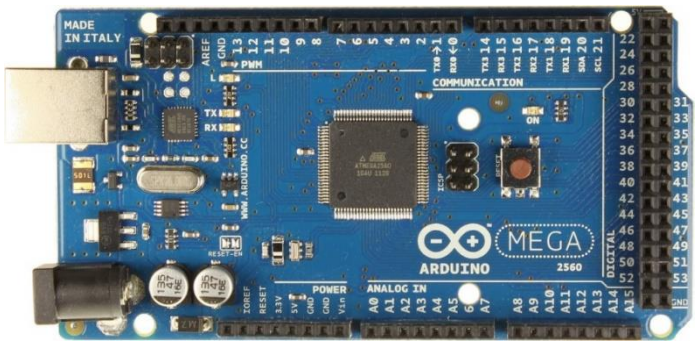


Figura 19 Arduino Mega2560 [21]

- ARM STM32F103

O microcontrolador que equipa a placa de desenvolvimento é o STM32F103 e sua imagem está na Figura 20, este pode operar até uma frequência de 72 Mhz. Para programar é necessário utilizar o conector SWD (*Serial Wire Debug*) que permite fazer a ligação entre um programador ST-LINK e o microcontrolador a programar. A placa Sistema Mínimo, não tem periférico nenhum acoplado, daí o seu nome Sistema Mínimo. Segundo o fabricante, a placa apenas dispõe de um LED no pino PC13 e um RTC (32,768 KHz), tudo o resto que a placa incorpora (resistências, condensadores, etc) é o necessário para o perfeito funcionamento do microcontrolador.

Embora esta placa seja útil para desenvolvimento, pois tem o mesmo *socket* que um microcontrolador PDIP de 40 pinos, a documentação sobre a placa em uso é escassa. No entanto sabendo que esta placa tem um STM32F103 pode-se contar com 64 Kbytes de *Flash* e 20 Kbytes de SRAM. Tem também 4 *timers*, 2 conversores A/D de 10 canais, 2 periféricos SPI e I2C, 3 USART e um periférico CAN e outro USB. Quanto aos GPIOs tem disponível 37 para uso do utilizador. A Figura 20 apresenta sua imagem.

Figura 20 ARM STM32F103 [22]

Após a análise de cada estrutura foi elaborado a Tabela 3 onde é permitido tirar as principais características e poder fazer a escolha mais equilibrada.

Tabela 3 Comparativos entre microcontroladores

Nome	Clock	RAM	ROM	Pinos	I/O	Série	Extras	Preço
Raspberry Pi	700 Mhz / 400 Mhz	512 Mb	Nenhum	27	Digital e analógico	I2C, SPI, USART, USB	HDMI, RCA e RJ45	37 €
Arduino Mega 2560	16 Mhz	8 Kbytes	256 Kbytes	100	Digital e analógico	I2C, SPI, USART	*Timers* e AD	35 €
ARM STM32F103	72 Mhz	20 Kbytes	64 Kbytes	40	Digital e analógico	I2C, SPI, USART e CAN	*Timers* e AD	8 €

Daqui a escolha para o hexápode incidiu sobre o microcontrolador Raspberry Pi, isto por possuir um processamento superior assim como a capacidade de memória. Para o telecomando a escolha foi o Arduino Mega2560 por conter boas características, programador embutido e boa capacidade de processamento assim como comunicações.

4.2.3. SERVOMOTOR

O servomotor é o que vai permitir controlar com precisão a posição das diferentes patas e seus elos. Usualmente é um motor DC, uma transmissão de rodas dentadas e a parte digital de controlo incorpora um PID que recebe impulsos PWM.

- MG996R

O servomotor MG996R, apresentado na Figura 21, tem as rodas dentadas em metal o que resulta num binário de 10 kgf.cm numa caixa pequena. Este é essencialmente um melhoramento do modelo MG995, tem a PCB à prova de choque e um controlo IC que

o faz mais preciso. Este servo pode rodar até 180º, 90º para cada lado, velocidade de 0,17 s/60º para 4,8 V e 0,14 s/60º para 6 V. Possui uma tensão de 4,8 a 7,2 V e consumo de corrente entre 500 mA a 900 mA. O seu preço ronda os 10€.

Figura 21 Servo MG966R [23]

- HS-645MG

O servomotor da Hitec HS-645MG, como demonstrado na Figura 22, tem rodas dentadas em metal, possui placa de circuito e do motor separados, tem 90º de rotação, 180º de rotação máxima, velocidade de 0,24 s/60º para 4,8 V e de 0,20 s/60º para 6 V. Necessita de uma tensão entre 4,8 a 6 V e um consumo de corrente entre 350 e 450 mA, e o seu preço ronda os 26 €.

Figura 22 Servo HS-645MG [24]

- AX-12

O servomotor da Robotis AX-12 tem rodas dentadas em metal, possui placa de circuito e do motor separados, tem 300º de rotação, ou rotação contínua. A velocidade de

0,169 s/60º para 9 V e de 0,169 s/60º para 12 V. Possui um consumo entre 9 a 12 V e de corrente entre 50 a 900 mA. Este servo tem algoritmo de controlo para a posição do veio para controlar a velocidade e o torque de resposta, o seu preço ronda os 38€. Na Figura 23 está a imagem do servomotor.

Figura 23 Servo Ax-12 [25]

Após a análise de cada servomotor foi elaborado a Tabela 4 onde é permitido tirar as principais características e poder fazer a escolha mais equilibrada.

Tabela 4 Comparativos entre servomotores

Nome	Consumo Corrente	Consumo Tensão	Torque	Alcance	Velocidade	Preço
MG966R	500-900 mA	4,8-6 V	10 kgf.cm	120º	0,17-0,12 s/60º	10 €
HS-645 MG	350-450 mA	4,8-6 V	10 kgf.cm	180º	0,24-0,20 s/60º	26 €
Ax-12	50-900 mA	9-12 V	15 kgf.cm	300º	0,169-0,169 s/60º	38 €

Reunindo toda a informação, a escolha atribuída foi o servomotor MG966R por possuir uma boa relação qualidade/preço em comparação com as restantes.

4.2.4. CONTROLADOR SERVOMOTOR

Os controladores de servomotor servem para simplificar o controlo de diversos servomotores. Quando vários servomotores são ligados diretos a um microcontrolador perdem precisão no sinal PWM. Também alguns servomotores necessitam de alta corrente, logo a parte de alimentação de potência é externa. Normalmente são comunicáveis por modos série como I2C ou USART e permitem interligar 18 ou mais servomotores com possibilidade de alimentação externa dos servos.

- PCA9685

O PCA9685 observável na Figura 24 é um *drive* controlado por I2C até 1 Mhz de velocidade com 16 canais que permite controlar a partir de PWM outros dispositivos ou cargas. Cada canal tem a resolução de 12 bits e pode operar a uma frequência de PWM programada de 24 Hz até 1526 Hz que permite um *duty cycle* de 0 % a 100 %. A tensão é de 2,4 V até 5,5 V e permite uma alimentação externa no caso de a carga consumir demasiada corrente, seu preço ronda os 12€.

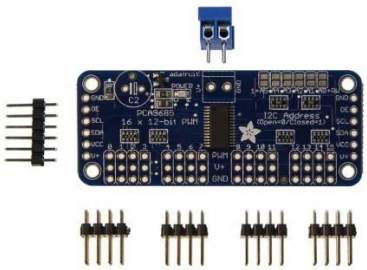


Figura 24 Controlador PCA9685 [26]

- 32-Channel Servo Controller Board V2

Este controlador não tem um circuito integrado próprio pois é uma placa com um microcontrolador programado para desempenhar o controlo de servos. Este contém 32 canais, a comunicação é por USB ou USART com o PWM ajustável e precisão de 1 µs, vindo já com o *software* onde é possível fazer as configurações iniciais do servo a usar, o seu preço ronda os 32 €. Na Figura 25 está a imagem do controlador do servomotor.

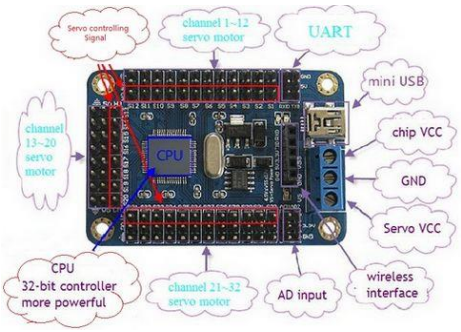


Figura 25 Controlador 32-Channel Servo Controller Board V2 [27]

- Mini Maestro 24-Channel USB Servo

Este controlador não tem um circuito integrado próprio à semelhança do anterior, este contém 24 canais, a comunicação é por USB ou USART com *baudrate* de 300 até 200000 bps com o PWM ajustável e precisão de 0,25 µs e impulsos até 333 Hz, o seu preço ronda os 52 €. Na Figura 26 está a imagem do controlador do servomotor.

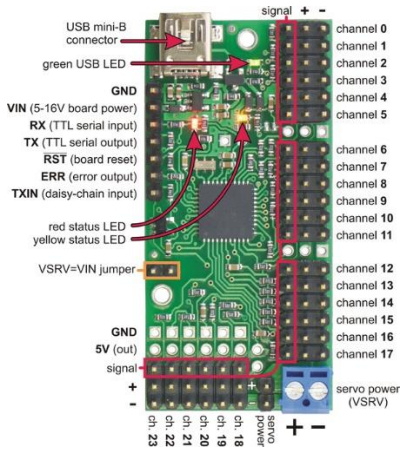


Figura 26 Controlador Mini Maestro 24-Channel USB Servo [28]

Após a análise de cada controlador de servo foi elaborado a Tabela 5 onde é permitido tirar as principais características e poder fazer a escolha mais equilibrada.

Tabela 5 Comparativos entre controladores de servos

Nome	Canais	Comunicação	Precisão	Preço
PCA9685	16	I2C	0,1 μs	12 €
32-Channel Servo Controller Board V2	32	USART e USB	1 μs	32 €
Mini Maestro 24-Channel USB Servo	24	USART e USB	0,25 μs	52 €

Daqui a escolha foi o controlador PCA9685 por possuir uma boa relação qualidade/preço em comparação com as restantes, apesar dos poucos canais, mas compensa com o emparelhar de vários módulos por I2C.

4.2.5. MÓDULO SEM-FIOS

Os módulos de comunicação sem-fios existem em diversas formas e feitios e para os mais diversos fins, neste caso foi abordado os de baixo consumo, pequena dimensão e bom alcance usando modo de comunicação série como USART ou SPI.

- nRF24L01

Este é um módulo transrecetor de baixo consumo, com tensão que oscila entre os 1,9 V e os 3,6 V com corrente máxima de 14 mA, possui 2 Mbps de velocidade, e opera na faixa de 2,4 Ghz. Este utiliza SPI para comunicar com o controlador, 3 FIFOs de 32 Bytes para RX e TX, auto-retransmissão com auto *acknowledgement* e alcance até 250 m, seu preço ronda os 6€. Na Figura 27 está a imagem do módulo.

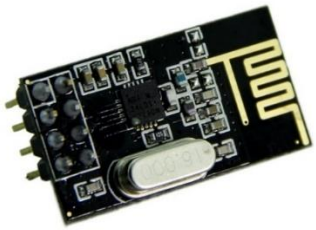


Figura 27 Módulo sem fios nRF24L01 [29]

- CC1101

Este é um módulo transrecetor de baixo consumo, na Figura 28 está o seu aspeto, tem uma tensão que oscila entre os 1,8 V e os 3,6 V com consumo de corrente máximo de 15,6 mA. Possui 500 kbps de velocidade com modulação FSK, GFSK e MSK, opera na faixa de 433 Mhz. Este utiliza SPI para comunicar com o controlador, 1 FIFOs de 64 Bytes para RX e outro para TX, auto-retransmissão com auto *acknowledgement* e cobertura de distância até 300 m, seu preço ronda os 8€.

Figura 28 Módulo sem fios CC1101 [30]

- XBee Pro

Este é um módulo muito popular da marca XBee, sua imagem está na Figura 29, operando na gama de 2,4 Ghz, possui o mesmo *pinout* e comandos dos da sua série, usa a *stack* de 802.15.4 que é a básica de Zigbee, a tensão é de 3,3 V e a corrente de 215 mA, a taxa é de 250 kbps enquanto o alcance atinge 1500 m, possui antena incorporada, 6 canais AD de 10 bits, 8 pinos I/O digitais, seu preço ronda os 32 €.

Figura 29 Módulo sem fios XBee Pro [31]

Após a análise de cada módulo sem fios foi elaborado a Tabela 6 onde é permitido tirar as principais características e poder fazer a escolha mais equilibrada.

Tabela 6 Comparativos entre módulos sem fios

Nome	Consumo Corrente	Consumo Tensão	Gama	Modulação/ Tecnologia	Alcance	Preço
nRF24L01	14 mA	1,9-3,6 V	2,4 Ghz	-	250 m	6 €
CC1101	15,6 mA	1,8-3,6 V	433 Mhz	FSK, GFSK e MSK	300 m	8 €
XBee Pro	215 mA	3,3 V	2,4 Ghz	Zigbee	1500 m	32 €

Daqui a escolha recaiu sobre o módulo sem fios XBee Pro porque apesar do preço tem excelente alcance e utiliza o método Zigbee que é bastante fiável e eficiente.

5. ARQUITETURA GERAL

Assim escolhido o *hardware* foi necessário ter a noção da arquitetura geral do sistema a desenvolver, como está descrita na Figura 30. Na parte do hexápode vai conter o Rapsberry Pi como unidade principal de processamento, ligado à unidade principal está o controlador de servos PCA9685, o qual se coneta os respetivos 18 servos MG996R que vão controlar o hexápode. Também ligado ao Raspberry Pi fica o módulo sem fios da XBee Pro. Na parte do telecomando vai conter o Arduino Mega 2560 como unidade principal de processamento, ligado à unidade Arduino estão os botões, os analógicos joysticks e o LCD num *shield*, isto para enviar os sinais para controlo do hexápode e observar o menu. Também ligado à unidade principal está o módulo sem fios XBee Pro.

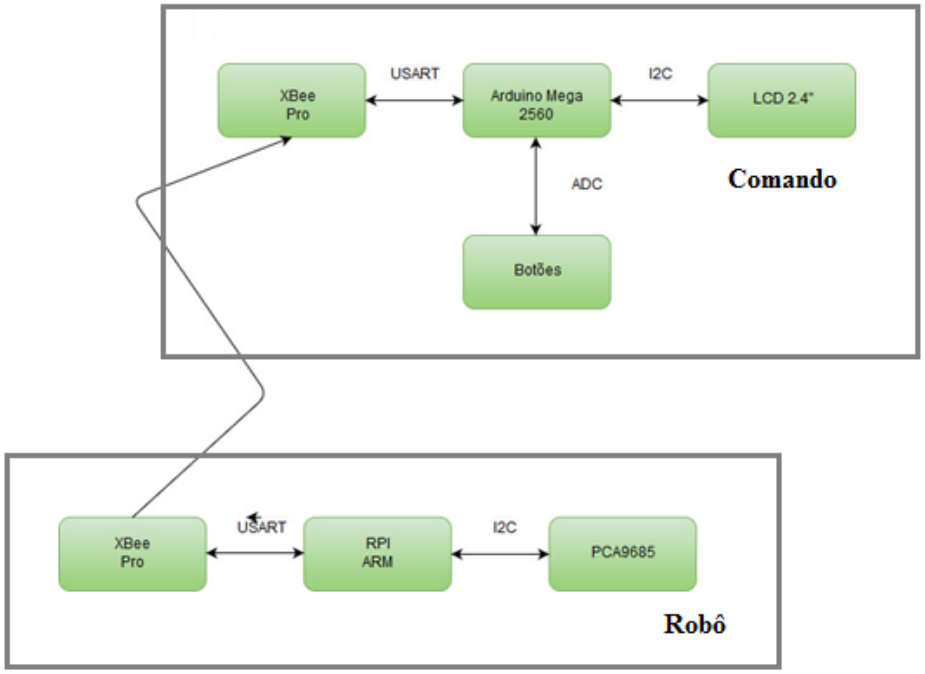


Figura 30 Diagrama geral do projeto

Após a definição da arquitetura geral foram de seguida efetuadas as ligações e inicializações de cada parte do robô singular.

5.1. LIGAÇÕES DO ROBÔ

Na parte do robô, foi ligado à unidade principal Raspberry Pi o módulo sem fios XBee, dois controladores servo PCA9685 com 9 servos cada acoplados. Foi dimensionado uma fonte de energia de 5 V / 10 A para cada controlador dos servos, a escolha de 10 A para cada controlador é devido à carga máxima que cada servo debita, como cada um consome o máximo de 900 mA, numa placa pode atingir 9 A se cada servo estiver em esforço máximo, portanto o indicado foi 10 A, e uma fonte de 5 V / 2 A para a unidade principal.

Na Figura 31 está descrito o esquema de ligações onde o módulo XBee foi ligado por USART, usando a linha representada por castanho Tx e a linha verde Rx para dados, a

linha de alimentação é a vermelha de 3,3 V e a linha azul de *Ground*. Os controladores PCA9685 foram ligados por I2C, usando a linha representada por amarelo SDA e a linha laranja SCL e as respetivas linhas de alimentação.

Os servos foram ligados ao controlador PCA9685 por meio de 3 linhas, a linha amarela (laranja) ao S que representa *Signal* e recebe os PWM, a linha vermelha ao V que representa Vcc e a linha castanha ao G que representa *Ground*, isto ocupando 9 servos para cada controlador. A sua disposição em relação aos membros do chassis é explicada mais à frente. O controlador da direita está ligado por I2C ao da esquerda e tem o *shunt* no A0 para alterar o seu endereço para 0x41 visto que o endereço por omissão do PCA9685 é 0x40.

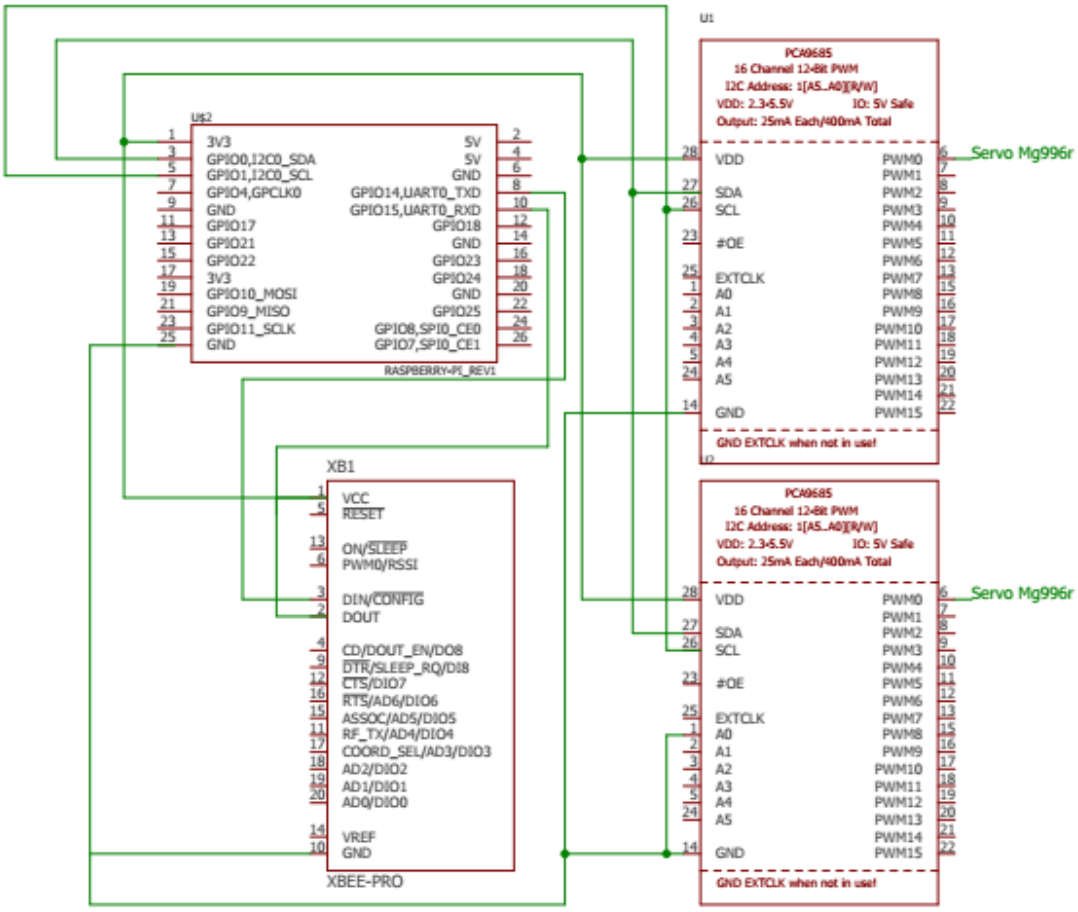


Figura 31 Ligações Rapsberry Pi

Assim concluídas todas as ligações elétricas entre os componentes eletrónicos foi feita a ligação do chassi aos servos, concluindo a montagem de todas as juntas e elos assim como membros. O aspeto final está ilustrado na Figura 32.

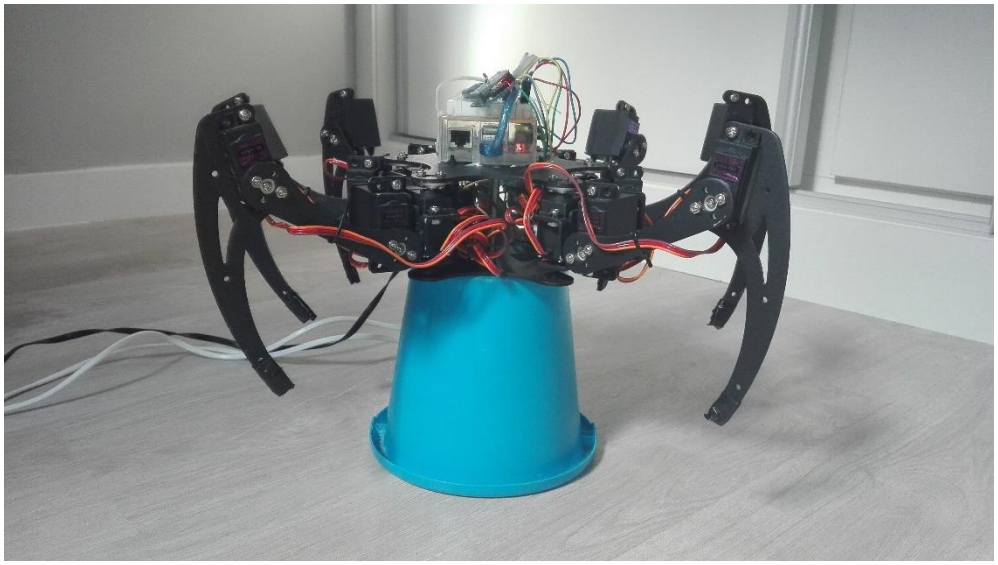


Figura 32 Aspeto do Robô após montagem

5.2. LIGAÇÕES DO COMANDO

Na Figura 33 representa a parte do comando onde foi ligado à unidade principal Arduino Mega 2560 uma alimentação de 5 V / 1 A, o LCD da Inhaos de 2'' por SPI, o módulo sem fios da XBee por USART representado pelas linhas amarelas RX e TX, um analógico com as linhas verdes ligadas às portas analógicas, e dois botões táteis incorporados num *shield* da funduino, que contém o analógico e quatro botões táteis.

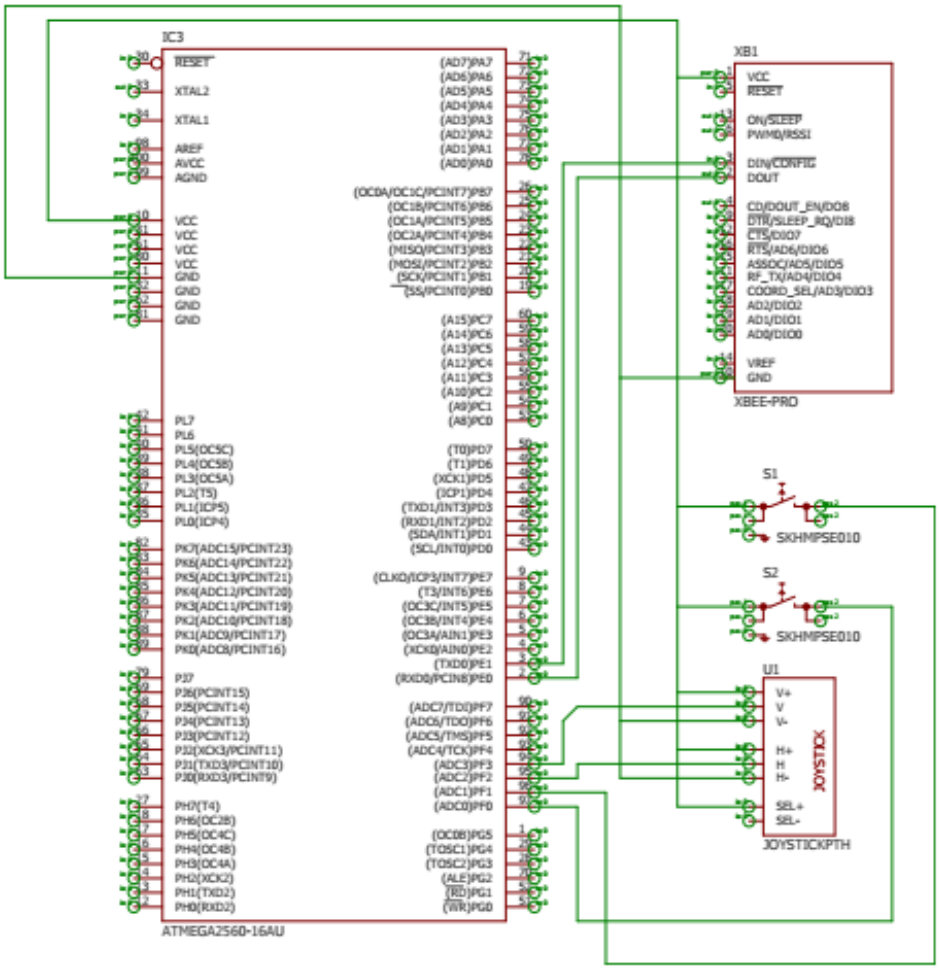


Figura 33 Ligações do Arduino Mega

5.3. INICIALIZAÇÕES DO ROBÔ

5.3.1. RASPBERRY PI

Na unidade do Raspberry Pi foi preciso alimentar com uma fonte de 5 V / 2 A e conter o sistema operativo instalado num cartão SD, neste caso foi usado o Raspbian com o aspeto da Figura 34.

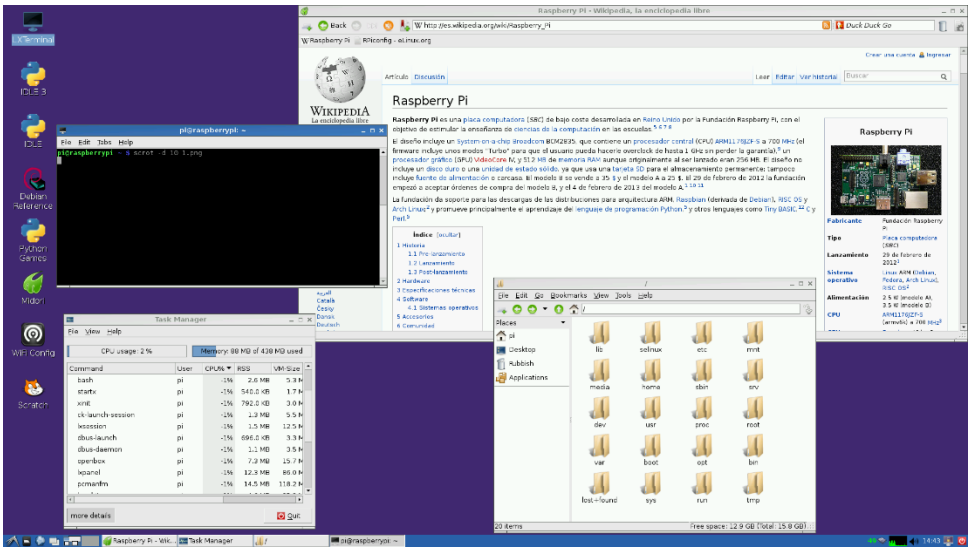


Figura 34 Raspbian

Em termos de configurações foi necessário aceder ao seu terminal *root* e à opção *raspi-config* onde é permitido o ativar do uso das comunicações série, como a I2C e USART. De seguida foi necessário fazer o *download* das ferramentas para programar e editar códigos, neste caso foi executado o comando *Apt-get Install* para realizar o *download* do editor de texto Gedit, e das ferramentas de Python 3 para a programação do seu GPIO. As bibliotecas para controlo dos servos foram descarregadas usando o Github como fonte dessa informação, a biblioteca do servo usado foi desenvolvida pela Adafruit e para o módulo sem fios foi a biblioteca da Python *Serial* instalada a partir do pacote *Apt-get install Python Serial*.

5.3.2. **PCA9685**

Os controladores PCA9685 foram conectados por meio do Protocolo I2C ao Raspberry Pi e alimentados cada um por 5 V / 10 A, ambos são controlados pela biblioteca desenvolvida pela Adafruit que permite comunicar com um módulo PCA9685 a partir do seu endereço, neste caso foram usadas duas unidades, uma com o endereço 0x40 e outra com endereço 0x41. Como são usados 18 servos foi preciso dividir em 9 para cada controlador, o de endereço 0x40 controla os 9 servos do hemisfério direito e o do

endereço 0x41 o do hemisfério esquerdo, mas isto será melhor explicado no próximo capítulo de vetor de testes.

Esta biblioteca permite também parametrizar o impulso de cada servo. Foram usados os parâmetros de defeito da Adafruit que contém o impulso mínimo e máximo a dar, assim como a sua frequência de resolução e comprimento. A principal função a usar foi a *setPWM*(*channel*, *on, off*) em que recebe três argumentos [32], dos quais:

1. *Channel* – Canal que pretende ativar, valor de 0 a 15 pois cada módulo PCA9685 permite ligar 16 servos;

2. *On* – Tempo em que pretende que o PWM esteja ativo, valor de 0 a 4096;

3. *Off* – Tempo em que pretende que o PWM esteja desativo, valor de 0 a 4096.

Esta função deixa controlar os servos ligados a cada canal com um impulso definido pelos argumentos mencionados.

5.3.3. **MG966R**

Os servos foram ligados ao controlador PCA9685 com as 3 linhas, castanha (preta) ao *ground*, vermelha ao VCC e laranja (amarela) ao PWM.

Como o impulso PWM tem uma duração de 20 ms, ou seja, 50 Hz, o *duty cycle* deste impulso vai ter que ser dividido em quatro, em que 25% vai ser a posição 0º, 50% vai ser a posição neutra ou 90º e 75% será 180º, como demonstrado na Figura 35.

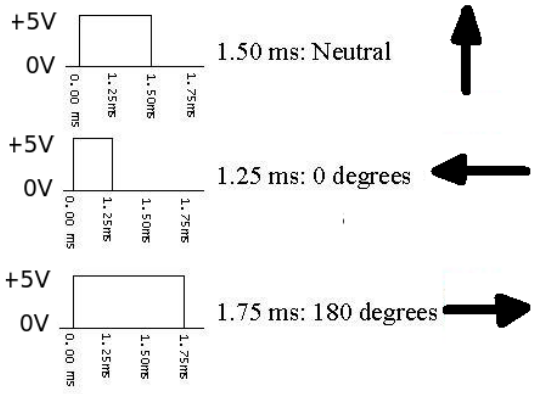


Figura 35 Impulsos PWM para servo [32]

Estes servos internamente possuem um controlador PID de posição que com o auxílio de lógica digital e um potenciómetro permite ter capacidade de obter uma boa precisão.

5.3.4. XBEE PRO S1

O módulo sem fios da XBee foi ligado ao Raspberry Pi e é alimentado com 5 V com o uso de um *shield* que converte o USART em USB por meio de um integrado FT232RL. Este é controlado pela biblioteca Python Serial para interface com o módulo. As funções usadas são o *serial.Serial* para atribuir a uma variável o endereço ou porta do módulo, neste caso é um USB com o endereço /dev/ttyUSB0. As funções para transferência de dados é a *serial.write* para escrever dados e o *serial.read* para realizar a leitura de dados provenientes do módulo.

5.4. INICIALIZAÇÕES DO COMANDO

5.4.1. ARDUINO

Na unidade principal de processamento é preciso a alimentação de 5 V / 1 , assim como realizar a programação a partir do *software* disponibilizado da Arduino, com o aspeto da IDE na Figura 36. A inclusão da biblioteca *LCD_2000_7775.h* é necessária para o LCD. No caso do módulo XBee usa a comunicação série já intrínseca e carregada por defeito nos Arduinos assim como as ferramentas e funções para controlo dos botões digitais e o analógico de dois eixos. Na Figura 36 está demonstrado um exemplo de código de piscar um LED.

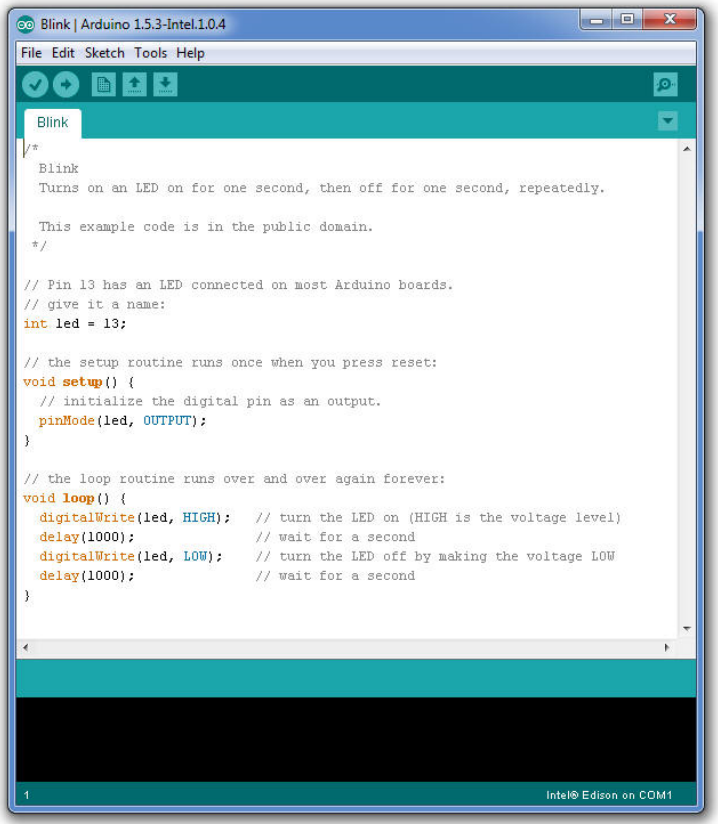


Figura 36 IDE Arduino

5.4.2. **LCD**

O LCD está incluído num *shield* próprio (Figura 37) que conecta ao Arduino de forma simples, usa comunicação série SPI ou digital com 8 bits, para realizar operações de escrita. A biblioteca foi descarregada do *site* dos vendedores com nome "*LCD_2000_7775.h*", e engloba as funções para imprimir informação na tela assim como as definições de cada pino do Arduino mega 2560. A função *TFTscreen.setdatapin* é para declarar os pinos para o uso de 8 bits, o *TFTscreen.background* para declarar a cor de fundo, *TFTscreen.stroke* para a cor do texto, *TFTscreen.setTextSize* declara o tamanho do texto e por fim *TFTscreen.text* declara o valor a imprimir e a sua posição no ecrã.

Figura 37 LCD Inhaos

5.4.3. **XBEE PRO S1**

O módulo sem fios da XBee foi ligado ao Arduino por USART e é alimentado com 5 V, sendo controlado pela biblioteca *Serial* para interface com o módulo. As funções usadas é o *Serial.begin*() que permite iniciar a porta série com valor de 19200 bps, o *Serial.print* que escreve dados para a porta série, existe ainda o *Serial.available* para verificar se existe dados na porta série e o *Serial.read* para ler os dados da porta série.

5.4.4. BOTÕES E ANALÓGICO

Os botões táteis usados foram dois, B e C, mais os analógicos de dois eixos, X e Y, todos eles usaram pinos analógicos do Arduino, A0 até A3. Desta forma foi possível fazer a leitura da tensão do pino e converter esse valor, de 0 a 5 V, diretamente num intervalo, de 0 a 1023, a partir da função *analogRead*. A Figura 38 mostra o aspeto do *shield* mencionado com os botões táteis e analógicos previamente retratados.

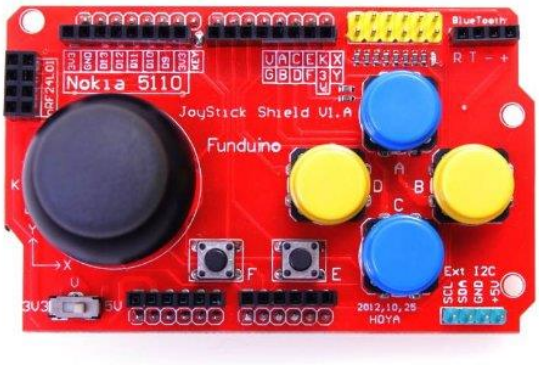


Figura 38 *Shield* da Funduino

6. TESTES REALIZADOS

Nesta fase do trabalho o crucial é colocar cada componente a comunicar com a sua respetiva unidade principal de processamento. Este capítulo aborda como cada bloco interage entre si assim como alguma programação. Vai ser dividido em subcapítulos separando o robô e a sua unidade principal Raspberry Pi, do comando e a sua unidade Arduino. Na parte do robô vai ser testado a comunicação com o módulo sem fios e o enviar e receber dados, assim como o controlador de servos e todas as configurações realizadas. Na parte do comando vai ser testado o módulo sem fios à semelhança do robô, o controlo do LCD e a leitura dos botões e analógico.

6.1. **ROBÔ**

No robô encontra-se a unidade Raspberry Pi e vai-se testar o módulo XBee, o controlador de servos PCA9685 e os respetivos servos MG996R.

6.1.1. XBEE S1

Para o bloco do módulo sem fios foi preciso realizar programação Python no Raspberry Pi [33]. Este já foi previamente inicializado para comunicação série e descarregadas as bibliotecas necessárias. É então aberto uma página no editor de texto e programado um pequeno algoritmo que recebe e envia dados, tal como descreve a Figura 39.

Este programa inicia com o *import serial* que vai buscar a biblioteca para comunicação série, após essa inicialização vai atribuir a uma variável chamada *ser* o endereço do USB que vai ligar ao *shield*, este por sua vez converte série para USB. Depois imprime localmente com a função *print* no terminal uma mensagem "A enviar: Ola do Raspberry Pi" e depois escreve a mensagem guardada na variável *string* que contém "Ola do Raspberry Pi" com a função *ser.write*. Concluído esta fase fica à escuta de resposta do outro módulo XBee num ciclo *while*, com a função *ser.read*, faz a leitura do que enviou o outro módulo, e imprime localmente no terminal com a função *print* "Recebido (carater recebido) ", por fim escreve "Rpi recebeu: (carater recebido) " com o *ser.write* para depois enviar a informação para o outro módulo.

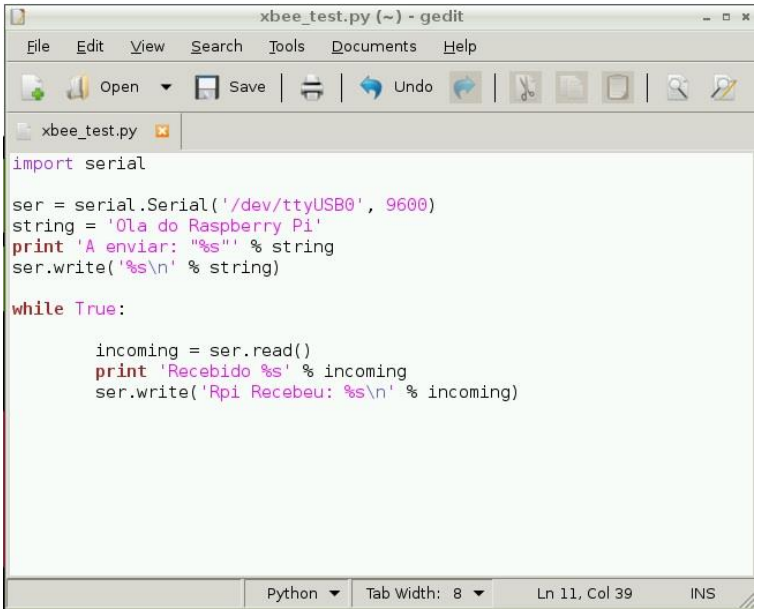


Figura 39 Código de Python para XBee

Este programa foi colocado a correr com auxílio das ferramentas de Python 3. O módulo que vai receber os dados foi ligado ao PC para testes apenas, este foi ligado por USB pelo *shield* e testado no *software* XCTU [35]. Na parte do XCTU recebe-se a mensagem "Ola do Rapsberry Pi", responde-se "ola" e o Raspberry Pi fez o eco de cada letra respondendo "Rpi Recebeu: o" e o mesmo para "l" e "a", como é possível ver na Figura 40.

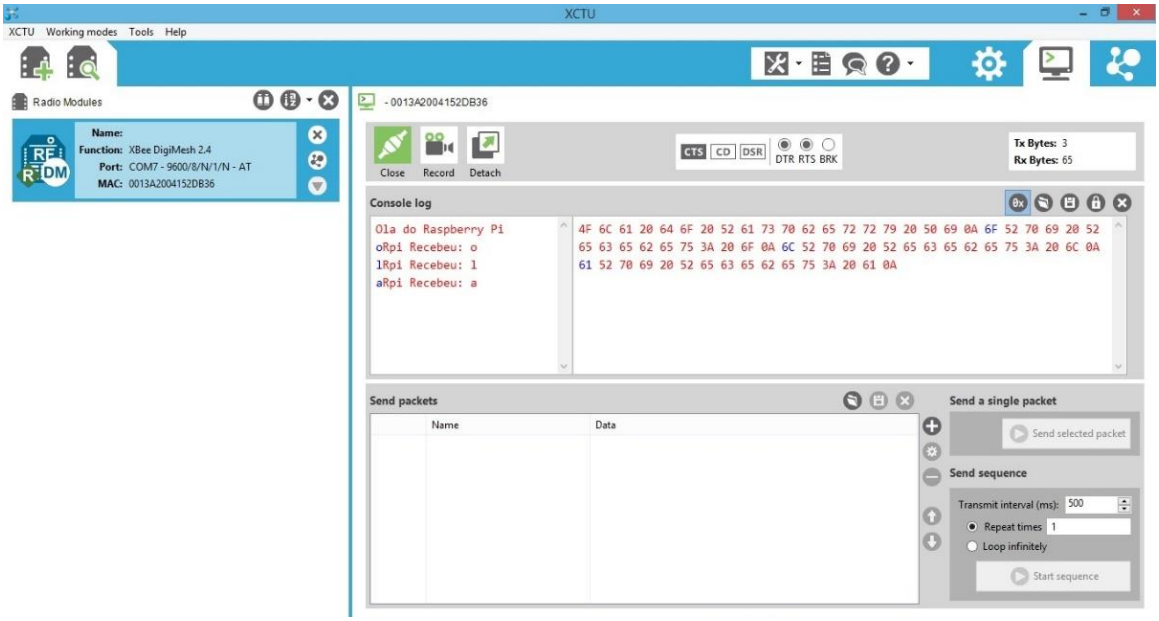


Figura 40 Teste entre Rpi e XCTU

6.1.2. **PCA9685**

Este bloco exige uma programação em Python assim como o módulo sem fios. Aqui é importado as bibliotecas descarregadas da Adafruit e configurado o endereço dos módulos [34]. A designação SD é para Servocontrolador Direito e SE para Servocontrolador Esquerdo, isto porque o hexápode visto de cima é dividido na vertical em duas secções simétricas, cada lado vai conter 9 servos relativos às três patas, como demonstrado na Figura 41.

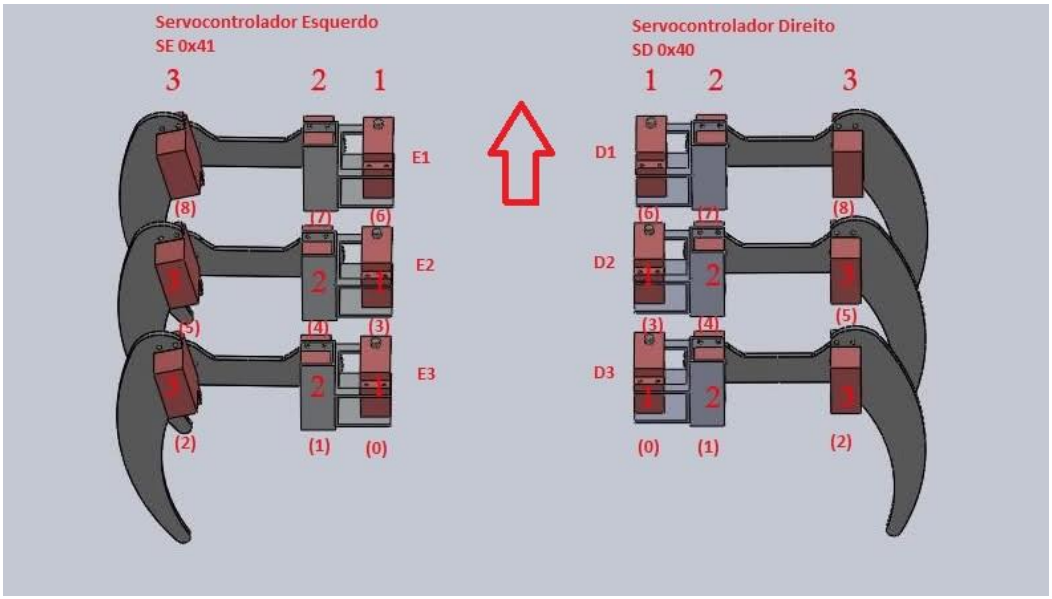


Figura 41 Posição e numeração de cada pata do robô

A seta representa o sentido de movimento para a frente, os valores entre parêntesis são relativos aos canais a que os servos estão ligados nos seus respetivos servocontroladores que vai de 0 a 8. Os valores 1, 2 e 3 são a numeração dos servos que atuam uma junta, o 1 é a coxa, 2 fémur e o 3 tíbia, isto para melhor conciliar com a informação dada na locomoção de hexápodes. Por fim, a descrição E1, E2 e E3 assim como D1, D2 e D3 é para representar uma pata na sua totalidade.

De seguida é declarado os limites mínimo e máximo do impulso PWM assim como a sua largura, frequência e resolução. Estes valores foram fornecidos já no código, portanto não foi alterado nada, o impulso mínimo corresponde aos 0º e o impulso máximo aos 180º.

Num ciclo *while* é testado as posições de cada servo. Numa configuração inicial foi colocar todos os servos numa posição média definida pelo 0º e que é calculado por:

$$servoMed = \frac{servoMax + servoMin}{2} \quad (24)$$

No caso do ângulo β sofreu uma deslocação de -90º, logo tem um *offset* de -90º. Deste modo foi possível afinar as posições de todas as juntas de forma a ficar como na Figura 42.

Figura 42 ServoMed em todos os servos visto de frente

Na Figura 43 é visto a disposição das patas de forma a ficarem uniformes.

Figura 43 ServoMed em todos os servos visto de cima

Destas posições é possível movimentar as juntas para +90º e -90º que é o intervalo permitido teoricamente no servo. Isto claro que é usado com cautela porque não será possível usar esse intervalo em todas as juntas para evitar colisões. Prosseguindo o teste foi verificado a posição no impulso máximo e mínimo de cada junta e dessa forma perceber os sentidos de cada movimento.

A partir daqui foi preciso realizar os cálculos necessários para obter os movimentos mais próximos a de um padrão de locomoção hexápode e o grau de movimento das juntas inerentes a essas locomoções.

6.2. COMANDO

No robô encontra-se a unidade Arduino Mega e vai-se testar o módulo XBee, o LCD Inhaos e o *shield* dos botões e analógicos [37].

6.2.1. XBEE S1

Para o bloco do módulo sem fios foi preciso realizar programação no IDE da Arduino. Como já estão carregadas as bibliotecas precisas é então aberto uma página no IDE e programado um pequeno algoritmo que recebe e envia dados, tal como mostra a Figura 44.

Este programa contém uma inicialização da comunicação série com 19200 bps dado pela função *Serial.begin*, e imprime localmente no terminal uma mensagem "A enviar: Ola do Arduino" com a função *Serial.Println*, agora envia para o outro módulo "Ola do Arduino" com a função *Serial.write.* Depois fica à escuta de resposta num ciclo *while* que aguarda a receção de dados com uma função *SerialEvent*, a qual consiste numa função *Serial.available* que verifica se existe dados na porta série e lê esses dados com *Serial.read,* coloca-os numa *string* e por fim escreve "*Arduino* recebeu: " e a seguir a informação recebida.

Figura 44 Código de Arduino para XBee

Este programa foi colocado a correr à semelhança do usado no Rapsberry Pi. O módulo que vai receber os dados foi ligado ao PC para testes apenas, este foi ligado por USB pelo *shield* e testado no *software* XCTU como feito anteriormente e demonstrado na Figura 45.

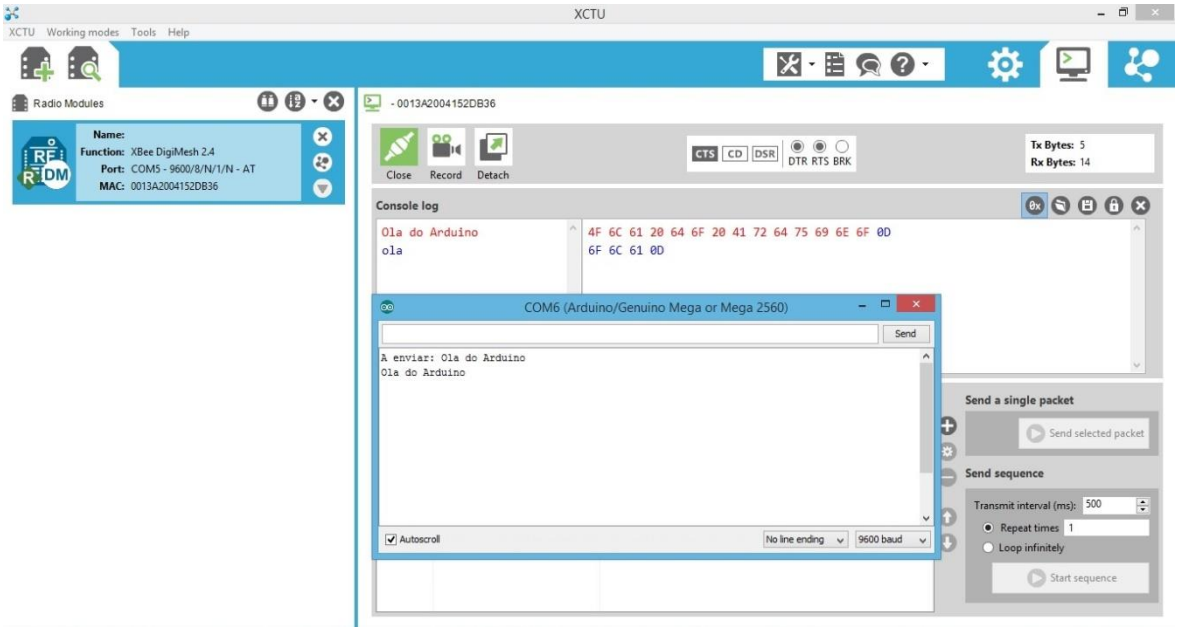


Figura 45 Teste entre Arduino e XCTU

6.2.2. **LCD**

No caso do LCD foi usado a biblioteca *LCD_2000_7775.h* disponibilizada pelo fabricante que é o nome do modelo em causa vendido pela Inhaos [36]. Esta contém todas as funções necessárias para fazer as operações de impressão no LCD, sendo utilizada para o caso de testes o exemplo que vem com a biblioteca que imprime uma mensagem "Sensor *Value*:" isto na primeira linha e na segunda o valor analógico da porta A0.

Este exemplo, apresentado na Figura 46, inicia uma instância da biblioteca com a função *TFTscreen* e o uso dos pinos para o modo de 8 bits com o *TFTscreen.setdatapin*, a função *TFTscreen.begin*() para o arranque do LCD, a função *TFTscreen.background*(r, g, b) para definir a cor do fundo com a conjugação dos parâmetros r, g e b que significa respetivamente vermelho, verde e azul variando de 0 a 255. Neste exemplo os valores são todos 0 para ter um fundo preto. A função *TFTscreen.stroke*(r, g, b) é semelhante à anterior *TFTscreen.background* mas é em respeito à cor dos carateres, sendo atribuído o valor de 255 de cor branca. A função *TFTscreen.setTextSize*() que recebe o valor do tamanho do carater a imprimir, que foi 2, para a parte "Sensor Value:", e de 5 para o valor lido na porta analógica. Por fim a *TFTscreen.text*(*string*, x, y) com o parâmetro *string* que se pretende imprimir, x é a coordenada com orientação na horizontal e y é a coordenada com a orientação na vertical. Na parte de "Sensor Value:" foi com coordenadas (0,0) que equivale à parte superior esquerda do LCD para iniciar a escrita, a parte do valor do sensor foi (0,20) que equivale à linha abaixo. No ciclo *loop* recebe o valor do pino A0 com um *analogRead* e vai converter a *string* lida num vetor de carateres.

```
File  Edit  Sketch  Tools  Help

TFTDisplayText_Mega2560 §
#define D6   31
#define D7   30

LCD_2000_7775 TFTscreen(cs,wr,rs,rst);

char sensorPrintout[4];

void setup() {


  TFTscreen.setdatapin(D0,D1,D2,D3,D4,D5,D6,D7);
  TFTscreen.begin();
  TFTscreen.background(0, 0, 0);
  TFTscreen.stroke(255,255,255);
  TFTscreen.setTextSize(2);
  TFTscreen.text("Sensor Value :\n ",0,0);
  TFTscreen.setTextSize(5);
}

void loop() {
  String sensorVal = String(analogRead(A0));
  sensorVal.toCharArray(sensorPrintout, 4);
  TFTscreen.stroke(255,255,255);
  TFTscreen.text(sensorPrintout, 0, 20);
  delay(250);
  TFTscreen.stroke(0,0,0);
  TFTscreen.text(sensorPrintout, 0, 20);
}
```

Figura 46 Código de Arduino do LCD

A partir deste código obtém-se uma representação com o aspeto demonstrado na Figura 47. O LCD imprime “Sensor Value :” e depois o valor lido na porta analógica.

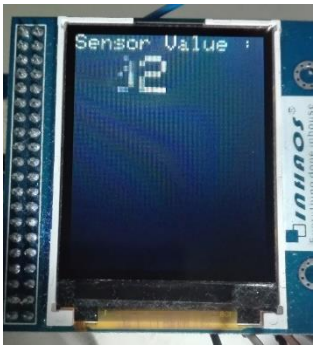


Figura 47 Teste de Arduino no LCD

6.2.3. BOTÕES E ANALÓGICO (SHIELD)

Por fim foi testado o analógico e botões recorrendo ao auxílio do terminal do Arduino, isto é foi colocado uma inicialização de comunicação série com 9600 bps e a declaração das variáveis correspondentes a cada botão, A, B, C, D, E, F e os analógicos, X e Y. Tal como mostra a Figura 48 na secção *Setup* são iniciadas cada variável com o *pinMode* como tipo digital e atribuídas como entradas. Na parte *loop* é lido cada valor analógico e valor digital de cada botão tátil, sendo impresso no terminal com uso da *Serial.println.*

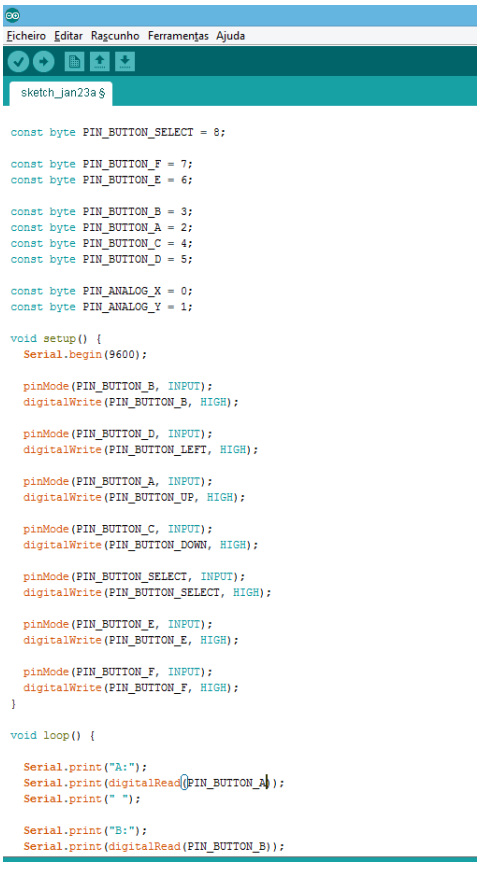


Figura 48 Código Arduino dos botões e analógico

Na Figura 49 é demonstrado o aspeto do *shield* usado para facilitar a interface com o microcontrolador.

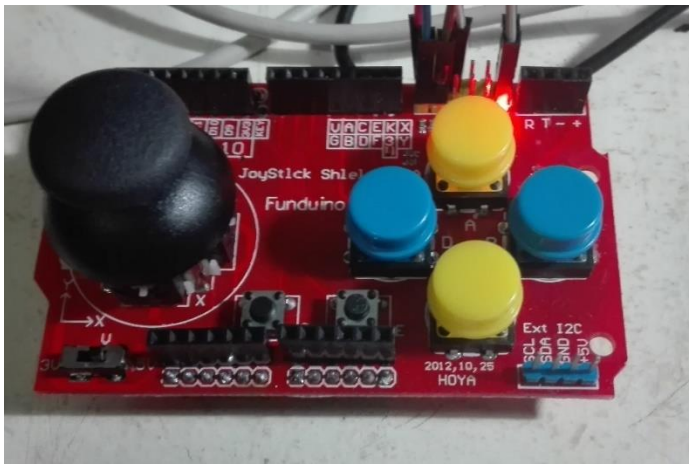


Figura 49 *Shield* Arduino

7. IMPLEMENTAÇÃO DO ROBÔ

Na última fase do trabalho é composto o código final no comando e no robô hexápode. No comando é explicado o algoritmo e a aparência do menu que vai operar nele para ser possível comunicar a partir do XBee com o robô. No robô é explicado o desenvolvimento de cada padrão de locomoção assim como um comparativo entre cada um e as respetivas conclusões retiradas.

7.1. DESENVOLVIMENTO DO MENU DO COMANDO

Nesta fase do projeto foi preciso desenvolver uma interface amigável e intuitiva no comando, cujo aspeto final está na Figura 50. Para comunicar com o robô são usados comandos enviados pelo XBee e interpretados pelo Arduino.

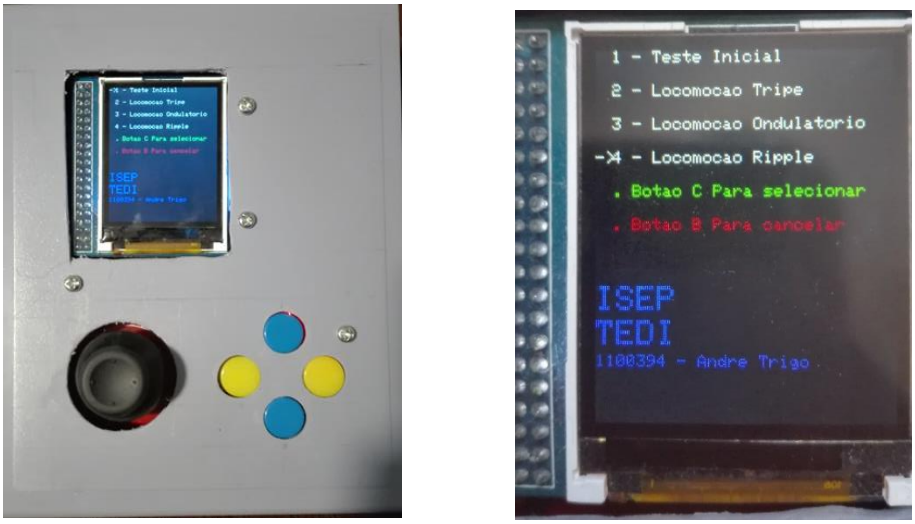


Figura 50 Comando do Hexápode

Na imagem é percetível a posição do LCD em cima, na parte de baixo à esquerda o *joystick* analógico, na parte de baixo à direita os botões.

As opções colocadas foram as seguintes:

- Teste Inicial
- Locomoção Tripé
- Locomoção Ondulatório
- Locomoção *Ripple*

O fluxograma equivalente do menu é demonstrado a seguir na Figura 51.

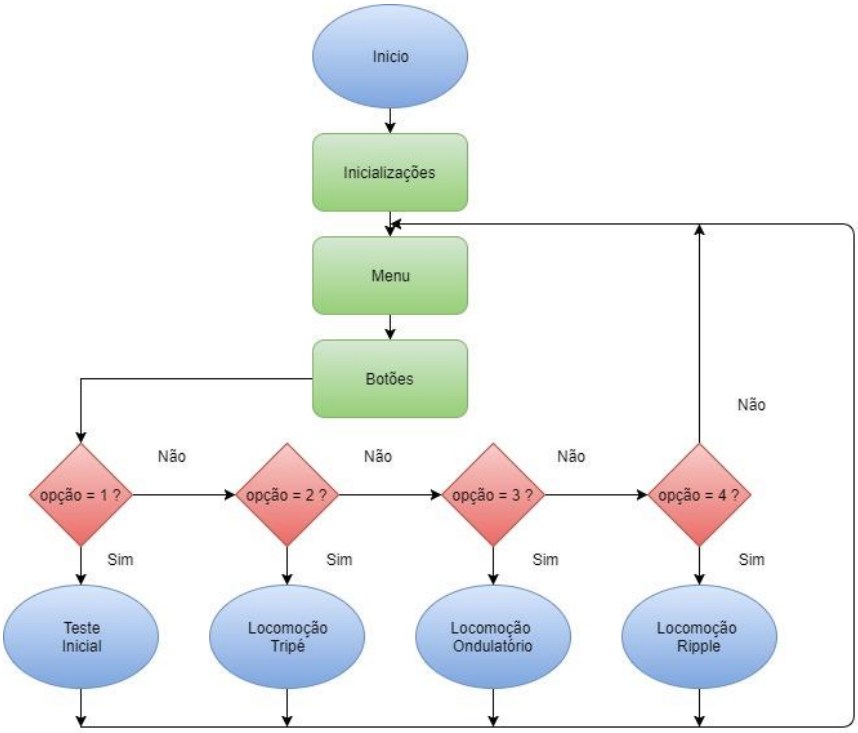


Figura 51 Fluxograma do menu do comando

Como é permitido perceber pelo fluxograma, o código, do anexo B, é composto pelas inicializações principais do programa Arduino, desde a porta série, entradas e saídas analógicas e as parametrizações para o LCD, como foi explicado anteriormente no vetor de testes.

O bloco menu é onde é desenhado o menu no LCD e onde é feito um ciclo *while* que permanece neste menu sempre, ou seja, o *main* é o menu principal que é onde todas as opções vão retornar. No bloco botões são lidas as portas analógicas de A0 a A3 tal como declarado nos testes anteriores, a partir dos valores é colocado um estado numa variável que permite saber qual o botão premido e a partir de aí operar o menu principal, desde navegar as opções para cima ou baixo, entrar e sair das mesmas, onde em cada opção é permitido colocar informação na porta série e enviar pelo XBee para o robô.

Na primeira opção, Teste Inicial, é enviado um grupo de carateres “TI” que simbolizam o início do algoritmo para o teste inicial, este é um procedimento relativamente simples que testa todos servos do robô para ser visualizável se todos se encontram em

funcionamento. Após um ciclo é possível retornar ao menu anterior pressionando o botão B.

Na segunda opção, Locomoção Tripé, é enviado um grupo de carateres “LT” que simbolizam o início do algoritmo para a locomoção tripé, este é um ciclo que envia os carateres correspondentes à deslocação que se pretende, se no analógico puxar para cima envia o carater “F”, esquerda “E”, direita “D” e trás “T”, botão B para regressar “B” .

Na terceira opção, Locomoção Ondulatório, é enviado um grupo de carateres “LO” que simboliza o início do algoritmo para a locomoção ondulatório; este é igual à locomoção anterior e envia os carateres correspondentes à posição do analógico, a diferença é apenas na interpretação do robô que difere a locomoção.

Na quarta e última opção, Locomoção *Ripple,* é enviado um grupo de carateres “LR” que simboliza o início do algoritmo para a locomoção *ripple*; este é igual às duas locomoções anteriores e envia os carateres correspondentes à posição do analógico, sendo a diferença apenas na interpretação do robô que difere a locomoção.

7.2. PARAMETRIZAÇÃO E SIMULAÇÃO DO ROBÔ

Esta secção descreve o processo de implementação de um robô hexápode assim como o modelo matemático 3D que se deseja projetado. Este modo é capaz de dar as posições da extremidade de cada uma das seis pernas do robô em relação ao centro do corpo. As extremidades das pernas são também conhecidas como pés. Cada perna tem três graus de liberdade (3 DOF), permitindo que o robô se mova lateralmente e também em rotação. O modelo é cinemático, significando que não há dinâmicas envolvidas como massa, atrito e gravidade.

7.2.1. DIMENSÕES E LIBERDADE DE MOVIMENTOS

O modelo é baseado no chassis já descrito anteriormente da Phoenix, a partir deste primeiro protótipo, as dimensões das pernas são tomadas e implementadas como parâmetros. Os parâmetros ainda podem ser atualizados para trabalhos futuros, mas são mantidos para um ponto de partida para simulações. O projeto do corpo do robô hexápode é escolhido de forma que o tamanho do robô com todos os elos abertos estará dentro das dimensões 0,6 m x 0,63 m x 0,44 m (largura x comprimento x altura) podendo também ser equivalente às coordenadas no espaço (*x*, *y* e *z*) [38].

Assim de seguida definiram-se bem os parâmetros que estão em jogo, cada pata possui três juntas e três elos, todas as juntas permitem uma amplitude de movimento de 180º, e cada elo tem um comprimento, daí se extrai a informação contida na Tabela 7.

Tabela 7 Parâmetros das juntas e elos

Coxa	0,030 m
Fémur	0,085 m
Tíbia	0,130 m
γ	0 - 180º
α	0 - 180º
β	0 - 180º

Apesar de cada junta permitir 180º na prática é preciso restringir esse intervalo para evitar colisão com outra pata, da junta γ advém um cuidado maior. Esta junta foi reduzida a 30º para cada lado sensivelmente, as restantes foram ajustadas com outros valores, mas nunca utilizam a liberdade total.

A Figura 52 mostra a pata do robô, as partes das pernas, coxa, fémur e tíbia. As pernas são equidistantes radialmente distribuídas em torno do robô. A direção para a frente está na direção do eixo +*y*.

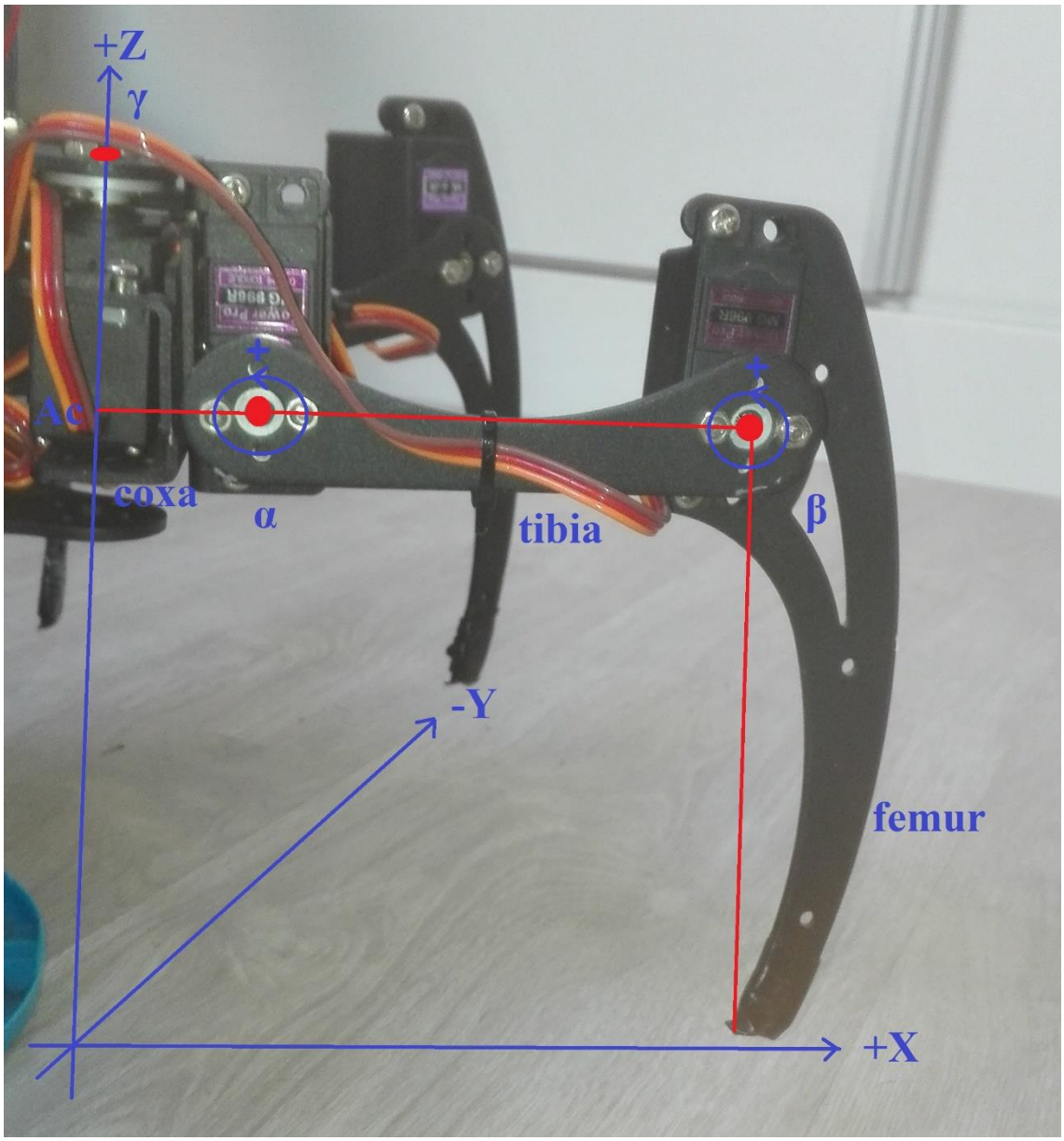


Figura 52 Pata do Hexápode e seus pontos zeros das juntas α e β

A posição angular das pernas é medida a partir da posição declarada no vetor de testes ServoMed, isto é cada perna roda a partir de ServoMed para -90º ou 90º, logo este é o ponto médio de deslocação permitida por cada junta, portanto o valor de referencia inicial. No entanto, como é possível observar pela Figura 52, a junta β está com um *offset* de -90º porque se considerar o ponto zero com a pata toda esticada iria retirar muita amplitude de movimento da pata; o sentido positivo também se encontra marcado em cada junta.

Na Figura 53 é observável os pontos zeros da junta γ e o seu sentido. Todo o código é escrito a partir deste parâmetro, por isso se por algum motivo no futuro se alterar algo em relação às posições, os valores de referência zero mantêm-se.

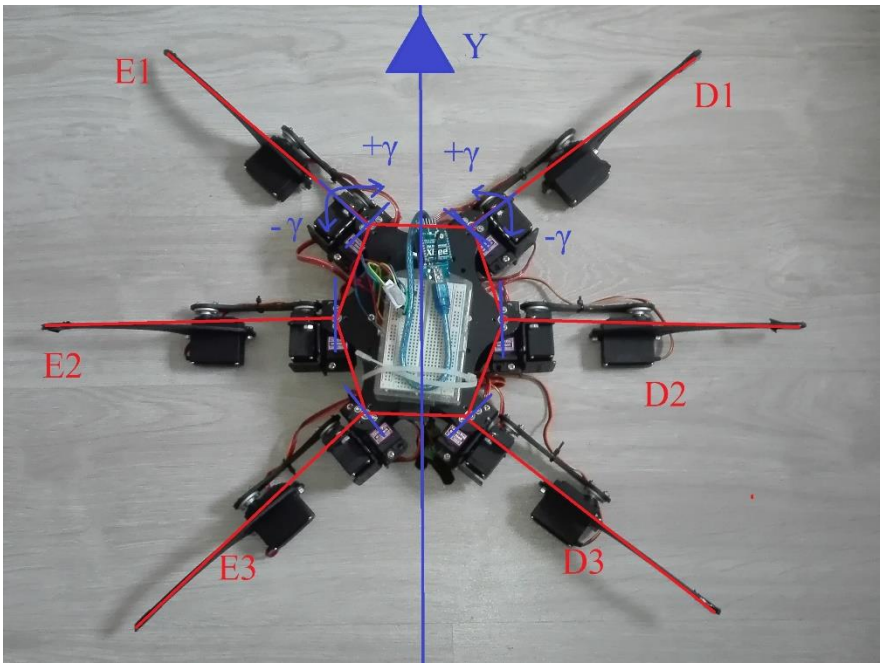


Figura 53 Disposição das patas e posição zero para a junta γ

O primeiro objetivo é determinar a área de trabalho total da perna. Os limites mecânicos das juntas são apresentados na Tabela 8. Para manter a simulação o mais real possível, o valor de 90º nunca deve ser atingido por segurança, logo um desconto de 10º a menos quando positivo, ou a mais se for negativo, é aplicado a cada junta, de modo a impedir o alcance dos limites mecânicos das juntas.

Tabela 8 Limites mecânicos das juntas

Conjunto	**Mínimo valor**	**Máximo valor**
γ	-30º	30º
α	-30º	90º
β	-80º	80º

Para obter uma noção da área de trabalho de uma perna esta é descrita na Figura 54. Os ângulos limites impostos para as juntas são definidos de γ = -30º a 30º e α = -30º a 90º e β = -80º a 80º. O gráfico é representado numa vista 2D (vista superior) para obter uma melhor visão geral da área de trabalho da perna.

A partir disto pode-se ver que a máxima altura possível em relação ao eixo *z* é 0,215 m com os elos fémur e tíbia todos esticados e na vertical [40]. A altura inicial do corpo é 0,130 m, devido à junta β estar na posição ServoMed, o que dá a altura do elo tíbia em relação ao solo.

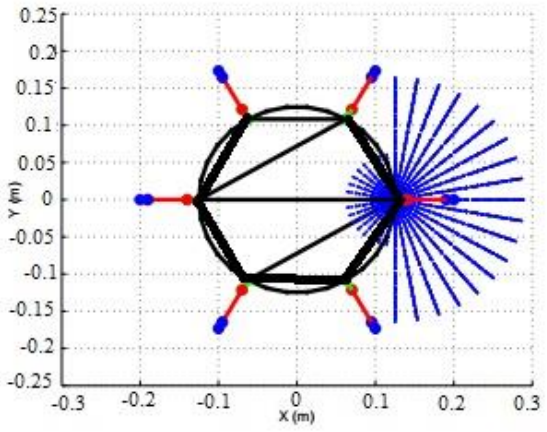


Figura 54 Área de trabalho de uma pata

A superfície de contato com o solo é a projeção dos pés dos robôs no plano horizontal *z* = 0. Essas superfícies de contato mostram que a área em que a ponta da perna pode tocar o chão varia muito com a altura do corpo do robô. A partir de *z* = 130 mm o contato com a superfície não muda muito quando a altura do corpo é aumentada.

O padrão de suporte total não é totalmente adequado, pois quando o ângulo que a junta α faz é menor que -30º em relação ao ponto zero, as forças desenvolvidas não são suficientes para suportar o robô, ou se a perna estiver totalmente esticada, o que tem uma grande amplitude, mas as juntas dificilmente vão ter binário suficiente para levantar o corpo. Para evitar isso, o intervalo da junta α é ajustado de -30° a 80º.

O mesmo se aplica à junta β, pois se ela estiver perto de 80º enquanto a junta α estiver no ponto zero, as patas vão ficar muito esticadas e o robô vai-se aproximar do solo e

deixa de ter força de suporte. No entanto, se α estiver perto dos -80º fica com a ponta da pata muito perto do centro de massa e acontece o mesmo problema, o ideal é encontrar a posição calculada para encontrar o equilíbrio. Dentro deste intervalo, assume-se que a perna tem ainda binário suficiente para levantar o robô [41].

7.2.2. TRAJETÓRIA DAS PATAS NO ESPAÇO

O algoritmo da trajetória dos pés será explicado para a direção *y* local porque o tamanho do passo é idêntico ao sentido *x* local.

Finalmente faz-se a trajetória da perna correta para as coordenadas globais para que as pernas se movam para a direção desejada.

A fim de projetar um algoritmo para a trajetória das pernas algumas suposições são feitas:

1. O corpo do robô tem que se mover com velocidade constante em qualquer direção.

2. A trajetória completa deve ser sincronizado e deve envolver todas as juntas e elos em cada pata.

3. A velocidade da trajetória deve ser igual em qualquer tipo de locomoção.

4. A trajetória de aceleração deve ser continuada sem colisões ou etapas na função. A razão para isso é superar as sobrecargas no motor que alimenta as juntas para que elas possam operar suavemente mesmo quando o sentido de rotação é alterado.

5. A maior velocidade da perna na fase de transferência deve estar entre β e aproximadamente 1 do tempo de ciclo normalizado.

6. As variáveis que influenciam a trajetória devem ser paramétricas: tamanho do passo, altura máxima da ciclóide, distância da extremidade da pata da sua origem, altura do corpo, tempo de ciclo, e *duty cycle*, onde este último não é

livre de se escolher porque depende da marcha, que pode ser metraconal, tripé ou *ripple*.

Esta trajetória foi planeada de forma a que a movimentação da pata descreva uma ciclóide e se desloque o mais eficientemente e harmoniosamente possível de forma a evitar colisões entre patas. Foi já definido previamente um intervalo útil de movimentação permitido para cada junta para ser possível usar em qualquer locomoção.

Esta ciclóide foi primeiramente explorada e simulada em Matlab, foi criado uma função, descrita em anexo A, que calcula as coordenadas no espaço *x*,*y* e *z* com as equações da cinemática direta da ciclóide, e a partir desses valores obtêm-se os ângulos das juntas γ, α e β, pelo princípio da cinemática inversa. Esta função utiliza os parâmetros que recebe, F_C é a altura máxima que se quer que a pata atinja no topo da ciclóide, L_S é a distância que o robô vai deslocar, *T* é o período do movimento, D_C que é o *duty cycle*; coxa, fémur e tíbia são os comprimentos de cada membro da pata, A_C é a altura do corpo do robô ao solo e *T*_amostra é o tempo de cada amostragem.

A função calcula os pontos da ciclóide na sua fase de transferência assim como de suporte a partir das suas equações já previamente expostas. Em seguida o cálculo dos valores da posição da ponta da pata no espaço é realizado, um ciclo calcula o valor de cada junta, γ, α e β na fase de suporte assim como a fase de transferência. Com estes valores é possível esboçar o gráfico do desenvolvimento da ponta da pata em relação aos eixos *z* e *y*, assim como o ângulo das juntas em relação ao tempo.

Os valores usados para a simulação foram:

F_C = 0,06 [m] – Altura máxima da trajetória da ciclóide;

L_S = 0,110 [m] – Deslocação frontal que o robô vai executar por período;

T = 1 [s] – Tempo de um período;

D_C = 0,50 – *Duty cycle* a usar, neste caso ½ que é o tripé;

Coxa = 0,030 [m] – Comprimento do membro coxa;

Fémur = 0,085 [m] – Comprimento do membro fémur;

Tíbia = 0,130 [m] – Comprimento do membro tíbia;

A_C = 0,095 [m] – Altura a que o robô vai ficar do solo em relação à origem das patas;

$T_$amostra = 0,01 [s] – Tempo de amostragem no período.

O *duty cycle* foi de 0,50 para simular a locomoção trípode. No caso das outras locomoções será de 0,83 para a ondulatória e de 0,75 para a *ripple*. A Figura 55 mostra a quase ciclóide ao longo de um ciclo nos eixos *y* e *z*, em que a altura máxima é o valor F_C de 0,06 m e a distância da pata no eixo *x* em relação ao centro de massa, D_X, igual a 0,085 m.

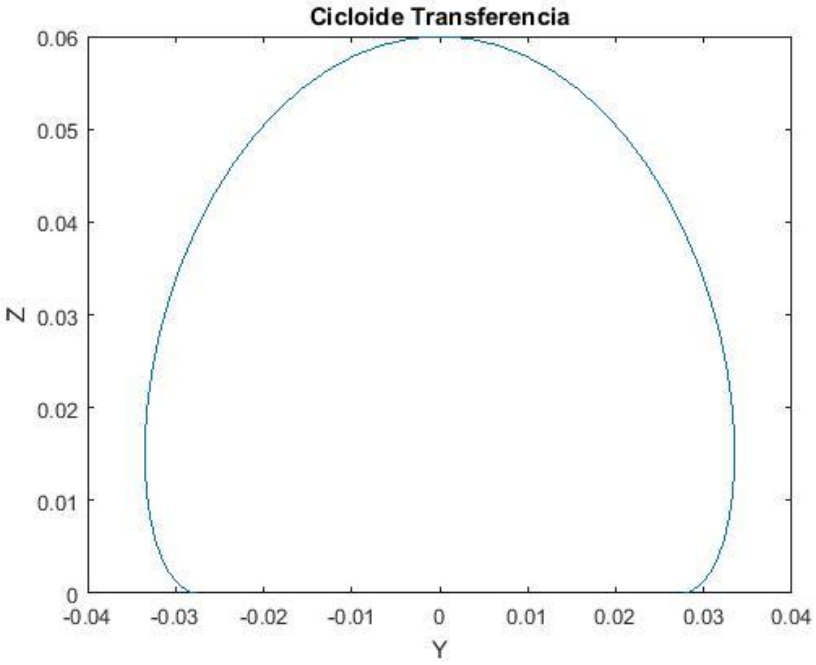


Figura 55 Ciclóide na fase de transferência em locomoção tripé

Na Figura 56 estão os valores do ângulo γ durante um ciclo, onde é possivel observar que o seu valor varia de -21,4º a 21,4º. Estes valores são maioritariamente influenciados pelo parâmetro $L_{S\,,}$ isto porque quanto mais se pretende de movimento frontal maior a amplitude de movimento no ângulo γ.

É possível averiguar os valores que o ângulo α tem ao longo de um período, variando de 23,3º até 86, 8º. Neste caso o parâmetro que influencia mais é o F_C , logo quanto mais alto o ponto máximo da ciclóide maior a amplitude deste ângulo. O parâmetro A_C também influencia porque está dependente da altura do corpo do robô.

Estão presentes os valores do ângulo β num período, variando de -154 a -121, 6º. Neste caso, os parâmetros que mais influenciam é o F_C e o D_X .

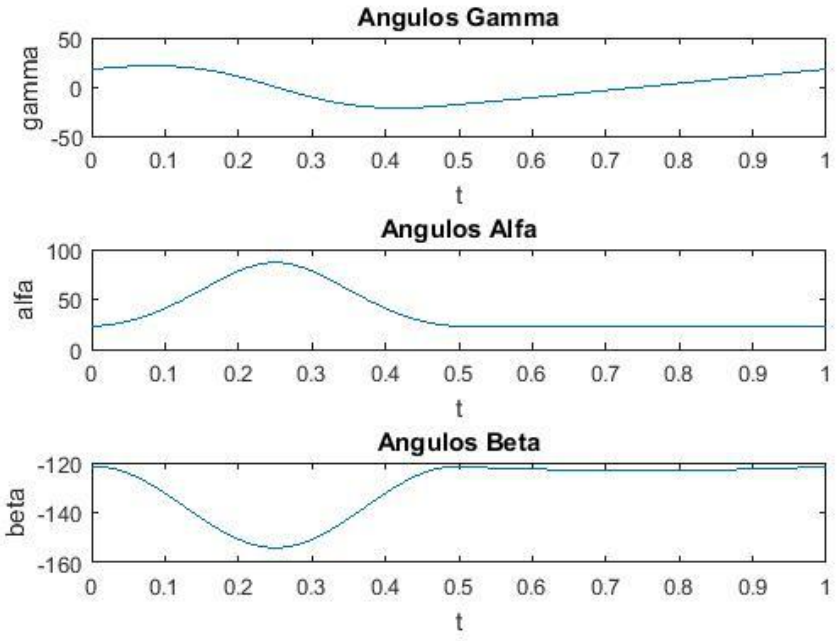


Figura 56 Ângulos γ, α e β ao longo de um período em locomoção tripé

O *duty cycle* foi alterado para 0,83 para a simulação da locomoção ondulatória. No caso dos restantes parâmetros tudo se mantém constante. A Figura 57 mostra a ciclóide ao longo de um ciclo nos eixos *y* e *z*. Comparando à da locomoção tripé é notável a diferença na distância em *y*, assim como a curva se assemelha mais a uma ciclóide convencional.

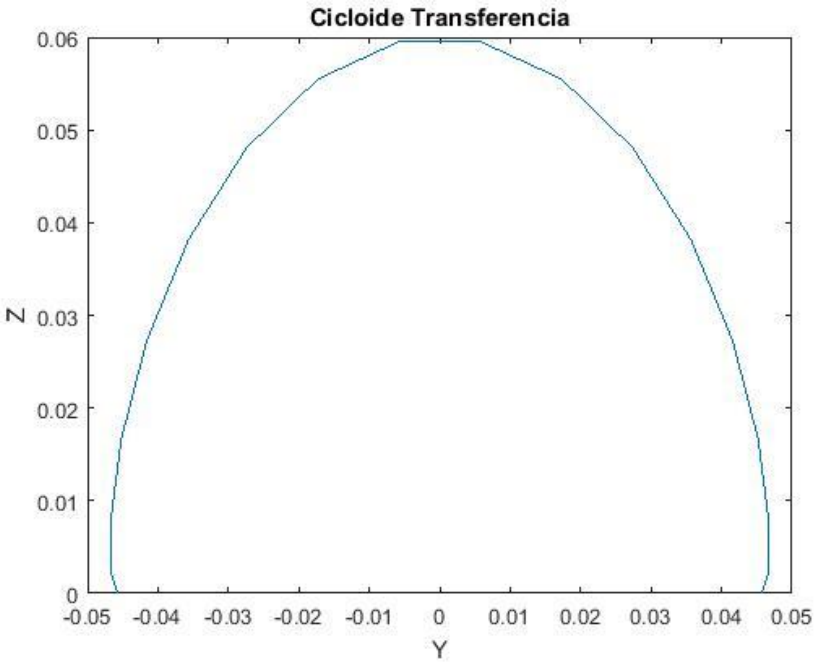


Figura 57 Ciclóide na fase de transferência em locomoção ondulatória

Na Figura 58 estão os valores dos ângulos γ, α e β durante um ciclo, onde é possivel observar que os seus valores variam de forma similar à locomoção tripé em termos de amplitude. A maior diferença está no tempo de suporte, que será $0{,}83 * T$, visto que o *duty cycle* é de 0,83, ou seja, é 0,83 s neste cenário. Enquanto que na fase de transferência é de $(1 - 0{,}83) * T$, logo 0,17 s.

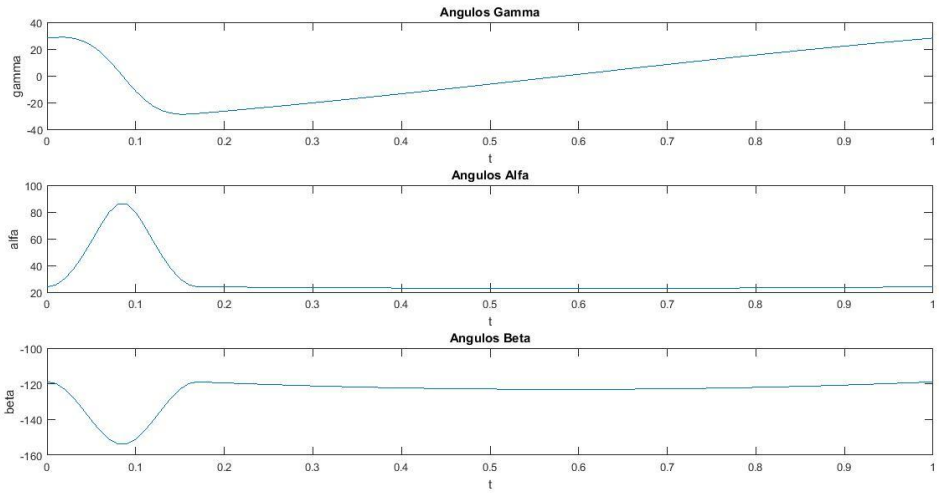


Figura 58 Ângulos γ, α e β ao longo de um período em locomoção ondulatória

Para o último teste, o *duty cycle* foi alterado para 0,75 na simulação da locomoção *ripple*. No caso dos restantes parâmetros tudo se mantém igual. A Figura 59 mostra a ciclóide ao longo de um ciclo nos eixos *y* e *z*.

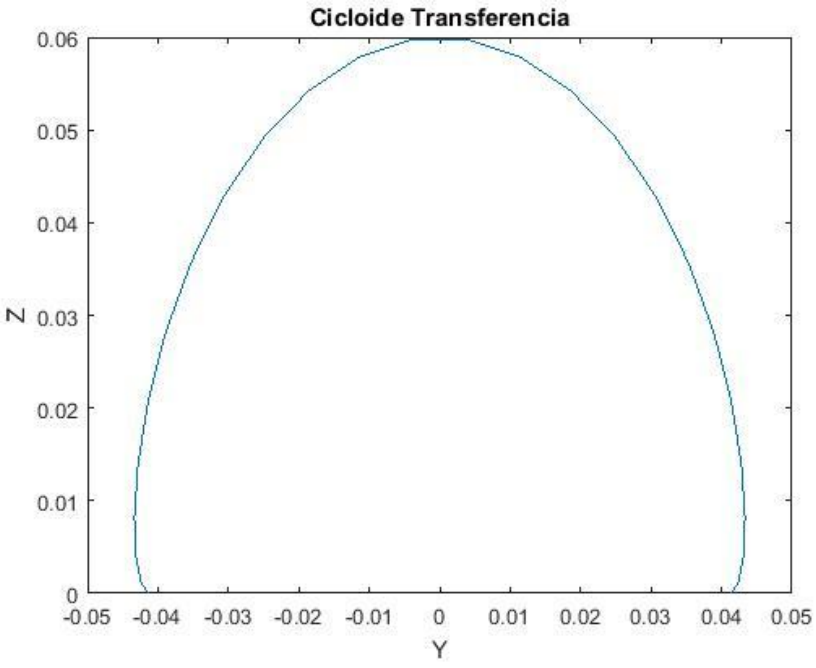


Figura 59 Ciclóide na fase de transferência em locomoção *ripple*

Na Figura 60 estão os valores dos ângulos γ, α e β durante um ciclo, onde é possivel observar que os seus valores variam muito similarmente às locomoções descritas anteriormente. A maior diferença está no tempo de suporte que é de 0,75 s, pois o *duty cycle* é de 0,75. O tempo de transferência é de 0,25 s.

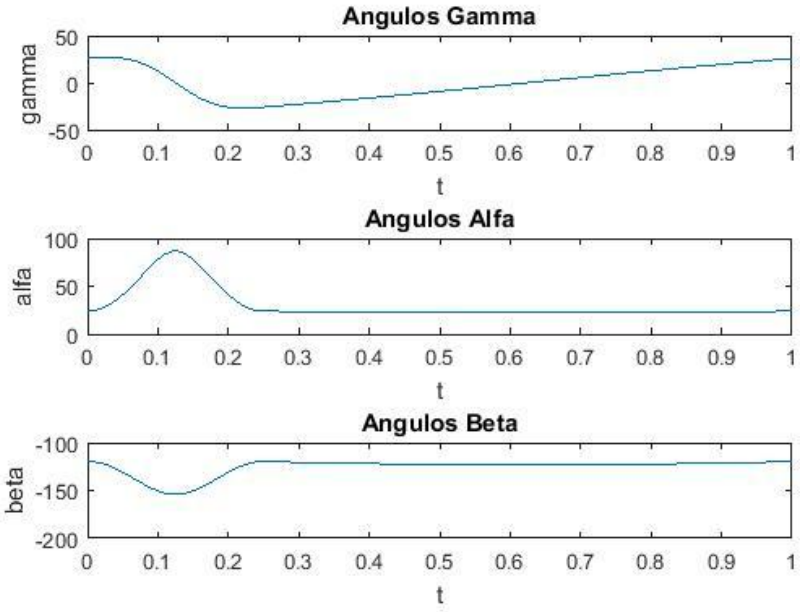


Figura 60 Ângulos γ, α e β ao longo de um período em locomoção *ripple*

Reunindo todos os dados dos três padrões de locomoção, os limites impostos anteriormente apenas refletem-se no ângulo α porque este tem que atingir 90º. No entanto é possível contornar esse problema usando o limite da junta nesse ângulo. No caso de β é preciso existir uma deslocação de *offset* de -90º no membro da tíbia em relação ao fémur, portanto o valor de -121,6º é na realidade uma deslocação de -31,6º, e no caso de -154º é de -64º, pois soma 90º aos valores obtidos.

7.3. DESENVOLVIMENTO DOS PADRÕES DE LOCOMOÇÃO

O desenvolvimento dos padrões de locomoção foi a fase mais exigente do trabalho pois exigiu bastante programação e conhecimento da cinemática assim como alguma noção da dinâmica da locomoção hexápode.

O fluxograma que explica o funcionamento do robô está descrito na Figura 61. O código presente implementado no robô está descrito no anexo C, o qual começa por arrancar com as inicializações do robô onde são armazenados os valores das variáveis globais importantes.

A variável D_X é a distância em *x* que a pata vai ficar da sua origem, F_C é o valor do ponto máximo da ciclóide, L_S é o comprimento da deslocação, o valor *T* é o tempo de período de cada locomoção, coxa, fémur e tíbia são os comprimentos dos membros da pata. A_C é a altura do robô em relação ao solo e *T_amostra* é o valor de cada amostra num período. O vetor *t* contém os valores do período desde 0 até *T* com incrementos de *T_amostra*, *ts* é o vetor com os valores do período em fase de suporte, de 0 até ao valor de $T * \beta$, com incrementos de *T_amostra*. Finalizando o vetor *tt* é semelhante a *ts*, mas este contém os valores do período em fase de transferência, desde 0 até $T * (1 - \beta)$.

Posicao_X1, *Posicao_Y1* e *Posicao_Z1* armazenam os valores das posições do cálculo da ciclóide na fase de transferência, *Posicao_X1_2, Posicao_Y1_2* e *Posicao_Z1_2* para os valores das posições na fase de suporte, *Theta_1*, *Theta_2* e *Theta_3* para os valores dos ângulos das juntas na fase de transferência, *Theta_1_2*, *Theta_2_2* e *Theta_3_2* para os valores dos ângulos na fase de suporte. *Gamma*, *Alfa* e *Beta* armazenam os valores dos ângulos durante um ciclo completo. As restantes *dgamma*, *dalfa* e *dbeta*, guardam os valores das juntas em posição de repouso ou *Home*.

A porta série é configurada para comunicação com o XBee pela porta USB usando uma taxa de 9600 bps. São também configurados os módulos de controlo dos servomotores com os seus endereços, assim como o PWM a ser usado, desde a sua frequência, largura e seu período.

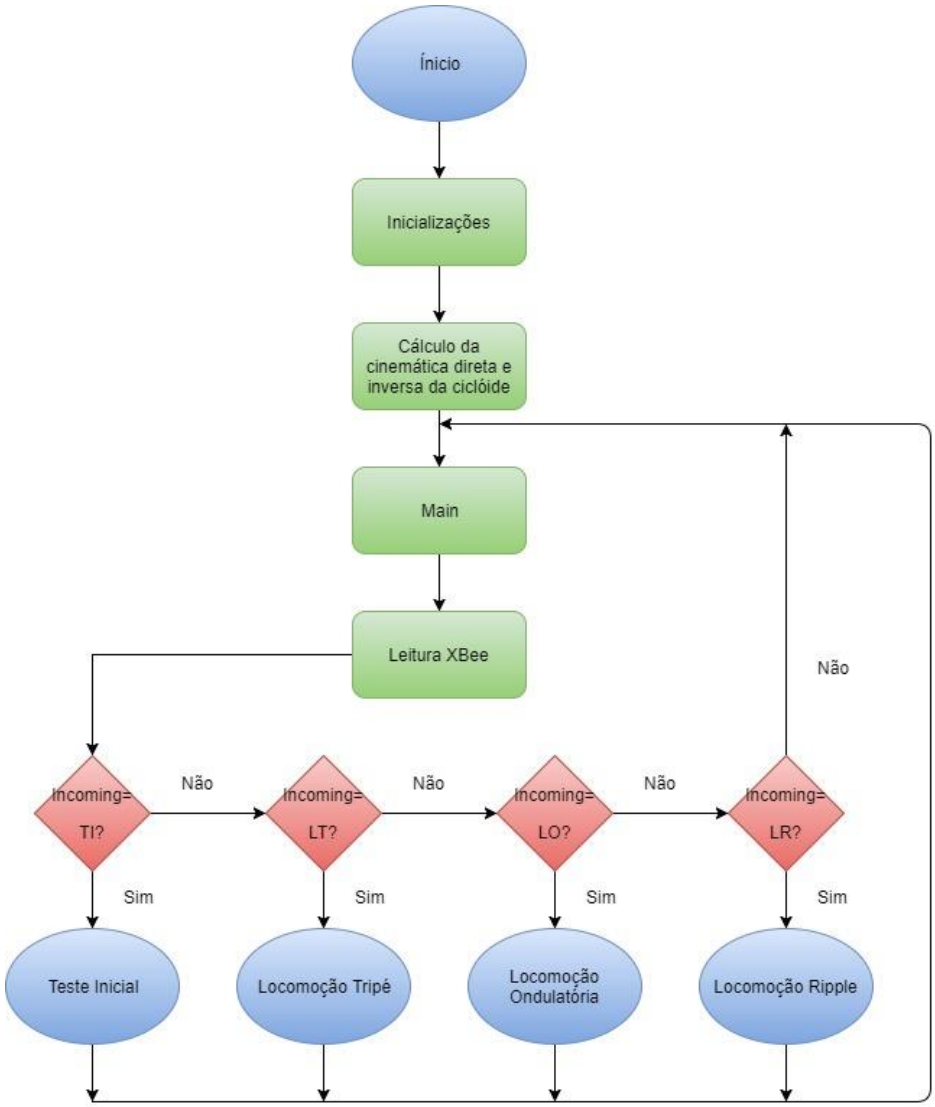


Figura 61 Fluxograma do robô

No bloco do cálculo da cinemática da ciclóide é recebido o parâmetro de β, são então calculados alguns parâmetros já estudados no planeamento de trajetórias como V_F e V_L, assim como são preenchidos os vetores *tt*, tempo da fase de transferência, com incrementos de *T_amostra* e *ts*, tempo da fase de suporte, com incrementos de *T_amostra*. Para o cálculo da cinemática da ciclóide em fase de transferência é usado um ciclo *for* que percorre o comprimento do vetor *tt* e em cada iteração usa as equações da ciclóide para cada eixo e armazena nos vetores anteriormente mencionados. O procedimento é idêntico para a fase de suporte, apenas muda o comprimento do ciclo, percorre o vetor *ts* e usa as equações da ciclóide em fase de suporte. No cálculo dos

ângulos, os ciclos *for* são iguais apenas mudando as equações da cinemática inversa já estudados.

Os valores calculados de posição da fase de transferência e suporte são armazenados nas variáveis já descritas nas inicializações. No final, os ângulos ficam armazenados nos vetores *Gamma*, *Alfa* e *Beta*, no entanto são usadas variáveis auxiliares para armazenar os valores das juntas das três locomoções, isto porque este procedimento é usado três vezes.

Assim que entra na *main* entra também num ciclo *while* e o hexápode fica na posição *Home* explícita na Figura 62. Esta posição é obtida pelo primeiro valor dos vetores *Alfa* para junta α e *Beta* para junta β, e o valor da junta γ é 0º. Neste caso específico os parâmetros usados foram os mesmos da simulação em Matlab.

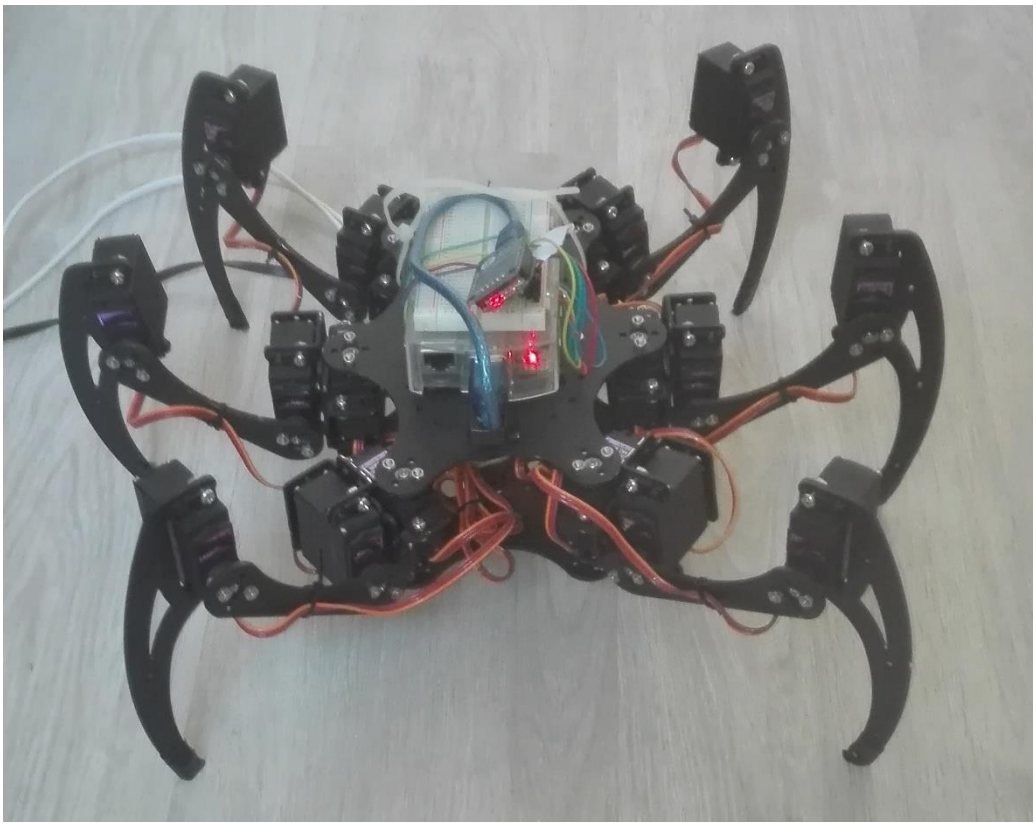


Figura 62 Posição *Home* vista de cima

O robô fica numa posição de descanso com todas a patas apoiadas no solo e todas distribuídas uniformemente, como já demonstrado anteriormente com a posição

ServoMed nas juntas γ. Contudo, as juntas α têm uma deslocação de 23,3º e β de -64º que obriga o robô a se aproximar do solo e a extremidade das patas do centro de massa para dar mais consistência e estabilidade à deslocação. Estes valores são obtidos no cálculo da ciclóide e na obtenção dos ângulos na fase de suporte. Esta posição pode ser verificada na Figura 63.

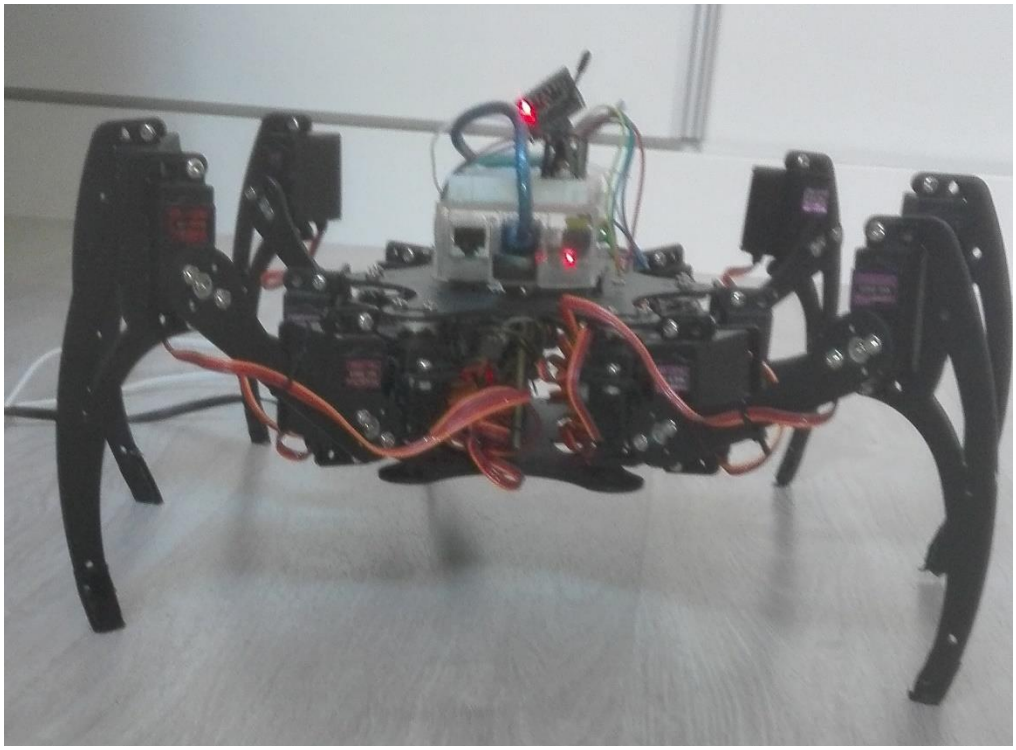


Figura 63 Posição *Home* vista de frente

A leitura do módulo XBee é então executada e armazenada na variável *incoming*, sendo então iniciado o ciclo da locomoção desenvolvida consoante o seu valor.

Se *incoming* for TI o teste inicial inicia, sendo apenas um deslocamento das patas para duas posições já conhecidas, a posição ServoMed e a posição *Home*. No enxerto de código apresentado de seguida é descrito a função que coloca o robô em posição ServoMed e *Home:*

```
def servomed():
     for x in range(0, 9):
          SE.setPWM(x, 0, equacao_reta(0))
          SD.setPWM(x, 0, equacao_reta(0))


def home():
     for x in range(0, 9, 3):
          SE.setPWM(x, 0, equacao_reta(0))
          SE.setPWM(x+1, 0, equacao_reta(Alfa[0]))
          SE.setPWM(x+2, 0, equacao_reta(-Beta[0]-90))


          SD.setPWM(x, 0, equacao_reta(0))
          SD.setPWM(x+1, 0, equacao_reta(-Alfa[0]))
          SD.setPWM(x+2, 0, equacao_reta(Beta[0]+90))
```

O código consiste em ciclos *for* que colocam as posições nas juntas do robô, no caso ServoMed é colocado o valor de 0º equivalente ao ServoMed, no caso *Home* é posto o valor nas juntas da primeira posição do vetor correspondente e 0º em γ. A utilização da função equacao_reta é com o intuito de converter o valor do ângulo em graus para um intervalo entre o valor mínimo e máximo do impulso PWM usado no controlo dos servos.

Algo a reter é a diferença dos sinais dos servos do lado esquerdo para o direito, isto porque na realidade são simétricos em posição, logo o sentido será contrário. No entanto, teoricamente e na explicação das locomoções são usados o mesmo sentido para

ambos os hemisférios. Os valores dos ângulos das juntas são os calculados com os parâmetros iguais aos usados na simulação de Matlab.

Se *incoming* for "LT"então a locomoção tripé inicia, esta já envolve o robô na posição *Home* e aguarda de seguida a deslocação lida pelo XBee, se for um carater "F", que é frente, inicia um período completo da locomoção usando todas as patas.

As patas E1, E3 e D2 entram em fase de transferência fazendo uma deslocação em ciclóide para a frente, simultaneamente as patas D1, D3 e E2 em fase de suporte deslocam-se para trás.

Na segunda metade do movimento para a frente, as patas D1, D3 e E2 em fase de transferência, descrevem a ciclóide para frente, as restantes patas E1, E3 e D2, em fase de suporte, movem-se para trás. Este processo é semelhante ao anterior, apenas muda o conjunto de patas que levantam e as que se movem para trás.

Na Figura 64 é representado graficamente o desenvolver das três juntas ao longo de cinco períodos com duração de 1 s cada. De realçar que estes gráficos sofrem um desfasamento quando são os diferentes conjuntos de patas, portanto o conjunto E1, E3 e D2 está desfasado 0,5**T*, logo 0,5 s, do conjunto D1, D3 e E2.

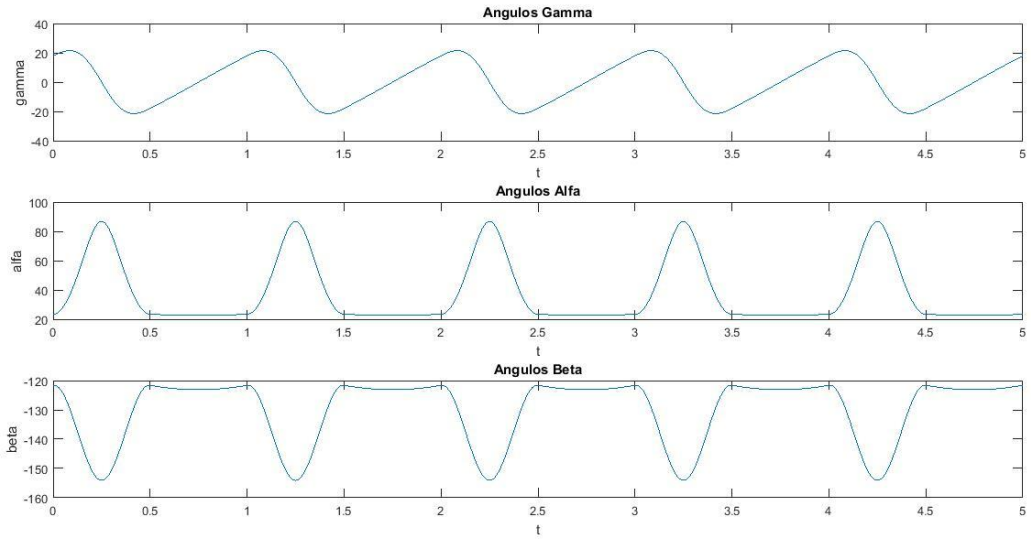


Figura 64 Ângulos γ, α e β ao longo de cinco períodos em locomoção tripé

Recordando que os valores indicados das juntas são apenas específicos para um conjunto de parâmetros dados para o cálculo do planeamento de trajetórias, além que os valores que são percorridos são os contidos nos vetores calculados.

Este procedimento é semelhante aos restantes movimentos, desde virar para esquerda ou direita ou mesmo para trás. Apenas difere o sentido das patas no caso de andar para trás, e no caso de virar para a esquerda ou direita as patas andam em sentidos contrários, os do hemisfério esquerdo das do hemisfério direito. Isto é possível, usando uma variável que multiplica pelo valor da junta γ, que pode ser positivo ou negativo conforme o sentido que se pretende a ciclóide. A sequência fica mais explícita no diagrama da Figura 65.

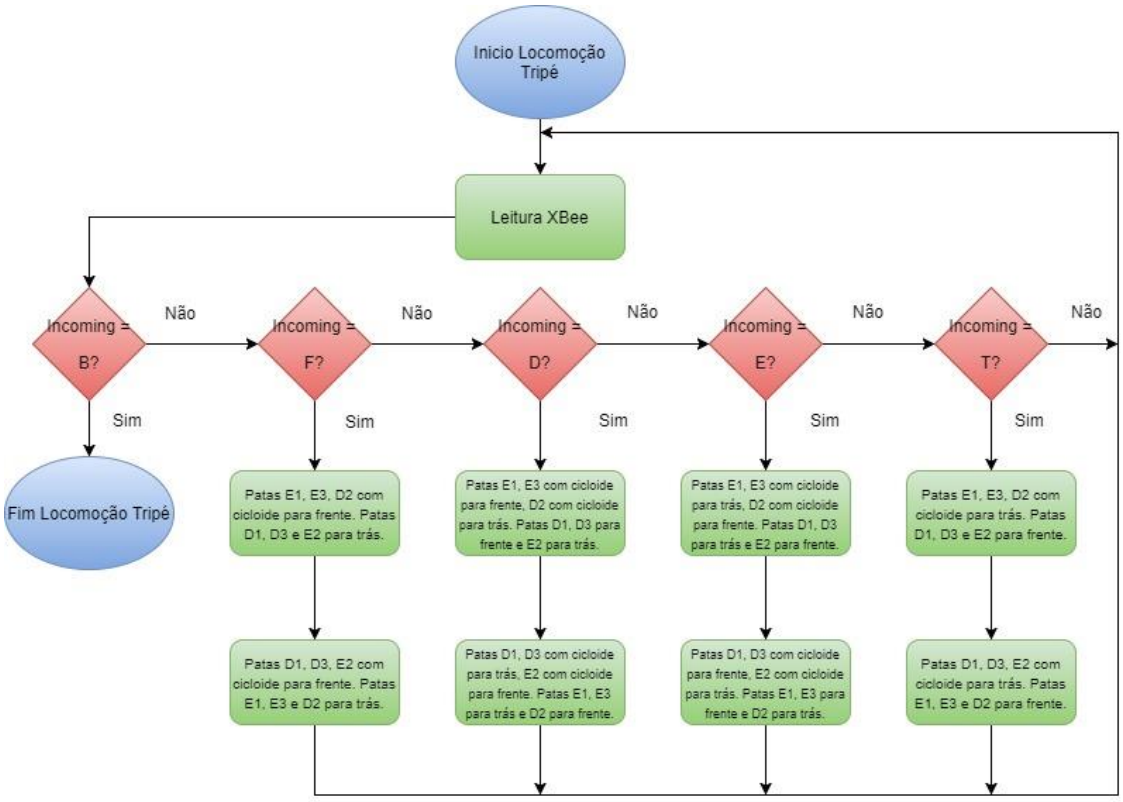


Figura 65 Diagrama da locomoção tripé

Se incoming tiver o valor igual a "LO" a locomoção ondulatória começa, a qual envolve a leitura pelo XBee tal como o caso anterior. Se for um carater "F", é feita uma ciclóide e um avanço pata a pata, começando pela pata D3, que em fase de transferência vai deslocar a junta γ para a frente, a junta α faz um movimento que levanta a pata toda e a

junta β levanta mais a extremidade da pata, ficando no topo da ciclóide. As restantes patas fazem movimento em γ contínuo para trás.

A segunda metade do movimento é depois na junta γ até ao limite de seu deslocamento, junta α para o ponto de repouso e junta β para o valor de contacto com o solo, pousando assim a pata. As restantes patas em fase de suporte vão-se movendo para trás continuamente até iniciarem as suas respetivas fases de transferência. Depois é a vez de D2 fazer o mesmo movimento, de seguida a D1, E3, E2 e por fim E1 e assim acaba um ciclo completo da deslocação para a frente.

Na Figura 66 é representado graficamente o desenvolver das três juntas ao longo de cinco períodos. É possível perceber que a fase de transferência é bastante reduzida, sendo de 0,17**T*, equivalente a 0,17 s e a fase de suporte muito longa de 0,83**T*, neste caso 0,83 s. Nesta locomoção as patas descrevem a ciclóide rapidamente, e lentamente retornam para o ponto onde reiniciam novamente a sua fase de transferência. Neste padrão as patas estão desfasadas entre cada uma delas o equivalente a 0,17**T*.

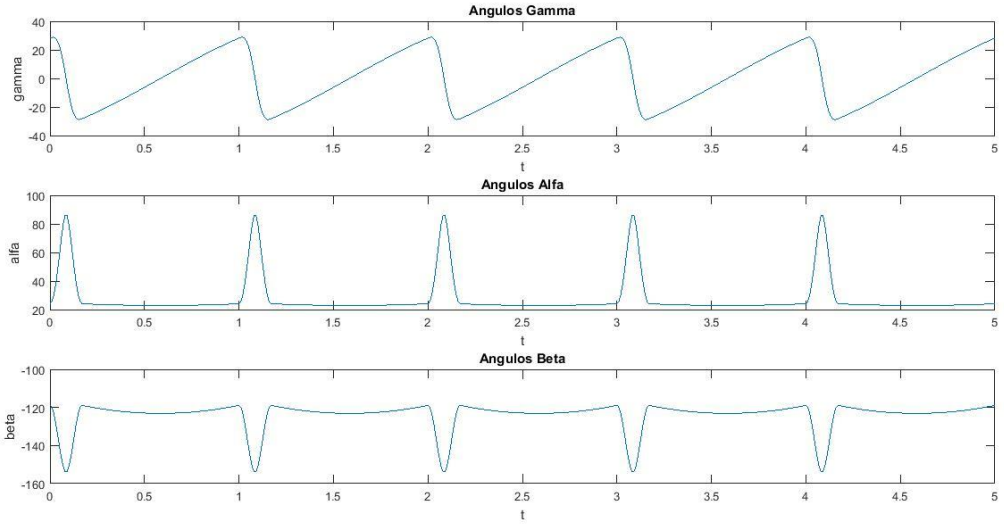


Figura 66 Ângulos γ, α e β ao longo de cinco períodos em locomoção ondulatória

O procedimento descrito é semelhante aos restantes movimentos, desde virar para esquerda ou direita ou mesmo para trás. O que difere na deslocação das patas em ciclóide é que num hemisfério a junta γ tem o sentido oposto do outro hemisfério. Na Figura 67 visualiza-se o diagrama completo.

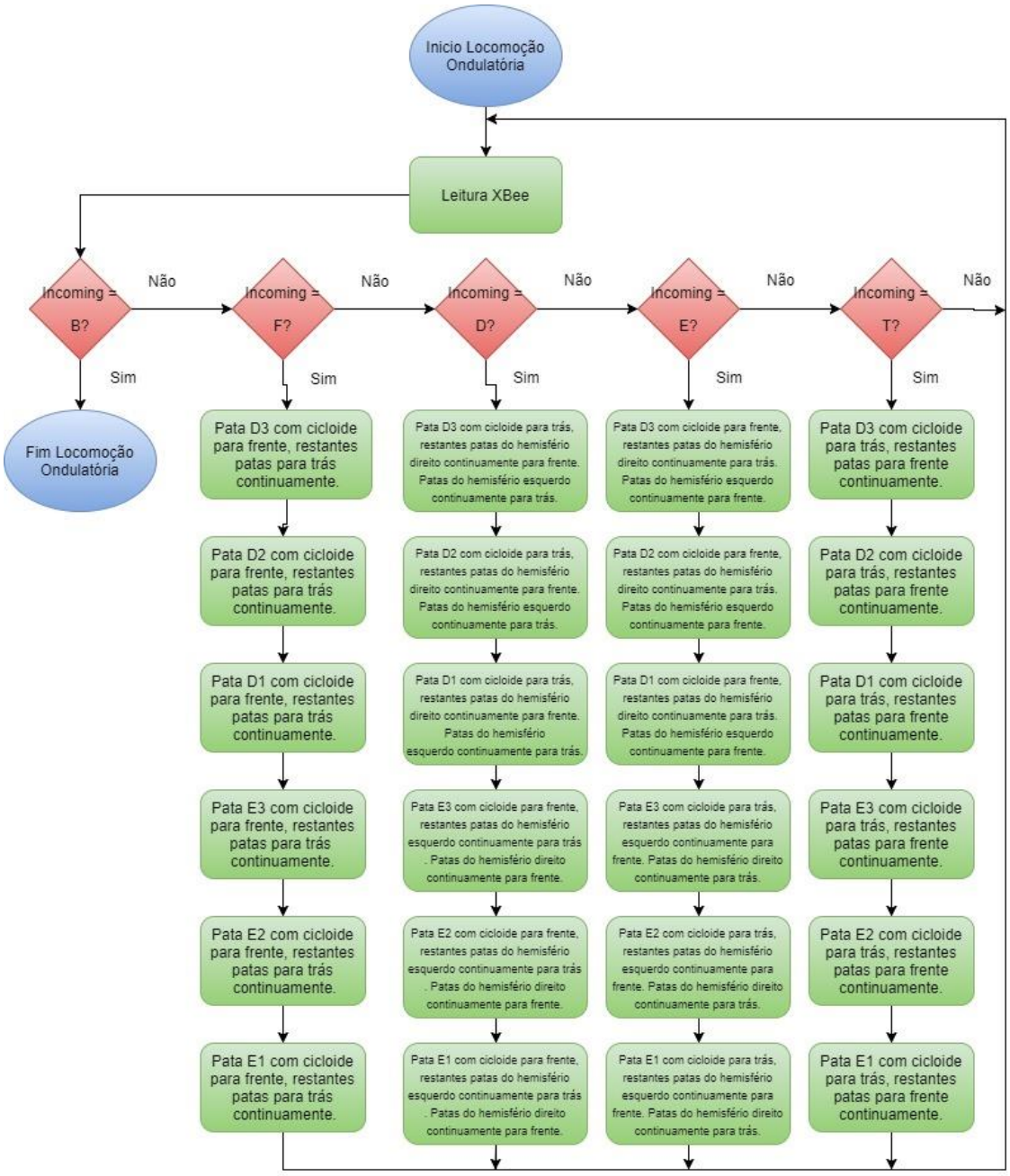


Figura 67 Diagrama da locomoção ondulatória

Com *incoming* de valor igual a "LR" a locomoção *ripple* começa. Esta foi um pouco semelhante à desenvolvida na ondulatória, a diferença é que tem uma sobreposição de movimentos, realizando a deslocação de duas patas em duas fases da locomoção. Tal como as anteriores esta envolve a leitura pelo XBee, se o carater lido for "F", é feita uma ciclóide nas patas começando no par E1 e D3. Estas em fase de transferência vão

deslocar as juntas γ para a frente, as juntas α fazem um movimento que levantam as patas e as juntas β levantam as extremidades delas, ficando ambas as patas no ar. Prosseguindo é a vez de a junta γ continuar o movimento para a frente, para terminar as juntas α e as juntas β retornam aos valores de repouso, atingindo o solo, enquanto que as restantes patas fazem movimento em fase de suporte até γ atingir o ponto de reiniciar a fase de transferência.

Depois é a pata D2, de seguida a E3 juntamente com D1, e por fim E2 e assim acaba um ciclo completo da deslocação para a frente.

Na Figura 68 é representado este padrão de locomoção ao longo de cinco períodos, onde é possível perceber que a fase de transferência é igual a 0,25**T*, equivalente a 0,25 s e a fase de suporte de 0,75**T*, neste caso 0,75 s. Nesta locomoção as patas descrevem a ciclóide em duas fases com duas patas, as outras duas fases é só uma pata a movimentar-se. Neste padrão as patas nas quatro fases estão desfasadas entre cada uma delas o equivalente a 0,25**T*.

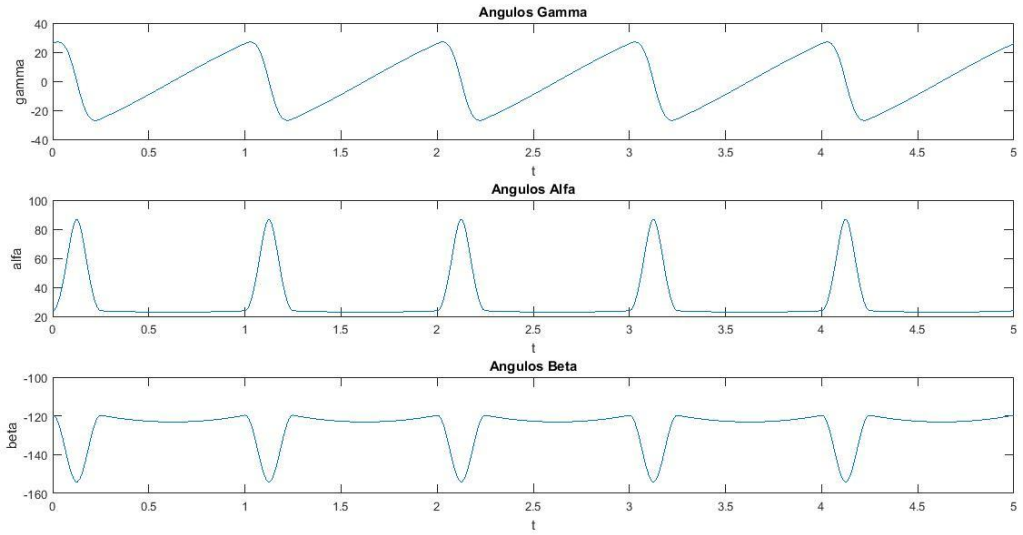


Figura 68 Ângulos γ, α e β ao longo de cinco períodos em locomoção *ripple*

Este procedimento é semelhante aos restantes movimentos, desde virar para esquerda ou direita ou mesmo para trás, apenas difere a deslocação das patas na junta γ para o sentido oposto. Na Figura 69 está retratado o diagrama da locomoção *ripple*.

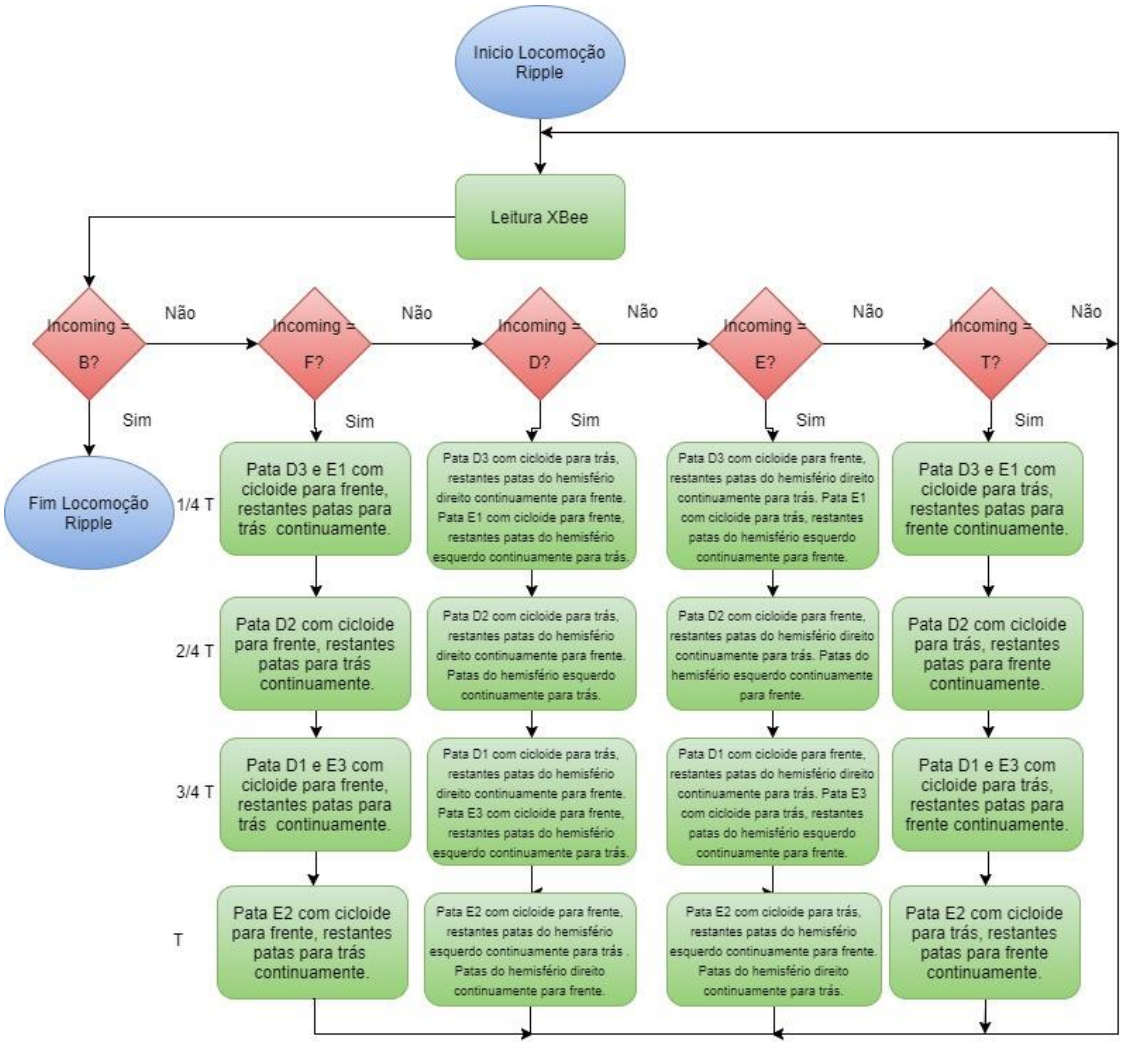


Figura 69 Diagrama da locomoção *ripple*

7.4. TESTES E RESULTADOS

Inicialmente, antes de colocar o robô hexápode em contacto com o solo, os testes das locomoções foram efetuados sobre um suporte que manteve o robô suspenso. Após verificação do funcionamento de todos os padrões, o robô foi colocado no solo.

Os testes realizados foram o de verificação visual de cada padrão, ilustrado nas Figuras 70, 71 e 72, o teste de velocidade, estabilidade e desvio no hexápode.

Iniciando o teste da verificação visual de cada padrão foram capturadas as imagens e o vídeo de cada padrão.

No caso da locomoção tripé, foi capturado na Figura 70 as duas fases da locomoção para a frente, sendo a primeira descrita por (a) e a segunda por (b).

(a) (b)

Figura 70 Locomoção tripé

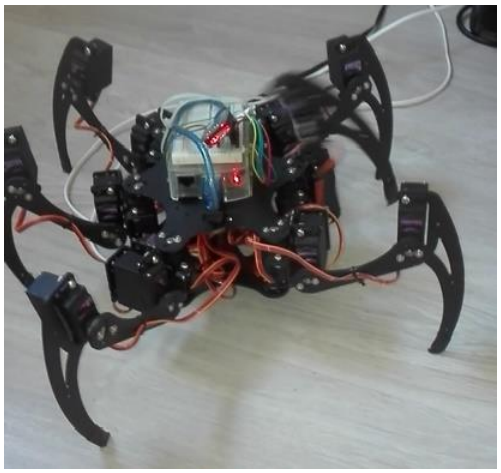
A locomoção ondulatória foi capturada também para as 6 fases da deslocação para a frente. A Figura 71 apresenta por ordem esse mesmo movimento designados por (a)-(f).

(a)

(b)

(c)

(d)

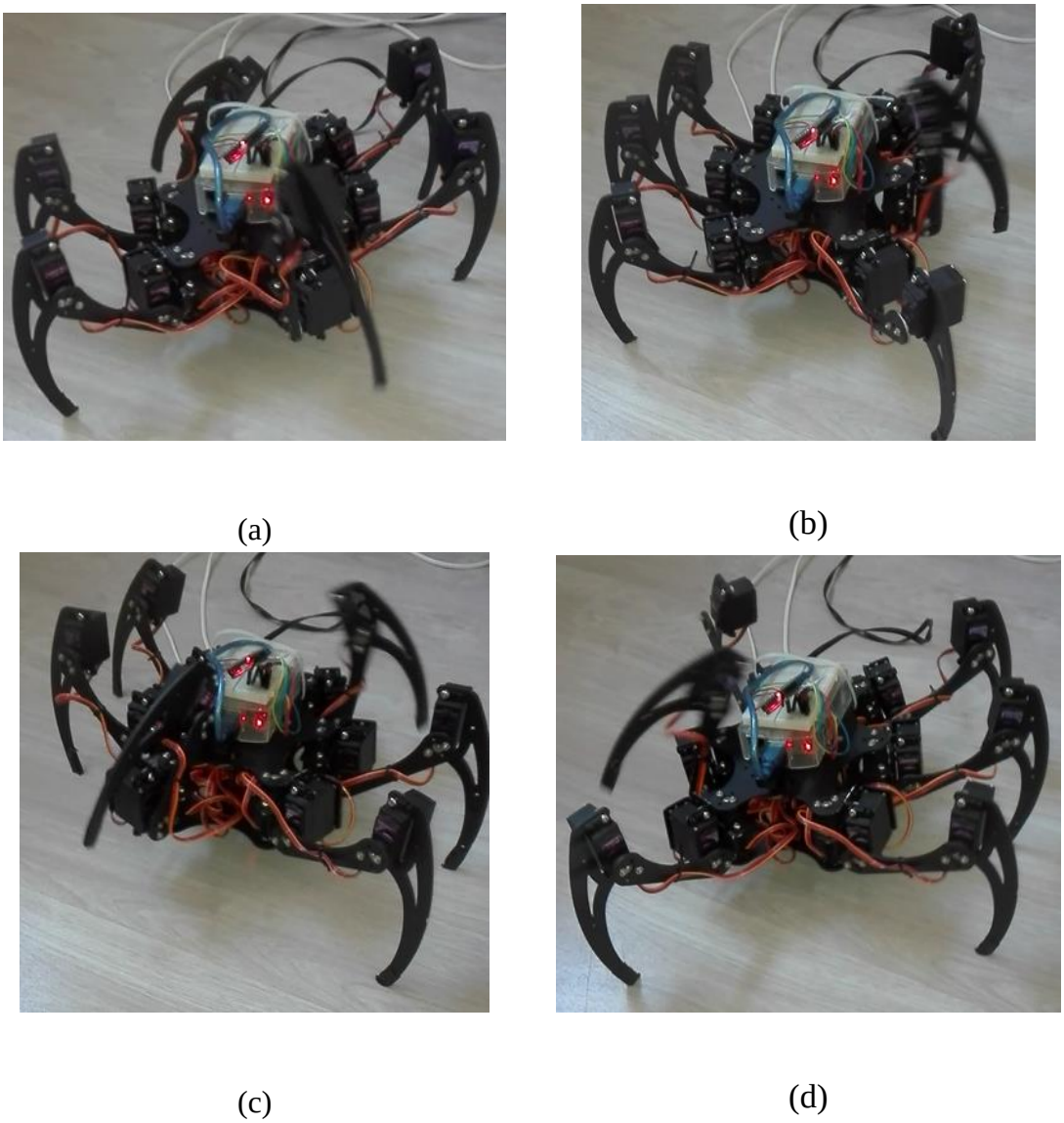


(e)

(f)

Figura 71 Locomoção ondulatório

Por fim a locomoção *ripple* foi capturada em 4 fases, sendo descritas na Figura 72 de (a)-(d).



(a) (b)

(c) (d)

Figura 72 Locomoção *ripple*

O teste de velocidade foi realizado com três configurações de parâmetros diferentes, as três locomoções possuem todas a mesma configuração durante o teste, ou seja, o deslocamento ou o período será sempre igual para todas. Foi traçado uma reta de 1,5 m e foi cronometrado o tempo que o hexápode demorou a percorrer esse espaço.

Na primeira configuração utilizaram-se os mesmos parâmetros da simulação do Matlab e da demonstração das deslocações das juntas nas locomoções, com os seguintes parâmetros:

F_C = 0,06 [m];

L_S = 0,110 [m]

D_X = 0,085 [m];

T = 1 [s];

Coxa = 0,030 [m];

Fémur = 0,085 [m];

Tíbia = 0,130 [m];

A_C = 0,095 [m];

T_amostra = 0,01 [s].

Daqui obtiveram-se os resultados descritos na Tabela 9.

Tabela 9 Resultados da primeira configuração

Locomoção	**Tempo [m:s:ms]**	**Desvio [m]**	**Desvio [º]**
Locomoção Tripé	00:36:44	0,30 m	11,6º
Locomoção Ondulatório	00:39:82	0,31 m	12º
Locomoção *Ripple*	00:37:94	0,28 m	10,8º

Daqui surge a primeira disparidade do simulado para o que foi medido, a passada sendo de 0,110 m por período, o robô deveria percorrer 1,5 m em aproximadamente 14 ciclos, pois $\frac{1{,}5\ m}{0{,}110\ m} = 13{,}63$.

Contudo a passada do robô que foi medida teve como valor 0,092 m por período, o que perfaz 1,5 m em 16 ciclos.

Esses mesmos 16 ciclos demorariam teoricamente e em simulação 16 s, devido a *T* ser de 1 s, porém cronometrado demorou por volta dos 35 s em cada padrão.

Na segunda configuração foram adotados os seguintes parâmetros:

F_C = 0,06 [m];

L_S = 0,110 [m]

D_X = 0,085 [m];

T = 2 [s];

Coxa = 0,030 [m];

Fémur = 0,085 [m];

Tíbia = 0,130 [m];

A_C = 0,095 [m];

T_amostra = 0,01 [s].

Em que se alterou o período para 2 s. Os resultados estão colocados na Tabela 10.

Tabela 10 Resultados da segunda configuração

Locomoção	**Tempo [m:s:ms]**	**Desvio [m]**	**Desvio [º]**
Locomoção Tripé	01:09:41	0,44 m	17,3º
Locomoção Ondulatório	01:16:55	0,45 m	17,7º
Locomoção *Ripple*	01:12:95	0,40 m	15,6º

Mais uma vez observa-se a disparidade dos valores da simulação para o medido, contudo os valores são aproximadamente o dobro dos valores obtidos para o período de 1 s, o que mantém a consistência nos testes reais.

Na terceira configuração foram adotados os seguintes parâmetros:

F_C = 0,06 [m];

L_S = 0,055 [m]

D_X = 0,085 [m];

T = 1 [s];

Coxa = 0,030 [m];

Fémur = 0,085 [m];

Tíbia = 0,130 [m];

A_C = 0,095 [m];

T_amostra = 0,01 [s].

Neste caso alterou-se o período T para 1 s e o L_S para 0,055 m, e os resultados obtidos foram os descritos na Tabela 11.

Tabela 11 Resultados da terceira configuração

Locomoção	**Tempo [m:s:ms]**	**Desvio [m]**	**Desvio [º]**
Locomoção Tripé	01:12:22	0,48 m	18,9º
Locomoção Ondulatório	01:21:66	0,38 m	14,8º
Locomoção *Ripple*	01:17:84	0,31 m	12,0º

É possível concluir que a mais rápida é a tripé e a mais lenta a ondulatória, indo de encontro ao estudado na teoria em que o *duty cycle* do tripé era o que tinha o valor mais baixo de fases de transferência, o que o torna ligeiramente mais rápido enquanto o da ondulatória era o que tinha valor mais elevado de fases de transferência, o que o torna mais lento.

Outro ponto a ressalvar é o tempo de duração cronometrado para o tempo de duração de simulação. O tempo cronometrado foi sempre consideravelmente superior ao tempo de duração de simulação, o qual se deve ao tempo da comunicação série entre o Raspberry pi e os servocontroladores, assim como na comunicação com cada servo. Como o número de amostras que são usadas são elevadas isso leva o servo a ser mais lento a cumprir cada posição, além de ter de superar o erro interno da posição de referência para a atual e a resposta de binário de cada servo. Outra causa de atraso são os tempos de processamento intermédios que levam a converter os graus em impulsos PWM.

A locomoção tripé tem o desvio mais acentuado, isto porque desloca mais patas ao mesmo tempo, o que agrava o desvio que é maioritariamente para a esquerda, isto devido parcialmente a um servo do hemisfério esquerdo, na junta γ, que tinha menos amplitude de movimento. Em situações normais sem defeitos nos servos e sem folgas mecânicas o desvio deveria ser bastante reduzido.

Em termos de estabilidade é o inverso, sendo o ondulatório mais estável pois tem sempre 5 patas em fase de suporte e tem 1 em fase de transferência, de seguida é o *ripple* porque tem sempre 4 patas de apoio em todas as fases da locomoção e por fim a tripé que tem apenas 3 patas.

8. CONCLUSÕES

Esta dissertação foi focada na modelação e desenvolvimento de um robô hexápode que fosse possível controlar por um comando sem fios. Os padrões foram desenvolvidos com recurso à teoria das locomoções conhecidas e com o planeamento de trajetórias convencional da ciclóide. Existiu a necessidade de implementar a parte de cinemática para ter noção da posição da pata no espaço, a dinâmica foi mencionada teoricamente, no entanto não foi usada. Existiam outras formas de abordar este trabalho com uso de outras ferramentas, contudo com os recursos disponíveis foi feito um trabalho bastante satisfatório.

Com o estudo da locomoção foi permitido descobrir assertivamente qual o melhor tipo de locomoção para um hexápode, além do tipo de estabilidade que advém de cada uma. Outros pontos a reter é que o uso deste tipo de robôs tem vantagens muito significativas, comparativamente a veículos com rodas devido à grande adaptabilidade a terrenos desnivelados, assim como capacidade de evasão de obstáculos e outros problemas no solo.

Dos testes efetuados foi possível esclarecer que a locomoção em tripé é a mais rápida, mas a menos estável, a ondulatória ou metacronal é a mais lenta, mas a mais estável, a *ripple* ou

tetrápode é a intermédia em temos de velocidade e estabilidade, portanto a velocidade é inversamente proporcional à estabilidade.

O *hardware* usado não foi o melhor em termos de precisão e fiabilidade, mas permitiu resultados satisfatórios, isto porque os servos usados assim como os respetivos controladores de servos interferiram na qualidade e precisão dos movimentos, o que causou os desvios nos testes e acabou por dificultar um pouco o desenvolvimento final.

Para melhorias futuras as locomoções podiam ser conciliadas com câmara para deteção de obstáculos assim como acelerómetro para desnível do robô e até mesmo GPS para ter uma noção do espaço e deslocações. Outra melhoria importante passaria por usar um RTC no Raspberry e o uso de interrupções para tornar a leitura do XBee em tempo real e a possibilidade de alterar movimentos mais rapidamente.

Referências Documentais

[1] http://hackerboards.com/mantis-hexapod-robot-embedded-linux-computer/ , “Inside Mantis: a 2-ton hexapod robot with a Linux brain”. Visitado em 15/09/2016.

[2] https://www.robotcenter.co.uk/products/topio , “Robot Center Topio”. Visitado em 16/09/2016.

[3] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, Rob Playter and the BigDog Team, “BigDog, the Rough-Terrain Quaduped Robot”.

[4] http://www.pcmag.com/article2/0,2817,1742094,00.asp, “Sony AIBO ERS-7M2”. Visitado em 18/09/2016.

[5] https://www.fzi.de/en/research/projekt-details/lauron/, “LAURON V” Visitado em 18/09/2016.

[6] https://www.researchgate.net/publication/263741408_Reactive_Posture_Behaviors_for_Stable_Legged_Locomotion_over_Steep_Inclines_and_Large_Obstacles, “Reactive Posture Behaviors for Stable Legged Locomotion over Steep Inclines and Large Obstacles”. Visitado em 18/09/2016.

[7] H.W. Ma, L.Q. Wang, D.L. Chen, X.W. Hao and H.W. Luo , “Design of A Crab-like Octopod Robot”.

[8] http://www.mondospider.com/, “Mondo Spider Photos”. Visitado em 20/09/2016.

[9] Xochitl Yamile Sandoval-Castro1,*, Mario Garcia-Murillo1, Luis Alberto Perez-Resendiz2 and Eduardo Castillo-Castañeda1, “Kinematics of Hex-Piderix - A Six-Legged Robot - Using Screw Theory”.

[10] https://oscarliang.com/inverse-kinematics-and-trigonometry-basics/, “Inverse Kinematics Tutorial Introduction”. Visitado em 04/10/2016.

[11] John Xiao, and Siegwart&Nourbakhsh , “Robot Locomotion and Kinematics”.

[12] M. F. S. Silva,“Sistemas robóticos de locomoção multipernas,”Ph.D. dissertation, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2005.

[13] Luiz Oliveira, “Modelação, Simulação e Implementação de Padrões de Locomoção para Robôs Hexápodes”, Porto, Portugal, 2016.

[14] Franco Tedeschi and Giuseppe Carbone , “Design Issues for Hexapod Walking Robots”, 2014.

[15] Ricardo Campos, Vitor Matos, Cristina Santos, “Hexapod Locomotion: a Nonlinear Dynamical Systems Approach”.

[16] Frank A. DeWitt IV , “An inverse kinematics approach to hexapod design and control”.

[17] http://www.robotshop.com/en/lynxmotion-phoenix-3dof-hexapod---black-no-servos---electronics.html, “Lynxmotion Phoenix 3DOF Hexapod”. Visitado em 02/11/2016.

[18] http://www.trossenrobotics.com/phantomx-ax-hexapod-mk1.aspx, “PhantomX AX Hexapod”. Visitado em 03/11/2016.

[19] www.lynxmotion.com/c-92-ah3-r.aspx, “AH3-R”. Visitado em 08/11/2016.

[20] https://www.raspberrypi.org/products/model-b/, “Raspberry Pi 1 Model B”. Visitado em 10/11/2016.

[21] http://www.electroschematics.com/7963/arduino-mega-2560-pinout/, “Arduino Mega 2560”. Visitado em 10/11/2016.

[22] http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f1-series/stm32f103.html?querycriteria=productId=LN1565, “STM32F103”. Visitado em 10/11/2016.

[23] http://www.servodatabase.com/servo/towerpro/mg996r, “TowerPro MG996R Servo”. Visitado em 14/11/2016.

[24] http://www.servodatabase.com/servo/hitec/hs-645mg, “Hitec HS-645MG - High Torque MG Servo”. Visitado em 14/11/2016.

[25] http://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx, “Dynamixel AX-12A Robot Actuator”. Visitado em 14/11/2016.

[26] https://www.adafruit.com/product/815, “Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685”. Visitado em 14/11/2016.

[27] https://www.elechouse.com/elechouse/index.php?main_page=product_info&cPath=100_146&products_id=1883, “32-Channel Servo Controller Board V2”. Visitado em 14/11/2016.

[28] https://www.pololu.com/product/1356, “Mini Maestro 24-Channel USB Servo Controller”. Visitado em 14/11/2016.

[29] http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01, “nRF24L01”.

[30] http://www.ti.com/product/CC1101, “CC1101” Visitado em 17/11/2016.

[31] https://www.sparkfun.com/products/8742, “XBee Pro”. Visitado em 17/11/2016.

[32] http://electronics.stackexchange.com/questions/70155/raspberry-pi-driving-servo-with-pwm, “PWM Raspberry Pi”. Visitado em 11/12/2016.

[33] https://docs.python.org/2/tutorial/, “The Python Tutorial”. Visitado em 12/11/2016.

[34] https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/ , “Adafruit 16-Channel PWM/Servo HAT for Raspberry Pi”. Visitado em 20/11/2016.

[35] https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu ,” Exploring XBees and XCTU ”. Visitado em 02/12/2016.

[36] http://www.inhaos.com/product_info.php?products_id=78 ,“LCD-2000-9225 2.0inch TFT”. Visitado em 02/12/2016.

[37] https://thearduinostore.wordpress.com/products/joystick-shield-v1-a/ , “Joystick Shield V1.A”. Visitado em 05/12/2016.

[38] Marek Zák, “Design, Construction and Control of Hexapod Walking Robot”, 2015.

[39] Ing. R.Woering, “Simulating the first steps of a walking hexapod robot”, 2010.

[40] Claudio Urrea, Luis Valenzuela and John Kern , “Design, Simulation, and Control of a Hexapod Robot in Simscape Multibody”.

[41] Zhiying Wang , Alberto Rovetta , “Locomotion Analysis of Hexapod Robot”, 2010.

[42] Simon Cherlet, Pieter Maelegheer , “Developing an autonomous hexapod robot for environmental exploration”, 2014.

Anexo A. Cálculo de Cinemática em Matlab

```
      function [gamma,alfa,beta,Posicao_X,Posicao_Y,Posicao_Z] = ...
Planeamento_Trajetorias(Fc,Ls,T,Dc,coxa,femur,tibia,Ac,T_amostra)
    close all
    t = 0:T_amostra:T;
    TS = (Dc*T); % Fase de suporte
    ts = 0:T_amostra:TS;
    TT = (1-Dc)*T; % Fase de transferencia
    tt = 0:T_amostra:TT;
    VF = Ls/T; % Velocidade do corpo frontal
    VL = (1/(1-Dc))*VF; % Velocidade de uma pata
    Gamma = [];
    Alfa = [];
    Beta = [];
    Gamma_2 = [];
    Alfa_2 = [];
    Beta_2 = [];

    % Calculo da Cicloide na fase de transferencia
    Posicao_X1(1,1:(length(tt))) = 0.085
    Posicao_Y1 = VL*(tt-sin(2*pi*tt/TT)*TT/(2*pi)) - VF.*tt - Ls*Dc/2;
    Posicao_Z1 = Fc/2*(1-cos(2*pi*tt/TT));
    %Inverte horizontalmente o vetor posicao X
    Posicao_X1 = wrev(Posicao_X1);
    %Inverte horizontalmente o vetor posicao Y
    Posicao_Y1 = wrev(Posicao_Y1);
    %Inverte horizontalmente o vetor posicao Z
    Posicao_Z1 = wrev(Posicao_Z1);
    figure
    plot(Posicao_Y1,Posicao_Z1), xlabel('Y'), ylabel('Z'),
title('Cicloide Transferencia')

    % Calculo angulos juntas na fase transferencia
    for i = 1:1:length(tt),
        Theta_1 = atan2(Posicao_Y1(1,i),Posicao_X1(1,i));
        Theta_3 = -acos((Posicao_X1(1,i)^2+Posicao_Y1(1,i)^2+ ...
        (Posicao_Z1(1,i)-Ac)^2+coxa^2-femur^2-tibia^2- ...

2*coxa*sqrt(Posicao_X1(1,i)^2+Posicao_Y1(1,i)^2))/(2*femur*tibia));
        Theta_2 = atan2((Posicao_Z1(1,i)-Ac), ...
        sqrt(Posicao_X1(1,i)^2+Posicao_Y1(1,i)^2) - coxa)- ...
        atan2((tibia*sin(Theta_3)),(femur+tibia*(Posicao_X1(1,i)^2+ ...
        Posicao_Y1(1,i)^2 + (Posicao_Z1(1,i)-Ac)^2+coxa^2-femur^2 - ...
        tibia^2-2*coxa*sqrt(Posicao_X1(1,i)^2+Posicao_Y1(1,i)^2)) ...
        /(2*femur*tibia)));
        Gamma = [Gamma 180/pi*Theta_1];
        Alfa = [Alfa 180/pi*Theta_2];
        Beta = [Beta 180/pi*Theta_3];
    end
    X = ((coxa + femur*cos(Theta_2) + ...
    tibia*cos(Theta_2 + Theta_3))*cos(Theta_1));

    % Calculo da cicloide na fase de suporte
    Posicao_X1_2(1,1:length(ts)) = X;
```

```
    Posicao_Y1_2 = -VF.*ts + Ls*Dc/2;
    Posicao_Z1_2(1,1:length(ts)) = Ac;
    %Inverte horizontalmente o vetor posicao X
    Posicao_X1_2 = wrev(Posicao_X1_2);
    %Inverte horizontalmente o vetor posicao Z
    Posicao_Z1_2 = wrev(Posicao_Z1_2);

    % Calculo angulos juntas na fase suporte
    for i = 1:1:length(ts),
        Theta_1 = atan2(Posicao_Y1_2(1,i),Posicao_X1_2(1,i));
        Theta_3 = acos(((sqrt(Posicao_X1_2(1,i)^2+ ...
        Posicao_Y1_2(1,i)^2) - coxa)^2 + Posicao_Z1_2(1,i)^2 - ...
        femur^2 - tibia^2)/(2*femur*tibia));
        Theta_2 = atan2(Posicao_Z1_2(1,i), ...
        sqrt(Posicao_X1_2(1,i)^2+Posicao_Y1_2(1,i)^2)-coxa) - ...
        atan2((tibia*sin(Theta_3)),(femur+tibia*cos(Theta_3)));
        Gamma_2 = [Gamma_2 180/pi*Theta_1];
        Alfa_2 = [Alfa_2 180/pi*Theta_2];
        Beta_2 = [Beta_2 180/pi*Theta_3];
    end
    %Salva o valor de theta1 em um unico vetor
    gamma = horzcat(Gamma,-Gamma_2(1,2:end))
    %Salva o valor de theta2 em um unico vetor
    alfa = horzcat(Alfa,-Alfa_2(1,2:end))
    %Salva o valor de theta3 em um unico vetor
    beta = horzcat(Beta,-Beta_2(1,2:end))
    Theta1 = gamma;
    Theta2 = alfa;
    Theta3 = beta;
    %Salva o valor da posicao X em um unico vetor
    Posicao_X = horzcat(Posicao_X1,Posicao_X1_2(1,2:end));
    %Salva o valor da posicao Y em um unico vetor
    Posicao_Y = horzcat(Posicao_Z1,Posicao_Z1_2(1,2:end));
    %Salva o valor da posicao Z em um unico vetor
    Posicao_Z = horzcat(Posicao_Y1,Posicao_Y1_2(1,2:end));
    posicao_X = Posicao_X;
    posicao_Y = Posicao_Y;
    posicao_Z = Posicao_Z;
    %Ajusta o numero de ciclos
    figure
    plot(t,gamma), xlabel('t'), ylabel('gamma'), title('Angulos Gamma')
    figure
    plot(t,alfa), xlabel('t'), ylabel('alfa'), title('Angulos Alfa')
    figure
    plot(t,beta), xlabel('t'), ylabel('beta'), title('Angulos Beta')
    save ('gammalr','gamma', '-ascii')
    save ('alfalr','alfa', '-ascii')
    save ('betalr','beta', '-ascii')
end
```

Anexo B. Código Arduino do Comando

```
#include <LCD_2000_7775.h>

// definição dos pinos
#define cs   40
#define wr   39
#define rs   38
#define rst  41

#define D0 37
#define D1 36
#define D2 35
#define D3 34
#define D4 33
#define D5 32
#define D6 31
#define D7 30

// cria uma instancia da biblioteca
LCD_2000_7775 TFTscreen(cs,wr,rs,rst);

// variaveis globais
 String inputString = "";
 boolean stringComplete = false;
 char sensorPrintout[4];
 int i=0;
 int inic=0;
 int out=0;
```

```
 int C = A0;
 int B = A1;
 int joyX = A2;
 int joyY = A3;
 int state = 0;
 int button=0;
 int anim=0;
 int flag_exit=0;

 // inicializações
void setup() {
   Serial.begin(9600);
   inputString.reserve(200);
   pinMode(C, INPUT);
   pinMode(B, INPUT);
   pinMode(joyX, INPUT);
   pinMode(joyY, INPUT);
   Serial.begin(9600);
    TFTscreen.setdatapin(D0,D1,D2,D3,D4,D5,D6,D7);
    TFTscreen.begin();
    TFTscreen.background(0, 0, 0);
    TFTscreen.stroke(255,255,255);
    TFTscreen.setTextSize(1);
}

void loop() {
  mainMenu();
}

void botoes(){
  state=0;
  if (analogRead(joyY)<450) {
    //baixo
    state = 1;
```

```
   button=1;
  } else if(analogRead(joyY)>550) {
   //cima
   state = 2;
   button=1;
  } else if (analogRead(joyX)<250) {
    //esquerda
   state = 3;
   button=1;
  } else if(analogRead(joyX)>850) {
  //direita
   state = 4;
   button=1;
  } else if(analogRead(C)<300){
  //C
   state = 5;
   button=1;
  } else if(analogRead(B)<300){
   //B
   state = 6;
   button=1;
  }

}


void mainMenu() {
  TFTscreen.setTextSize(1);
  out=0;
  botoes();
  if(button=1){
     if (state == 1) {
        // baixo?
        clearPoint();
```

```
        i = i + 20;
      }
      if (state == 2) {
        // cima?
        clearPoint();
        i = i - 20;  }

  if (i <= 0) {
   i = 0;
  }
  else if (i > 60) {
   i = 60;
  }
   if (state == 5) {
        // C?
       TFTscreen.fillScreen(RGB(0,0,0));
       selectMenu(i);
          }
  displayMenu(i);
}

}


// imprime menu principal
void displayMenu(int x) {
   TFTscreen.stroke(255,255,255);
   TFTscreen.text("-> ",0,x);
   TFTscreen.text("1 - Teste Inicial",10,0);
   TFTscreen.text("2 - Locomocao Tripe", 10, 20);
   TFTscreen.text("3 - Locomocao Ondulatorio", 10, 40);
   TFTscreen.text("4 - Locomocao Ripple", 10, 60);
   TFTscreen.stroke(0,255,0);
   TFTscreen.text(". Botao C Para selecionar", 10, 80);
```

```
  TFTscreen.stroke(255,0,0);
  TFTscreen.text(". Botao B Para cancelar", 10, 100);
  TFTscreen.stroke(0,0,255);
  TFTscreen.setTextSize(2);
  TFTscreen.text("ISEP", 0, 140);
  TFTscreen.text("TEDI", 0, 160);
  TFTscreen.setTextSize(1);
  TFTscreen.text("1100394 - Andre Trigo", 0, 180);
}

// limpa a seta
 void clearPoint() {
  TFTscreen.stroke(0,0,0);
  TFTscreen.text("-> ",0,i);

}



// imprime cada opcao
void selectMenu(int x) {
   TFTscreen.fillScreen(RGB(0,0,0));
   TFTscreen.setTextSize(1);
   out=0;
   TFTscreen.stroke(255,0,0);
   TFTscreen.text(". Botao B Para cancelar", 10, 200);

  switch (x) {

     case 0:
       TFTscreen.setTextSize(1.5);
       TFTscreen.stroke(0,0,255);
       TFTscreen.text("Teste Inicial",10,0);
       Serial.write("TI");
```

```
  while(out==0){
    teste_inicial();
    if(flag_exit==1){
    TFTscreen.fillScreen(RGB(0,0,0));
    out=1;
    flag_exit=0;
    }
  }
  break;

case 20:
  TFTscreen.setTextSize(1.5);
  TFTscreen.stroke(0,0,255);
  TFTscreen.text("Locomocao Tripe",10,0);
  Serial.write("LT");
  while(out==0){
    locomocao();
    printhexapodtripod();
    if(flag_exit==1){
    TFTscreen.fillScreen(RGB(0,0,0));
    out=1;
    flag_exit=0;
    }
  }
  break;

 case 40:
    TFTscreen.stroke(0,0,255);
    TFTscreen.text("Locomocao Ondulatorio",10,0);
    Serial.write("LO");
    while(out==0){
      locomocao();
      printhexapodwave();
      if(flag_exit==1){
```

```
            TFTscreen.fillScreen(RGB(0,0,0));
            out=1;
            flag_exit=0;
           }
        }
        break;

      case 60:
        TFTscreen.stroke(0,0,255);
        TFTscreen.text("Locomocao Ripple",10,0);
        Serial.write("LR");
        while(out==0){
          locomocao();
          printhexapodwave();
          if(flag_exit==1){
              TFTscreen.fillScreen(RGB(0,0,0));
              out=1;
              flag_exit=0;
              }
        }
        break;
}
}

// imprime a animação do tripe
void printhexapodtripod(){
  if(anim==0){
    TFTscreen.stroke(0,0,0);
    TFTscreen.line(85,150,105,145);
    TFTscreen.line(95,130,115,135);
    TFTscreen.line(85,110,105,105);
    TFTscreen.line(65,150,45,155);
    TFTscreen.line(55,130,35,125);
    TFTscreen.line(65,110,45,115);
```

```
    TFTscreen.drawCircle(75,130,20,RGB(255,255,255));
    TFTscreen.stroke(200,255,255);
    TFTscreen.line(85,150,105,155);
    TFTscreen.line(95,130,115,125);
    TFTscreen.line(85,110,105,115);
    TFTscreen.line(65,150,45,145);
    TFTscreen.line(55,130,35,135);
    TFTscreen.line(65,110,45,105);
    anim=1;
    }
  else if(anim==1){
    TFTscreen.stroke(0,0,0);
    TFTscreen.line(85,150,105,155);
    TFTscreen.line(95,130,115,125);
    TFTscreen.line(85,110,105,115);
    TFTscreen.line(65,150,45,145);
    TFTscreen.line(55,130,35,135);
    TFTscreen.line(65,110,45,105);
    TFTscreen.drawCircle(75,130,20,RGB(255,255,255));
    TFTscreen.stroke(200,255,255);
    TFTscreen.line(85,150,105,145);
    TFTscreen.line(95,130,115,135);
    TFTscreen.line(85,110,105,105);
    TFTscreen.line(65,150,45,155);
    TFTscreen.line(55,130,35,125);
    TFTscreen.line(65,110,45,115);
    anim=0;
    }
}

// imprime a animacao do ondulatorio
void printhexapodwave(){
  if(anim==0){
    TFTscreen.stroke(0,0,0);
```

```
    TFTscreen.line(85,150,105,145);
    TFTscreen.line(95,130,115,125);
    TFTscreen.line(85,110,105,105);
    TFTscreen.line(65,150,45,145);
    TFTscreen.line(55,130,35,125);
    TFTscreen.line(65,110,45,105);
    TFTscreen.drawCircle(75,130,20,RGB(255,255,255));
    TFTscreen.stroke(200,255,255);
    TFTscreen.line(85,150,105,155);
    TFTscreen.line(95,130,115,135);
    TFTscreen.line(85,110,105,115);
    TFTscreen.line(65,150,45,155);
    TFTscreen.line(55,130,35,135);
    TFTscreen.line(65,110,45,115);
    anim=1;
}
  else if(anim==1){
    TFTscreen.stroke(0,0,0);
    TFTscreen.line(85,150,105,155);
    TFTscreen.line(95,130,115,135);
    TFTscreen.line(85,110,105,115);
    TFTscreen.line(65,150,45,155);
    TFTscreen.line(55,130,35,135);
    TFTscreen.line(65,110,45,115);
    TFTscreen.drawCircle(75,130,20,RGB(255,255,255));
    TFTscreen.stroke(200,255,255);
    TFTscreen.line(85,150,105,145);
    TFTscreen.line(95,130,115,125);
    TFTscreen.line(85,110,105,105);
    TFTscreen.line(65,150,45,145);
    TFTscreen.line(55,130,35,125);
    TFTscreen.line(65,110,45,105);
    anim=0;
```

```
}




}

// imprime o teste inicial
void teste_inicial(){
  TFTscreen.setTextSize(1);
  TFTscreen.stroke(255,255,255);
  botoes();
  if(button=1){
      if (state == 6) {//If B
      Serial.write("B");
      flag_exit=1;
            }
      }
  TFTscreen.fillRoundRect(20,45,130,25,10,RGB(0,0,255));
  TFTscreen.text("A testar....",30,55);



}

// imprime os sentidos
void locomocao(){
  TFTscreen.setTextSize(1);
  TFTscreen.stroke(255,255,255);
  botoes();
  TFTscreen.fillRoundRect(20,45,130,25,10,RGB(0,0,255));
  if(button=1){
  if (state == 1) {
    Serial.write("T");//If tras
    TFTscreen.text("Tras",30,55);
```

```
 }
 if (state == 2) {//If frente
    Serial.write("F");
    TFTscreen.text("Frente",30,55);}
 if (state == 3) {//If esquerda
    Serial.write("E");
    TFTscreen.text("Esquerda",30,55);
   }
 if (state == 4) {//If direita
    Serial.write("D");
    TFTscreen.text("Direita",30,55);
   }
 if (state == 5) {//If C
    Serial.write("C");
    TFTscreen.text("Botao C",30,55);
   }
 if (state == 6) {//If B
   Serial.write("B");
   flag_exit=1;
   TFTscreen.text("Botao B",30,55);
   }
 delay(100);
 TFTscreen.stroke(0,0,255);
 TFTscreen.text("Tras",30,55);
 TFTscreen.text("Frente",30,55);
 TFTscreen.text("Esquerda",30,55);
 TFTscreen.text("Direita",30,55);
 TFTscreen.text("Botao C",30,55);
 TFTscreen.text("Botao B",30,55);
 }
}
```

Anexo C. Código Python do Raspberry Pi

```
#!/usr/bin/python
# Inicializacoes

from Adafruit_PWM_Servo_Driver import PWM
import time
import serial
import math
import numpy as np
import pylab as pl
import matplotlib.pyplot as plt

ser = serial.Serial('/dev/ttyUSB0', 9600)
SD = PWM(0x40)
SE = PWM(0x41)

servoMin = 400
servoMax = 1700
servoMed = 950
flag = 0
dgamma = 0
dalfa = 0
dbeta = 0
Dx = 0.075
Fc = 0.075
Ls = 0.110
T = 1
```

```
coxa = 0.030
femur = 0.085
tibia = 0.130
Ac = 0.095
T_amostra = 0.01
t = []
ts = []
tt = []
ttlt = []
ttlo = []
ttlr = []
Posicao_X1 = []
Posicao_Y1 = []
Posicao_Z1 = []
Posicao_X1_2 = []
Posicao_Y1_2 = []
Posicao_Z1_2 = []
Theta_1 = []
Theta_2 = []
Theta_3 = []
Theta_1_2 = []
Theta_2_2 = []
Theta_3_2 = []
Gamma = []
Alfa = []
Beta = []
Gamma_lt = []
Alfa_lt = []
Beta_lt = []
Gamma_lo = []
Alfa_lo = []
Beta_lo = []
Gamma_lr = []
Alfa_lr = []
```

```
Beta_lr = []

# Parametrizacao do impulso PWM
def setServoPulse(channel, pulse):
    pulseLength = 1000000                   # 1,000,000 us por segundo
    pulseLength /= 50                       # 50 Hz
    print "%d us per period" % pulseLength
    pulseLength /= 4096                     # 12 bits resolucao
    print "%d us per bit" % pulseLength
    pulse *= 1000
    pulse /= pulseLength
    SD.setPWM(channel, 0, pulse)
    SE.setPWM(channel, 0, pulse)
    SD.setPWMFreq(50)
    SE.setPWMFreq(50)

# Parametrizacao da equacao da reta
def equacao_reta(angulo):
    res = int(angulo*6.66)+950
    return res

# Calculo da cinematica direta e inversa da cicloide
def cicloide(Dc):
    t = np.arange(0,T,T_amostra)
    TS = Dc*T
    ts = np.arange(0,TS,T_amostra)
    TT = (1-Dc)*T
    tt = np.arange(0,TT,T_amostra)
    VF = Ls/T
    VL = (1/(1-Dc))*VF
    del Posicao_X1[:]
    del Posicao_Y1[:]
    del Posicao_Z1[:]
    del Posicao_X1_2[:]
```

```
    del Posicao_Y1_2[:]
    del Posicao_Z1_2[:]
    del Theta_1[:]
    del Theta_2[:]
    del Theta_3[:]
    del Theta_1_2[:]
    del Theta_2_2[:]
    del Theta_3_2[:]
    del Gamma[:]
    del Alfa[:]
    del Beta[:]

    # Calculo da cicloide em transferencia
    for i in np.arange(0, len(tt), 1):
            Posicao_X1.append(Dx)
            Posicao_Y1.append(VL*(tt[i]-
math.sin(2*math.pi*tt[i]/TT)*TT/(2*math.pi)) - VF*tt[i] - Ls*Dc/2)
            Posicao_Z1.append(Fc/2*(1-math.cos(2*math.pi*tt[i]/TT)))

    # Calculo dos angulos das juntas em transferencia
    for i in np.arange(0, len(tt), 1):
            Theta_1.append(math.atan2(Posicao_Y1[i],Posicao_X1[i]))
            Theta_3.append(-
math.acos((math.pow(Posicao_X1[i],2)+math.pow(Posicao_Y1[i],2)+math.pow((Posic
ao_Z1[i]-Ac),2)+math.pow(coxa,2)-math.pow(femur,2)-math.pow(tibia,2)-
2*coxa*math.sqrt(math.pow(Posicao_X1[i],2)+math.pow(Posicao_Y1[i],2)))/(2*femur
*tibia)))
            Theta_2.append(math.atan2((Posicao_Z1[i]-
Ac),math.sqrt(math.pow(Posicao_X1[i],2)+math.pow(Posicao_Y1[i],2)) - coxa) -
math.atan2((tibia*math.sin(Theta_3[i])),(femur+tibia*(math.pow(Posicao_X1[i],2) +
math.pow(Posicao_Y1[i],2) + math.pow(Posicao_Z1[i]-Ac,2)+math.pow(coxa,2)-
math.pow(femur,2) - math.pow(tibia,2)-
2*coxa*math.sqrt(math.pow(Posicao_X1[i],2)+math.pow(Posicao_Y1[i],2))) /
(2*femur*tibia))))
```

```
            Gamma.append(180/math.pi*Theta_1[i])
            Alfa.append(180/math.pi*Theta_2[i])
            Beta.append(180/math.pi*Theta_3[i])

    # Calculo da cicloide em suporte
    for i in np.arange(0, len(ts), 1):
            X = ((coxa + femur*math.cos(Theta_2[0]) + tibia*math.cos(Theta_2[0] +
Theta_3[0]))*math.cos(Theta_1[0]))

            Posicao_X1_2.append(X)
            Posicao_Y1_2.append(-VF*ts[i] + Ls*Dc/2)
            Posicao_Z1_2.append(Ac)

    # Calculo dos angulos das juntas em suporte
    for i in np.arange(0, len(ts), 1):
            Theta_1_2.append(math.atan2(Posicao_Y1_2[i],Posicao_X1_2[i]))
            Theta_3_2.append(-
math.acos((math.pow(math.sqrt(math.pow(Posicao_X1_2[i],2)+
math.pow(Posicao_Y1_2[i],2)) - coxa,2) + math.pow(Posicao_Z1_2[i],2) -
math.pow(femur,2) - math.pow(tibia,2))/(2*femur*tibia)))
            Theta_2_2.append(-math.atan2(Posicao_Z1_2[i],
math.sqrt(math.pow(Posicao_X1_2[i],2)+math.pow(Posicao_Y1_2[i],2))-coxa) -
math.atan2((tibia*math.sin(Theta_3_2[i])),(femur+tibia*math.cos(Theta_3_2[i]))))

            Gamma.append(180/math.pi*Theta_1_2[i])
            Alfa.append(180/math.pi*Theta_2_2[i])
            Beta.append(180/math.pi*Theta_3_2[i])

# Funcao que coloca robo em posicao ServoMed (posicao 0)
def servomed():
    for x in range(0, 9):
            SE.setPWM(x, 0, equacao_reta(0))
            SD.setPWM(x, 0, equacao_reta(0))
```

```
# Funcao que coloca robo em posicao home
def home():
    for x in range(0, 9, 3):
            SE.setPWM(x, 0, equacao_reta(0))
            SE.setPWM(x+1, 0, equacao_reta(Alfa[0]))
            SE.setPWM(x+2, 0, equacao_reta(-Beta[0]-90))

            SD.setPWM(x, 0, equacao_reta(0))
            SD.setPWM(x+1, 0, equacao_reta(-Alfa[0]))
            SD.setPWM(x+2, 0, equacao_reta(Beta[0]+90))

# Funcao que ajusta o vetor para a locomocao tripe
def ajustelt(i):
    if i>1:
            i=i-2
    return i

# Funcao executa a locomocao tripe
def LT(sentidoE, sentidoD):
    for i in range(0, 2, 1):
            for x in range(0, len(ttlt), 1):
                    SE.setPWM(0,                                    0,
equacao_reta(sentidoE*Gamma_lt[ajustelt(i)*len(ttlt)+x]))
                    SE.setPWM(1, 0, equacao_reta(Alfa_lt[ajustelt(i)*len(ttlt)+x]))
                    SE.setPWM(2, 0, equacao_reta(-Beta_lt[ajustelt(i)*len(ttlt)+x]-90))
                    SE.setPWM(3,                                    0,
equacao_reta(sentidoE*Gamma_lt[ajustelt(i+1)*len(ttlt)+x]))
                    SE.setPWM(4, 0, equacao_reta(Alfa_lt[ajustelt(i+1)*len(ttlt)+x]))
                    SE.setPWM(5,   0,   equacao_reta(-Beta_lt[ajustelt(i+1)*len(ttlt)+x]-
90))
                    SE.setPWM(6,                                    0,
equacao_reta(sentidoE*Gamma_lt[ajustelt(i)*len(ttlt)+x]))
                    SE.setPWM(7, 0, equacao_reta(Alfa_lt[ajustelt(i)*len(ttlt)+x]))
```

```
                SE.setPWM(8, 0, equacao_reta(-Beta_lt[ajustelt(i)*len(ttlt)+x]-90))
                SD.setPWM(0, 0, equacao_reta(sentidoD*-
Gamma_lt[ajustelt(i+1)*len(ttlt)+x]))
                SD.setPWM(1, 0, equacao_reta(-Alfa_lt[ajustelt(i+1)*len(ttlt)+x]))
                SD.setPWM(2, 0,
equacao_reta(Beta_lt[ajustelt(i+1)*len(ttlt)+x]+90))
                SD.setPWM(3, 0, equacao_reta(sentidoD*-
Gamma_lt[ajustelt(i)*len(ttlt)+x]))
                SD.setPWM(4, 0, equacao_reta(-Alfa_lt[ajustelt(i)*len(ttlt)+x]))
                SD.setPWM(5, 0, equacao_reta(Beta_lt[ajustelt(i)*len(ttlt)+x]+90))
                SD.setPWM(6, 0, equacao_reta(sentidoD*-
Gamma_lt[ajustelt(i+1)*len(ttlt)+x]))
                SD.setPWM(7, 0, equacao_reta(-Alfa_lt[ajustelt(i+1)*len(ttlt)+x]))
                SD.setPWM(8, 0,
equacao_reta(Beta_lt[ajustelt(i+1)*len(ttlt)+x]+90))

# Funcao que ajusta o vetor para a locomocao ondulatoria
def ajustelo(i):
    if i>5:
        i=i-6
    return i

# Funcao executa a locomocao ondulatoria
def LO(sentidoE, sentidoD):
    for i in range(0, 6, 1):
        for x in range(0, len(ttlo), 1):

                SE.setPWM(0, 0,
equacao_reta(sentidoE*Gamma_lo[ajustelo(i+3)*(len(ttlo)-1)+x]))
                SE.setPWM(1, 0, equacao_reta(Alfa_lo[ajustelo(i+3)*(len(ttlo)-
1)+x]))
                SE.setPWM(2, 0, equacao_reta(-Beta_lo[ajustelo(i+3)*(len(ttlo)-
1)+x]-90))
```

```
                SE.setPWM(3, 0, equacao_reta(sentidoE*Gamma_lo[ajustelo(i+2)*(len(ttlo)-1)+x]))
                SE.setPWM(4, 0, equacao_reta(Alfa_lo[ajustelo(i+2)*(len(ttlo)-1)+x]))
                SE.setPWM(5, 0, equacao_reta(-Beta_lo[ajustelo(i+2)*(len(ttlo)-1)+x]-90))
                SE.setPWM(6, 0, equacao_reta(sentidoE*Gamma_lo[ajustelo(i+1)*(len(ttlo)-1)+x]))
                SE.setPWM(7, 0, equacao_reta(Alfa_lo[ajustelo(i+1)*(len(ttlo)-1)+x]))
                SE.setPWM(8, 0, equacao_reta(-Beta_lo[ajustelo(i+1)*(len(ttlo)-1)+x]-90))
                SD.setPWM(0, 0, equacao_reta(sentidoD*-Gamma_lo[ajustelo(i)*(len(ttlo)-1)+x]))
                SD.setPWM(1, 0, equacao_reta(-Alfa_lo[ajustelo(i)*(len(ttlo)-1)+x]))
                SD.setPWM(2, 0, equacao_reta(Beta_lo[ajustelo(i)*(len(ttlo)-1)+x]+90))
                SD.setPWM(3, 0, equacao_reta(sentidoD*-Gamma_lo[ajustelo(i+5)*(len(ttlo)-1)+x]))
                SD.setPWM(4, 0, equacao_reta(-Alfa_lo[ajustelo(i+5)*(len(ttlo)-1)+x]))
                SD.setPWM(5, 0, equacao_reta(Beta_lo[ajustelo(i+5)*(len(ttlo)-1)+x]+90))
                SD.setPWM(6, 0, equacao_reta(sentidoD*-Gamma_lo[ajustelo(i+4)*(len(ttlo)-1)+x]))
                SD.setPWM(7, 0, equacao_reta(-Alfa_lo[ajustelo(i+4)*(len(ttlo)-1)+x]))
                SD.setPWM(8, 0, equacao_reta(Beta_lo[ajustelo(i+4)*(len(ttlo)-1)+x]+90))

# Funcao executa a locomocao ripple
def ajustelr(i):
    if i>3:
```

```
            i=i-4
    return i

# Funcao executa a locomocao ripple
def LR(sentidoE, sentidoD):
    for i in range(0, 4, 1):
            for x in range(0, len(ttlr), 1):
                    SE.setPWM(0, 0, equacao_reta(sentidoE*Gamma_lr[ajustelr(i+2)*len(ttlr)+x]))
                    SE.setPWM(1, 0, equacao_reta(Alfa_lr[ajustelr(i+2)*len(ttlr)+x]))
                    SE.setPWM(2, 0, equacao_reta(-Beta_lr[ajustelr(i+2)*len(ttlr)+x]-90))
                    SE.setPWM(3, 0, equacao_reta(sentidoE*Gamma_lr[ajustelr(i+3)*len(ttlr)+x]))
                    SE.setPWM(4, 0, equacao_reta(Alfa_lr[ajustelr(i+3)*len(ttlr)+x]))
                    SE.setPWM(5, 0, equacao_reta(-Beta_lr[ajustelr(i+3)*len(ttlr)+x]-90))
                    SE.setPWM(6, 0, equacao_reta(sentidoE*Gamma_lr[ajustelr(i)*len(ttlr)+x]))
                    SE.setPWM(7, 0, equacao_reta(Alfa_lr[ajustelr(i)*len(ttlr)+x]))
                    SE.setPWM(8, 0, equacao_reta(-Beta_lr[ajustelr(i)*len(ttlr)+x]-90))
                    SD.setPWM(0, 0, equacao_reta(sentidoD*-Gamma_lr[ajustelr(i)*len(ttlr)+x]))
                    SD.setPWM(1, 0, equacao_reta(-Alfa_lr[ajustelr(i)*len(ttlr)+x]))
                    SD.setPWM(2, 0, equacao_reta(Beta_lr[ajustelr(i)*len(ttlr)+x]+90))
                    SD.setPWM(3, 0, equacao_reta(sentidoD*-Gamma_lr[ajustelr(i+1)*len(ttlr)+x]))
                    SD.setPWM(4, 0, equacao_reta(-Alfa_lr[ajustelr(i+1)*len(ttlr)+x]))
                    SD.setPWM(5, 0, equacao_reta(Beta_lr[ajustelr(i+1)*len(ttlr)+x]+90))
                    SD.setPWM(6, 0, equacao_reta(sentidoD*-Gamma_lr[ajustelr(i+2)*len(ttlr)+x]))
                    SD.setPWM(7, 0, equacao_reta(-Alfa_lr[ajustelr(i+2)*len(ttlr)+x]))
```

```
                SD.setPWM(8,                                    0,
equacao_reta(Beta_lr[ajustelr(i+2)*len(ttlr)+x]+90))

# Ciclo principal
while (True):
    cicloide(0.5)
    ttlt = np.arange(0,(1-0.5)*T,T_amostra)
    Gamma_lt=Gamma
    Alfa_lt=Alfa
    Beta_lt=Beta
    cicloide(0.833)
    t = np.arange(0,T,T_amostra)
    ttlo = np.arange(0,(1-0.833)*T,T_amostra)
    Gamma_lo=Gamma
    Alfa_lo=Alfa
    Beta_lo=Beta
    cicloide(0.75)
    ttlr = np.arange(0,(1-0.75)*T,T_amostra)
    Gamma_lr=Gamma
    Alfa_lr=Alfa
    Beta_lr=Beta
    while (True):
        home()
        incoming = ser.read()
        if (incoming == "T"):
            flag = 1
        if (incoming == "L"):
            flag = 3
        incoming = ser.read()
        if (incoming == "I"):
            flag = 2
        if (incoming == "T"):
            flag = 4
        if (incoming == "O"):
```

```
        flag = 6
if (incoming == "R"):
        flag = 8
print 'Flag %s' % flag

# Teste inicial
while (flag == 2)       :
        print 'Recebido %s' % incoming
        servomed()
        time.sleep(1)
        home()
        time.sleep(1)
        incoming = ser.read()
        if (incoming == 'B'):
                flag = 0

#------------------------------------------------------------
# Locomocao Tripe
while (flag == 4)       :
        incoming = ser.read()
        print 'Recebido %s' % incoming
        if (incoming == 'B'):
                flag = 0
        if incoming == 'F':
                LT(1, 1)
        elif incoming == 'D':
                LT(1, -1)
        elif incoming == 'E':
                LT(-1, 1)
        if incoming == 'T':
                LT(-1, -1)

#------------------------------------------------------------
# Locomocao Ondulatoria
```

```
while (flag == 6)        :
        incoming = ser.read()
        print 'Recebido %s' % incoming
        if (incoming == 'B'):
                flag = 0
        if (incoming == 'F'):
                LO(1, 1)
        if (incoming == 'D'):
                LO(1, -1)
        if (incoming == 'E'):
                LO(-1, 1)
        if (incoming == 'T'):
                LO(-1, -1)

#---------------------------------------------------
# Locomocao Ripple
while (flag == 8)        :
        incoming = ser.read()
        print 'Recebido %s' % incoming
        if (incoming == 'B'):
                flag = 0
        if (incoming == 'F'):
                LR(1, 1)
        if (incoming == 'D'):
                LR(1, -1)
        if (incoming == 'E'):
                LR(-1, 1)
        if (incoming == 'T'):
                LR(-1, -1)

    #---------------------------------------------------
```