UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FELIPE MICHELS FONTOURA
LEANDRO PIEKARSKI DO NASCIMENTO
LUCAS LONGEN GIOPPO

ROBÔ HEXÁPODE CONTROLADO POR FPGA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2013

FELIPE MICHELS FONTOURA
LEANDRO PIEKARSKI DO NASCIMENTO
LUCAS LONGEN GIOPPO

ROBÔ HEXÁPODE CONTROLADO POR FPGA

Trabalho de Conclusão do Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso II do Curso Superior em Engenharia de Computação dos Departamentos Acadêmicos de Eletrônica e Informática da Universidade Tecnológica Federal do Paraná, apresentado como requisito parcial para obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Carlos Raimundo Erig Lima

CURITIBA
2013

FELIPE MICHELS FONTOURA
LEANDRO PIEKARSKI DO NASCIMENTO
LUCAS LONGEN GIOPPO

ROBÔ HEXÁPODE CONTROLADO POR FPGA

Este trabalho de conclusão de curso foi julgado e aprovado como requisito parcial para obtenção do título de Engenheiro de Computação pela Universidade Tecnológica Federal do Paraná.

Curitiba, 14 de maio de 2013.

____

Prof. Dr. Hugo Vieira Neto
Coordenador de Curso
Departamento Acadêmico de Eletrônica

____

Prof. Dr. Dario Eduardo Amaral Dergint
Responsável pela Disciplina de Trabalho de Conclusão de Curso II
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

______________________________ Prof. Dr. Carlos Raimundo Erig Lima Orientador	______________________________ Prof. Dr. Douglas Paulo Bertrand Renaux
______________________________ Prof. Dr. Dario Eduardo Amaral Dergint	

Let’s start with the three fundamental Rules of Robotics.... We have: one, a robot may not injure a human being, or, through inaction, allow a human being to come to harm. Two, a robot must obey the orders given it by human beings except where such orders would conflict with the First Law. And three, a robot must protect its own existence as long as such protection does not conflict with the First or Second Laws. (ASIMOV, Isaac, 1942)

Vamos começar com as três Leis Fundamentais da Robótica... nós temos: um, Um robô não pode causar dano a um ser humano nem, por omissão, permitir que um ser humano seja ferido. Dois, um robô deve obedecer às ordens dadas por seres humanos, exceto quando essas ordens entrarem em conflito com a Primeira Lei. E três, um robô deve proteger sua própria existência, desde que essa proteção não entre em conflito com a Primeira nem com a Segunda Lei. (ASIMOV, Isaac, 1942)

RESUMO

FONTOURA, Felipe Michels; GIOPPO, Lucas Longen; DO NASCIMENTO, Leandro Piekarski. **Robô Hexápode Controlado por FPGA.** 2013. Monografia (Graduação) – Curso de Engenharia de Computação, UTFPR, Curitiba, Brasil.

Robôs hexápodes são comumente utilizados como ferramenta de estudo de robótica, logo é de interesse acadêmico a elaboração de um robô hexápode com especificação aberta. O objetivo geral do projeto descrito é desenvolver um robô hexápode controlado por uma FPGA, recebendo comandos de um computador. O desenvolvimento segue o processo de desenvolvimento em espiral, em três etapas: projeto, construção e testes. Foram necessários diversos estudos para sua concretização, especialmente o da cinemática inversa. O projeto constitui-se de seis partes: mecânica do robô, eletrônica dos motores, hardware de controle robô, firmware, driver de comunicação e software de interface gráfica. A estrutura mecânica é a MSR-H01, desenvolvida pela Micromagic Systems, com três motores por pata. Utilizaram-se seis motores Corona DS329MG nos ombros, e doze BMS-620MG nas demais articulações, todos alimentados por uma fonte ATX (140W em regime na linha de 5V), isolada do restante do hardware por acoplamento ótico. O hardware de controle do robô engloba a FPGA e seus periféricos: os optoacopladores dos motores, o magnetômetro (HMC5883), o acelerômetro (ADXL345) e o módulo XBee. A FPGA gera os sinais de controle para os motores e para os demais dispositivos e embarca um processador NIOS II, com arquitetura RISC de 32 bits de até 250 DMIPS. O firmware, desenvolvido em linguagem C++, é responsável pela leitura dos sensores, por enviar sinais de controle para os motores e por comunicar-se com o driver, desenvolvido em Java, através do módulo XBee. O software de interface gráfica permite ao usuário enviar comandos de movimentação para o robô através do driver e apresenta leituras dos sensores. O resultado final foi um robô capaz de movimentar-se usando cinemática e diversas tecnologias acopladas. Tecnologicamente, o projeto se destaca pela extensibilidade, pois a FPGA permite reprogramação do hardware e o software é modular. Socioeconômicamente, a flexibilidade do robô permite utilização em atividades de ensino e pesquisa. Além disso, sua documentação e sua especificação são abertas, de forma que é possível replicá-lo sem grande esforço.

Palavras-chave: Robô Hexápode. Servomotor. Lógica Reconfigurável. Orientado a objetos. Desenvolvimento em Espiral.

ABSTRACT

FONTOURA, Felipe Michels; GIOPPO, Lucas Longen; DO NASCIMENTO, Leandro Piekarski. **Robô Hexápode Controlado por FPGA (Hexapod Robot Controlled by FPGA).** 2013. Monograph (Undergraduate) – Computer Engineering Course, UTFPR, Curitiba, Brazil.

Hexapod robots are commonly used as platform for studies on robotics, hence it is of academic interest to develop such kind of robot with open specifications. The main goal ot this project is the development of a hexapod robot controlled by FPGA, receiving high-level commands from a computer. The development follows a spiral model in three stages: project, development and tests. Many studies were necessary in order to make the project possible, especially those regarding inverse kinematics. The project itself is divided in six parts: robot mechanics, motor electronics, robot control hardware, firmware, driver and user interface software. The robot is based on a MSR-H01 mechanical structure developed by Micromagic Systems which requires three motors per leg. There are six Corona DS329MG motors on the shoulders, and twelve BMS-620MG on other joints, all supplied by an ATX power source (140W at 5V) and optically isolated from remaining hardware. The robot control hardware includes the FPGA and its peripherals such as optocouplers, magnetometer (HMC5883), accelerometer (ADXL345) and the XBee device. The FPGA generates the control signals for the servos and other devices while embedding a NIOS II processor with 32-bit RISC architecture, capable of performances up to 250 DMIPS. The firmware was developed in C++ and is responsible for reading the sensors, sending control signals to the servos and connecting to the driver, developed in Java, through the XBee channel. The user interface software allows the user to send commands to the robot through the driver and displays readings from the sensors. The result was a robot capable of moving using a combination of inverse kinematics and other technologies. Technologically, the project has the quality of being extensible, as the FPGA allows hardware reprogramming and the software is split into individual modules. Socioeconomically, the flexibility of the robot allows using it for both teaching and research. It is also remarkable that its specification is open and so more robots like this one can be made with little effort.

Keywords: Hexapod Robot. Servomotor, Reconfigurable Logic. Object-Oriented. Spiral Method.

LISTA DE FIGURAS

Figura 1: Diagrama de blocos simplificado do projeto. 17
Figura 2: Diagrama de blocos do projeto..26
Figura 3: "The Different Parts of an FPGA"...27
Figura 4: Kit DE0-Nano...28
Figura 5: Arquitetura macro do NIOS II...29
Figura 6: Módulo XBee S1...30
Figura 7: MSR-H01 Hexapod Kit...32
Figura 8: Servo BMS-620MG...33
Figura 9: Servo Corona DS329MG..34
Figura 10: Exemplo de sistemas de coordenadas em robôs...36
Figura 11: Transformação de um vetor posição P do sistema A para o sistema B, usando matriz de transformação..36
Figura 12: Convenção de numeração das patas...38
Figura 13: Esquemas de movimentação...39
Figura 14: Diagrama do sistema de controle do servomotor próprio...............................40
Figura 15: "Sample I2C Implementation."..41
Figura 16: "I2C Communication Protocol"..41
Figura 17: Diagrama de blocos do hardware embarcado e periféricos............................48
Figura 18: Opções de processador Nios disponíveis no SOPC.......................................49
Figura 19: Configurações da controladora SDRAM, página 1..50
Figura 20: Configurações da controladora SDRAM, página 2..51
Figura 21: Configuração da PLL da SDRAM...51
Figura 22: Configuração da UART...52
Figura 23: Sistema SOPC completo...53
Figura 24: Geometria de uma pata em detalhe..54
Figura 25: Screenshot durante simulação de trajetória...55
Figura 26: Bloco de cálculo da cinemática inversa..56
Figura 27: Bloco de conversão ângulo – largura de pulso...57
Figura 28: Sinal PWM..58
Figura 29: Bloco gerador de sinal PWM..58
Figura 30: Bloco gerenciador de sinais..59
Figura 31: "Internal structure I2C Master Core"...60
Figura 32: Servo desmontado..63
Figura 33: Configurações de LM555..64

Figura 34: Esquemático do circuito da PCI da FPGA..66
Figura 35: Pinos de I/O da FPGA (GPIO-0 e GPIO-1)...67
Figura 36: Pinos da IMU...68
Figura 37: Pinos do Módulo XBee...70
Figura 38: Diagrama da PCI da FPGA...71
Figura 39: PCI da FPGA com todos os componentes soldados...71
Figura 40: PCI da FPGA ligada com o Xbee, a FPGA e a IMU..72
Figura 41: Esquemático do circuito da PCI dos servos..73
Figura 42: Diagrama da PCI dos servos...75
Figura 43: PCI dos servos sem os componentes soldados...76
Figura 44: Versão final da PCI dos servos parafusada ao hexápode.................................76
Figura 45: Conector Molex de 20 pinos...77
Figura 46: Peças do MSR-H01 com alguns servos encaixados...81
Figura 47: Kit MSR-H01 montado com todos os servos encaixados..................................82
Figura 48: Hexápode com as PCIs e com a FPGA mas sem a PCI superior ter sido soldada...83
Figura 49: Circuito com as resistências geradas pelo "cordão umbilical"...........................84
Figura 50: Hexápode completo..85
Figura 51: Formato de uma mensagem do protocolo de baixo nível.................................87
Figura 52: Formato geral de mensagens do protocolo entre o robô e o driver...................89
Figura 53: Formato da mensagem de definir movimento para o robô................................90
Figura 54: Formato da mensagem de ler sensor para o robô..90
Figura 55: Formato da mensagem de movimento terminado com sucesso........................90
Figura 56: Formato da mensagem de movimento abortado...90
Figura 57: Formato da mensagem de enumerar portas do cliente....................................91
Figura 58: Formato da mensagem de abrir porta do cliente..92
Figura 59: Formato da mensagem de movimentar do cliente..92
Figura 60: Formato da mensagem de ler sensor do cliente...92
Figura 61: Formato da mensagem de movimento terminado com sucesso........................93
Figura 62: Formato da mensagem de movimento abortado...93
Figura 63: Diagrama de casos de uso...94
Figura 64: Diagrama de classes de baixo nível..99
Figura 65: Diagrama de classes de protocolo...101
Figura 66: Diagrama de classes do firmware..102
Figura 67: Statechart do firmware..103

Figura 68: Diagrama de classes do driver..105
Figura 69: Statechart do driver..106
Figura 70: Diagrama de classes da biblioteca de cliente..107
Figura 71: Statechart da biblioteca de cliente..108
Figura 72: Tela inicial do software de interface gráfica..109
Figura 73: Tela de conexão com robô do software de interface gráfica............................109
Figura 74: Tela de execução de movimentos e leitura de sensores do software de interface gráfica...110
Figura 75: Tela de movimento em andamento...111
Figura 76: Diagrama de Gantt do cronograma preliminar..128

LISTA DE TABELAS

Tabela 1: Preços de componentes..25
Tabela 2: Comparação entre servos HS-645MG e BMS-620MG..33
Tabela 3: Comparação entre servos HS-225MG e Corona DS329MG...............................33
Tabela 4: Configuração dos pinos do conector BOARD-C0..67
Tabela 5: Configuração dos pinos do conector BOARD-C1..68
Tabela 6: Configuração dos pinos da IMU...69
Tabela 7: Configuração dos pinos do Módulo XBee na PCI...70
Tabela 8: Configuração dos pinos da Fonte ATX utilizados na PCI dos Servomotores......74
Tabela 9: Pinos de saída do conector de 20 pinos da fonte ATX..79
Tabela 10: Tabela de gastos do projeto...126
Tabela 11: Tabela resumida de atividades do projeto...130

LISTA DE SIGLAS

AC	Alternating Current (Corrente Alternada)
CA	Corrente Alternada
CC	Corrente Contínua
CI	Circuito Integrado
CPU	Central Processing Unit (Unidade Central de Processamento)
DC	Direct Current (Corrente Contínua)
FPGA	Field-programmable Gate Array
GUI	Graphical User Interface (Interface Gráfica com o Usuário)
I/O	Input/Output (Entrada/Saída)
OO	Orientação a Objetos
PCI	Placa de Circuito Impresso
RAM	Random Access Memory
ROM	Read-only Memory
RTOS	Real-time Operating System (Sistema Operacional de Tempo Real)

SUMÁRIO

1 Introdução....15
1.1 Motivação e justificativa....16
1.2 Objetivos e escopo....16
1.2.1 Objetivo geral....16
1.2.2 Objetivos específicos....16
1.3 Requisitos do sistema....18
1.3.1.1 Requisitos funcionais....18
1.3.1.2 Requisitos não-funcionais....18
1.4 Metodologia....19
1.4.1 Fundamentos....19
1.4.2 Tecnologias....19
1.5 Estrutura do trabalho....20
1.5.1 Hardware....20
1.5.2 Software....21
2 Estudos....23
2.1 Produtos similares....23
2.2 Viabilidade....24
2.2.1 Viabilidade financeira....24
2.2.2 Viabilidade técnica....25
2.3 Dispositivos....25
2.3.1 Diagrama de Blocos....25
2.3.2 FPGA – Kit DE0-Nano....26
2.3.3 Processador embarcado – Nios II....28
2.3.4 RTOS – MicroC (μC/OS-II)....29
2.3.5 ZigBee – Xbee S1....30
2.3.6 Acelerômetro – ADXL345....31
2.3.7 Magnetômetro – HMC5883....31
2.3.8 Estrutura mecânica – MSR-H01....31
2.3.9 Servomotores – BMS-620MG & Corona DS329MG....32
2.3.10 Fonte de alimentação dos servos – Fonte ATX 450W....34
2.3.11 Estação Base....35
2.4 Parte teórica....35
2.4.1 Cinemática inversa....35
2.4.2 “Gait”....37

2.4.3 Desenvolvimento de um servomotor próprio..39
2.4.4 I2C...41
2.4.5 Alocação dinâmica, alocação estática e object pooling....................................42
2.4.6 Garbage collection e reference counting...43
2.5 Convenções de programação...44
2.5.1 Regras de nomeação...45
2.5.2 Regras de codificação...46
2.5.3 Regras de estilo..46
2.6 Considerações...47
3 Desenvolvimento..48
3.1 Hardware na FPGA...48
3.1.1 NIOS – Microcontrolador Embarcado...48
3.1.2 Cinemática inversa..53
3.1.3 Conversor entre ângulo e largura de pulso...57
3.1.4 Gerador de sinal PWM...57
3.1.5 Gerenciador de sinais para os servomotores...58
3.1.6 Comunicação I2C...60
3.1.6.1 Software de interfaceamento...61
3.2 Hardware externo...62
3.2.1 Desenvolvimento dos servomotores próprios..62
3.2.2 Placas de Circuito Impresso...65
3.2.2.1 PCI da FPGA..65
3.2.2.2 PCI dos Servos...72
3.2.3 Fonte de alimentação..77
3.3 Montagem do hexápode..80
3.4 Protocolos de comunicação...85
3.4.1 Protocolo de baixo nível...86
3.4.1.1 Mensagens...86
3.4.1.2 Controle de fluxo..87
3.4.2 Protocolos de alto nível...88
3.4.2.1 Protocolo entre robô e driver..89
3.4.2.2 Protocolo entre cliente e driver...91
3.5 Software...93
3.5.1 Projeto de software de interface gráfica...93
3.5.1.1 Requisitos funcionais...93

3.5.1.2 Requisitos não-funcionais..94
3.5.1.3 Casos de uso..94
3.5.1.3.1 Enviar comandos ao robô..94
3.5.1.3.2 Obter leituras dos sensores..96
3.5.2 Detalhes da implementação em C++..97
3.5.2.1 Alocação de memória..97
3.5.2.2 Gerência de memória..97
3.5.3 Código de baixo nível..98
3.5.4 Código de alto nível..100
3.5.4.1 Protocolo..100
3.5.4.2 Firmware..101
3.5.4.3 Driver..104
3.5.4.4 Cliente..107
3.5.4.5 Interface gráfica..108
3.6 Movimentos..111
3.6.1 Movimentos básicos..112
3.6.2 Movimentos com magnetômetro..113
3.6.3 Movimentos demonstrativos..115
3.7 Considerações..116
4 Plano de testes..119
4.1 Caso de teste I: Conexão com o driver..119
4.2 .Caso de teste II: Conexão com o robô..119
4.3 Caso de teste III: Posição de repouso do robô..120
4.4 .Caso de teste IV: Movimentação do robô..120
4.5 Caso de teste V: Leitura de sensores do robô..121
4.6 Considerações..121
5 Gestão..122
5.1 Escopo..122
5.2 Custos e Cronograma..124
5.3 Análise de Riscos..131
5.4 Considerações..133
6 Trabalhos futuros..134
7 Considerações finais..136
Apêndice A: Movimentos suportados pelo robô..147
Apêndice B: Sensores suportados pelo robô..148

Apêndice C: Mensagens do protocolo entre robô e driver..149
Apêndice D: Mensagens do protocolo entre cliente e driver...151
Apêndice E: Tabela completa de atividades realizadas..153
Apêndice F: Riscos identificados..156

1 Introdução

A roda está indubitavelmente entre as invenções mais brilhantes do ser humano. Há um vasto repertório de artefatos móveis que se utilizam de rodas para realizar seus objetivos de deslocamento, como é o caso de diversos tipo de robôs. Há desde robôs mais simples, com duas rodas, até os omnidirecionais, como os utilizados em competições de futebol de robôs[1]. Apesar de amplamente utilizados, eles compartilham de dificuldades de locomoção em superfícies que não sejam lisas ou pouco rugosas[2]. Como eles não funcionam bem em certos ambientes, há um incentivo para elaborar robôs que utilizem outro meio de locomoção, como patas por exemplo. Tais robôs poderiam ser utilizados para acesso a lugares inóspitos, como um território devastado por terremoto ou afetado por radiação[3].

É uma tendência atual que o movimento de robôs com patas (que são, em sua maioria, bípedes, quadrúpedes e hexápodes) seja biologicamente inspirado, isto é, baseado no movimento de seres vivos[2], em especial insetos. Há também robôs que têm seu modelo em outros tipos de animais, como o robô BigDog, desenvolvido pela empresa Boston Dynamics, que tem por base o movimento de cães[4].

O problema mais estudado no que diz respeito a robôs “multipodes” consiste em determinar a melhor sequência de levantar, deslocar e baixar as patas que resulte em uma locomoção mais rápida ou com menor gasto de energia. Existem também estudos que buscam determinar qual o melhor formato corporal desses robôs, confrontando corpos hexagonais e retangulares[5]. Para descrever e coordenar o modo de andar e de se locomover de robôs hexápodes, várias estratégias podem ser adotadas, entre elas sistemas não-lineares[6] e algoritmos genéticos[7]. Há ainda trabalhos bastante completos sobre o desenvolvimento de hexápodes controlados por cabos. Tais trabalhos são completos ao ponto de descrever propriedades dinâmicas e de performance[8].

No ano de 1949 já existia a ideia de robôs hexápodes, mas apenas em torno do ano de 1997 foi possível realizar esses robôs em termos de custos computacionais[9]. Os hexápodes são os robôs paralelos mais utilizados da indústria, principalmente para simulação de movimentos (ex: simuladores de voo). De poucos anos para cá há intenção de utilizá-los em linhas de montagem de aeronaves devido a precisão que possuem[10].

1.1 Motivação e justificativa

Uma motivação para o projeto é a elaboração de um robô com documentação totalmente aberta, de forma que possa ser utilizado futuramente como plataforma de estudo de robótica ou mesmo como base para um hexápode comercial. Além disso, todos os integrantes da equipe já tem experiência prévia com robôs que fazem uso de rodas e tem interesse em estudar um tipo diferente de estrutura mecânica, de eletrônica e sobretudo de forma de locomoção.

Há ainda o interesse de estudar e implementar um projeto utilizando uma plataforma de hardware completamente distinta das utilizadas anteriormente pelos integrantes, mais especificamente uma FPGA (*field-programmable gate array*). A principal ideia, neste sentido, é explorar as capacidades da lógica reconfigurável, utilizando recursos que um microcontrolador comum não possui.

1.2 Objetivos e escopo

1.2.1 Objetivo geral

O objetivo geral do projeto é o desenvolvimento e a montagem de um robô hexápode. Tal robô deve ser controlado por um dispositivo de lógica reconfigurável, e ser capaz de receber comandos a partir de algum dispositivo externo. Dentro do possível a arquitetura do robô deve ser extensível, no sentido de permitir a adição de novos recursos e/ou funcionalidades sem necessidade de alterar significativamente o design de software e hardware. Idealmente, deve ser possível adicionar funcionalidades de hardware sem alterar (ou alterando minimamente) a forma que os dispositivos já presentes no robô estão conectados à FPGA. De forma semelhante, o software existente deve requerer pouca ou nenhuma alteração para incluir novas funcionalidades.

1.2.2 Objetivos específicos

O projeto envolve diversas áreas da engenharia de computação, sendo efetivamente uma aplicação da maioria dos conhecimentos adquiridos durante o curso. O objetivo primário do projeto é, portanto, a aprendizagem e aplicação das técnicas necessárias para a confecção do robô.

O robô deve comunicar-se com uma estação-base através de comunicação *wireless*. Ele deve locomover-se ao receber comandos vindos de um *driver* na estação-base. O software de interface com o usuário deve comunicar-se com este *driver* utilizando uma biblioteca orientada a objetos a ser desenvolvida. O robô deve ao menos ser capaz de andar em uma direção em particular, saber em que direção está orientado, e girar sua estrutura para uma certa direção. Na figura 1 é apresentado um diagrama de blocos simplificado do sistema.

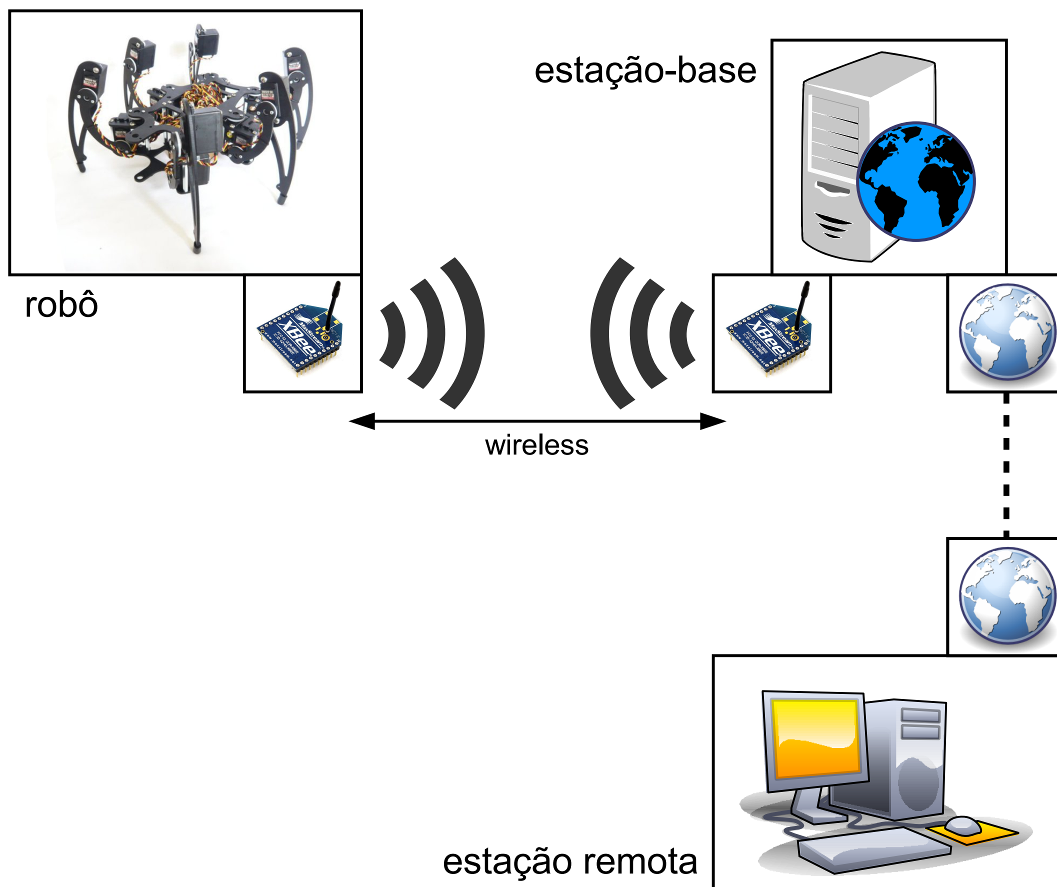


Figura 1: Diagrama de blocos simplificado do projeto.

Fonte: Autoria própria

Uma vez efetivado, o projeto poderia ser estendido para mais funcionalidades e ter outras aplicações. Portanto um dos objetivos é criar um projeto que de alguma forma seja expansível futuramente podendo ser integráveis outros dispositivos como garras, sensores térmicos, câmeras, microfones, dentre outros. Este é um dos incentivos para o uso de uma FPGA pois com ela é possível ampliar o hardware de maneira mais fácil.

Na FPGA, há uma grande quantidade de conectores livres, e é possível alterar o circuito digital que ela implementa, adicionando novos blocos. Portanto, para adicionar

novos dispositivos, não deve ser necessário alterar os que já estão conectados ao robô. O único elemento de hardware que requer grande mudança neste caso é a placa de circuito impresso.

É tido como objetivo também explorar os limites do hardware disponível. Certos movimentos do robô podem ser inspirados nos de algum animal, como, por exemplo, uma aranha ou uma formiga.

1.3 Requisitos do sistema

1.3.1.1 Requisitos funcionais

RF1 . O robô deve manter suas patas em uma posição estável enquanto não estiver em movimento.

RF2 . O robô deve ser capaz de movimentar suas patas.

RF3 . O robô deve ser capaz de se comunicar com um software de *driver* na estação base.

RF4 . O robô deve movimentar-se apenas quando receber comandos de movimentação a partir do *driver*.

RF5 . O robô deve abortar o movimento em andamento quando receber comandos de parada a partir do *driver*.

RF6 . O robô deve enviar para o *driver* as leituras de seus sensores quando receber comandos de leitura de sensor a partir do *driver*.

RF7 . O *driver* de comunicação com o robô deve receber conexões de uma biblioteca de comunicação.

RF8 . A biblioteca de comunicação deve permitir a comunicação com o *driver*.

RF9 . O software de interface com o usuário deve conectar-se ao *driver* usando a biblioteca de comunicação.

1.3.1.2 Requisitos não-funcionais

RNF1. O hardware do robô deve ser realizado em um dispositivo de lógica reconfigurável.

RNF2. O robô deve ter seis patas.

RNF3. O robô deve ter sensores.

RNF4. O robô deve ter três motores por pata.

RNF5. O robô deve ser alimentado por cabo.

RNF6. A comunicação entre o robô e a estação base com o *driver* deve ser sem fio (*wireless*).

RNF7. A comunicação entre o *driver* e a biblioteca de comunicação deve ocorrer por *sockets* TCP.

1.4 Metodologia

1.4.1 Fundamentos

O projeto do robô hexápode envolve, entre outros, a aplicação de conhecimentos na área de controle, que são aplicado para movimentação das patas, lógica reconfigurável, para toda a integração das partes e controle de cada parte do robô, e sistemas embarcados, para a elaboração do sistema embarcado. Além dessas áreas, espera-se que haja uma quantidade razoável de modelagem matemática para a movimentação do robô, especialmente para o movimento das patas. O projeto do circuito do robô ainda explora conhecimento de linguagens de modelagem de circuitos, conhecimento de circuitos digitais e analógicos, conhecimento de filtros, processamento digital de sinais e arquitetura de computadores.

No projeto do software são utilizados conhecimentos de fundamentos de programação (para a programação do sistema de maneira geral), estrutura de dados (para organização dos dados recebidos) e comunicação de dados (para que seja realizada uma comunicação confiável entre a estação base e o robô). Também serão abordados conceitos de análise e projetos de sistemas e engenharia de software para realizar a documentação, planejamento e organização do software.

1.4.2 Tecnologias

As principais tecnologias que deverão ser utilizadas no projeto são: FPGA, servomotores, meios de comunicação *wireless*, linguagem C++ e linguagem Java.

Para fazer o controle do robô foi utilizada uma FPGA que coordena os servomotores e realiza leitura dos sensores. O *firmware* na FPGA ainda comunica-se com um software na estação base, no caso um computador.

Os servomotores serão utilizados para controlar o movimento de cada articulação das pernas do hexápode. Cada perna deverá possuir três servomotores para que ele possa realizar movimentos diversos, com três graus de liberdade. Entre os movimentos

possíveis encontram-se: deslocamento horizontal das patas para que ele possa se locomover para frente, deslocamento vertical da perna para que seja possível erguê-la e um terceiro para deslocamento angular, para rotacionar a perna em torno do joelho. Essa terceira articulação funciona como uma espécie de joelho, capaz de movimentos mais complexos como deslocamentos laterais.

Para que não seja necessária a utilização de um cabo conectando a FPGA ao computador, foi utilizado ZigBee, uma tecnologia de rádio frequência[11]. O ZigBee funciona de forma similar a um cabo serial, se utilizado no modo transparente.

Na estação base há um *driver* de comunicação que envia comandos ao robô. O *driver* de baixo nível é capaz de comunicar-se com uma biblioteca, que serve de interface entre o *driver* e o software de interface gráfica. O *driver*, a biblioteca de comunicação e o software de interface gráfica foram desenvolvidos em linguagem Java, em ambiente Eclipse[12].

Para a modelagem dos “circuitos” de controle que deverão ser carregados na FPGA será utilizada a linguagem de construção de circuitos VHDL. A ferramenta escolhida para elaboração do código VHDL foi o *software* Quartus II.

1.5 Estrutura do trabalho

O projeto a ser desenvolvido, como lida com robótica, envolverá ferramentas e recursos de hardware e software. Essa seção fornece uma descrição preliminar das ferramentas e recursos a serem utilizados.

1.5.1 Hardware

Para o desenvolvimento das pernas do hexápode, foram usados servomotores, como é comum em robôs desse tipo. Cada pata do robô possui três motores, permitindo uma maior mobilidade, este é um número bastante utilizado em robôs similares[5]. Dentro do possível, evitou-se fazer peças sob medida, priorizando o uso de peças disponíveis comercialmente, com o objetivo de reduzir os custos. Considerou-se anteriormente a possibilidade de construir os próprios servomotores a partir de motores DC simples, para obter maior precisão com menor custo. Essa última ideia foi abandonada no estágio inicial do projeto, como descrito na seção de desenvolvimento, devido ao fato de demandar muito tempo sem diminuir significativamente o custo ou aumentar consideravelmente a precisão dos motores.

O controle de cada pata, bem como do robô como um todo, foi construído em uma FPGA , que é um dispositivo de lógica reprogramável. A opção por uma FPGA em lugar de um microcontrolador se deve à sua capacidade de lidar com vários dispositivos interagindo em paralelo[13], como acontece em um robô hexápode.

Duas placas de circuito impresso foram desenvolvidas para fazer toda a conexão de potência e alimentação do robô. Elas são responsáveis tanto por distribuir a energia do robô, quanto por organizar outros dispositivos periféricos como acelerômetro, giroscópio e motores.

O grupo optou por utilizar um kit de desenvolvimento DE0-Nano, produzido pela Terasic, que utiliza uma FPGA Cyclone IV 4C22[14] da Altera. Sua escolha é devido as pequenas dimensões e por ele possuir algumas funcionalidades muito úteis para o projeto: um acelerômetro com 13 bits de precisão, 66 pinos genéricos de I/O, conexão USB, dentre outras.

Originalmente, imaginava-se que o robô poderia receber alimentação de uma bateria, de forma a poder deslocar-se com maior liberdade do que alimentado por cabo. Chegou-se à conclusão que seria inviável alimentar o robô por bateria (maior preço, maior peso, menor vida útil), portanto foi utilizado um cabo de alimentação.

Para a comunicação com a estação base, de onde o robô receberá instruções, foi utilizado um dispositivo ZigBee. A opção pelo ZigBee em lugar de outros meios deve-se à relativa simplicidade de uso, a seu preço relativamente baixo e pela qualidade da tecnologia[11]. Funciona bem para as distâncias de comunicação desejadas para o projeto (até 10 metros de distância) e é tratada de forma similar a um dispositivo de interface serial.

O computador da estação base, com o *driver* de comunicação, deve ter, ao menos, uma porta USB, sistema operacional Windows® XP (ou superior), mouse, teclado e monitor. Para o desenvolvimento computadores com especificação semelhante devem ser utilizados, preferencialmente com capacidades de processamento suficientes para o uso das ferramentas de software citadas a seguir.

1.5.2 Software

O robô deve receber comandos de um *driver* na estação base, o qual foi desenvolvido em linguagem Java. Originalmente a linguagem utilizada para seu desenvolvimento seria C++, mas esta ideia foi abandonada ao longo do projeto devido à

maior facilidade de desenvolver, manter e testar código em uma linguagem que apresente gerenciamento automático de memória. A linguagem Java é um exemplo de linguagem com tal característica[15], com a qual os membros da equipe já têm prática. A biblioteca de comunicação com o robô e a interface gráfica também são feitas em linguagem Java, de forma que há aproveitamento de parte do código do *driver* e menos linhas de desenvolvimento paralelas no projeto. Para que haja comunicação ZigBee deve haver uma forma de receber/enviar comandos tanto no PC como no sistema embarcado, ou seja, um protocolo único implementado em duas linguagens distintas, em plataformas distintas.

Para o desenvolvimento do sistema de lógica reconfigurável foi utilizado o software proprietário Quartus II 12.0[16]. Esta opção se deve à experiência de todos os membros da equipe na utilização desse software e ao fato da FPGA utilizada ser da empresa Altera.

Para o desenvolvimento do software embarcado, do *driver*, da biblioteca e do software de interface gráfica foi utilizada a IDE Eclipse[12]. Tal escolha se deve à experiência que os membros da equipe têm com esta ferramenta e também por ser gratuita e extensível, permitindo programação em mais de uma linguagem.

Na elaboração das PCIs foi utilizado o software Eagle[17]. Esse software em particular foi escolhido dada a experiência de um dos membros da equipe com o mesmo.

Todos os recursos software a serem utilizados nesse trabalho já foram previamente utilizados inúmeras vezes em outras oportunidades durante o curso de Engenharia de Computação. Portanto, os membros da equipe já possuem uma certa experiência com todos eles.

2 Estudos

2.1 Produtos similares

Existe uma ampla variedade de robôs hexápodes, sejam de projetos acadêmicos ou comerciais, os quais diferem tanto em característica mecânicas quanto eletrônicas. Pesquisaram-se alguns robôs hexápodes de destaque, cujas características são apresentadas a seguir.

O robô hexápode mais barato encontrado é o Stiquito, desenvolvido originalmente por Jonathan Mills, da universidade de Indiana (Estados Unidos da América). O Stiquito é diferente da maioria dos robôs hexápodes por não utilizar motores, mas sim nitinol, uma liga metálica capaz de contrair-se e dilatar-se, de forma semelhante a um músculo. Ele também não conta com qualquer sistema microcontrolado, sendo normalmente controlado pela porta paralela do computador. Suas aplicações são geralmente educativas, devido à sua simplicidade: ele tem apenas um grau de liberdade (mas é possível comprar um kit para adicionar mais um grau de liberdade). Na loja oficial, um Stiquito pode ser comprado por US$ 22, ou US$ 24 para compradores internacionais (dados de 22 de janeiro de 2013), mais o preço do frete e impostos[18].

Outro robô de destaque é o RHex. Foi desenvolvido originalmente como um projeto multidisciplinar patrocinado pela DARPA (Defense Advanced Research Projects Agency, "Agência de Projetos de Pesquisa Avançada em Defesa", agência do governo dos Estados Unidos da América). O robô é construído com motores Maxon RE118751 (motor de corrente contínua com escovas), cujo controle é feito por software (controle PD digital, amostrado a 1 kHz), usando um processador Intel 486[19]. O Rhex é capaz de andar em terrenos bastante acidentados, tem uma estrutura mecânica à prova d’água, pode ser controlado à distância de até 700 metros e apresenta câmeras e faróis na parte frontal e na parte traseira[20].

A empresa LynxMotion produz soluções comerciais completas de robôs hexápodes, incluindo chassis, motores, eletrônica e até software de exemplo. Há diversos modelos de robôs disponíveis, a saber: AH3-R, BH3-R, CH3-R, A-Pod, T-Hex, Phoenix, BH3, AH3, MH2, MAH3-R. Os robôs podem acompanhar dois tipos de placas de microcontroladores, a BotBoard II, que utiliza controladores Basic Atom ou Basic Atom Pro de 24 ou 28 pinos, ou a BotBoarduino, que é semelhante a um Arduino Duemilanove. Dependendo do

modelo do robô, são utilizados servos HS-422, HS-475, HS-485, HS-645 e/ou HS-755. O controle dos servos é realizado com auxílio de um controlador SSC-32, que, entre outras funcionalidades, ajuda a suavizar os movimentos. Os preços dos robôs, na loja oficial, variam entre US$ 459,43 e US$ 1337,00, mais frete e impostos (dados de 18 de janeiro de 2013)[21].

Outro robô hexápode comercial é o Hexy. Ele é um robô de baixo custo, com estrutura mecânica de acrílico e servomotores Tower Pro SG92R (sem engrenagens metálicas). O microcontrolador utilizado é um Servotor32, baseado no Arduino Leonardo, e o robô acompanha um software que permite a programação dos seus movimentos utilizando arquivos de texto. O preço de um kit de montagem do Hexy, na loja oficial, é de US$ 250,00, mais frete e impostos (dados de 22 de janeiro de 2013)[22].

Há ainda o robô que inspirou o projeto descrito neste documento, o “Robô Hexápode Orientado para Aplicações Pedagógicas”. Tal robô tem uma estrutura mecânica personalizada, desenvolvida sob medida, feita de alumínio e utiliza servo-motores da marca TowerPro de torque 10,5 kgf. Na descrição do projeto em questão, não foi informado seu preço[23].

2.2 Viabilidade

2.2.1 Viabilidade financeira

O controle de uma perna robótica foi realizado como projeto final dos três integrantes da equipe na disciplina de Lógica Reconfigurável, no primeiro semestre de 2011. O sistema projetado foi utilizado como ponto de partida, do robô hexápode.

O projeto teve como custos principais a compra da FPGA, dos servomotores, da estrutura mecânica, das PCIs e dos módulos ZigBee. Além disso, foi comprado um adaptador USB-serial para usar e configurar os módulos ZigBee no PC. A estimativa de gasto com somente essas peças, que constituem a maior parte dos gastos, pode ser vista na tabela 1.

Tabela 1: Preços de componentes.

Fonte: Autoria própria.

Quantidade	Material	Preço por unidade*
1	FPGA DE0-Nano	R$: 380,00
1	Módulo Xbee-Pro ZB	R$ 170,00
1	Placa CON-USBBEE Rogercom com módulo Xbee-Pro ZB	R$ 270,00
9	Driver de motor L298N	R$ 9,00
36	Optointerruptor PHCT103	R$ 3,00
18	Motor D/C simples ou servomotor**	USS$ 6,00 ≈ R$ 25,00
1	PCI personalizada	R$ 250,00
1	Mecânica do robô	R$ 250,00
1	Bateria Li-Pol 14.8V 3Ah (potência)	R$ 152,00
1	Bateria Li-Pol 11.1V 1.5Ah (eletrônica de controle)	R$ 110,00
1	Bússola HMC6352	USS$ 35,00 ≈ R$ 54,60[24]
1	Troca de peças danificadas	20% total = R$ 403,28
Total		R$: 2000,00 ≈ R$2450,00

2.2.2 Viabilidade técnica

Os membros da equipe já tem prática em programação em linguagens Java e C++, em elaboração de circuitos em VHDL, em uso de interfaces de comunicação UART, em uso de tecnologia ZigBee e em uso de motores[11,25]. Os maiores riscos do trabalho encontraram-se na falta de experiência de toda a equipe em questões mecânicas e de eletrônica de potência.

2.3 Dispositivos

2.3.1 Diagrama de Blocos

Na figura 2, apresenta-se o diagrama de blocos do sistema de hardware idealizado. Basicamente, representa-se o sistema embarcado, com as PCIs, os 18 servos, a estrutura

* desconsiderando custos de importação, impostos e frete.

** caso sejam comprados servomotores não serão utilizados optointerruptores

mecânica e a FPGA, o *driver* da estação base, cuja conexão se dá por conexão wireless usando módulos ZigBee e a biblioteca de cliente ligada a um software de interface gráfica. O software de interface gráfica comunica-se com o *driver* através da biblioteca de cliente utilizando *sockets*.

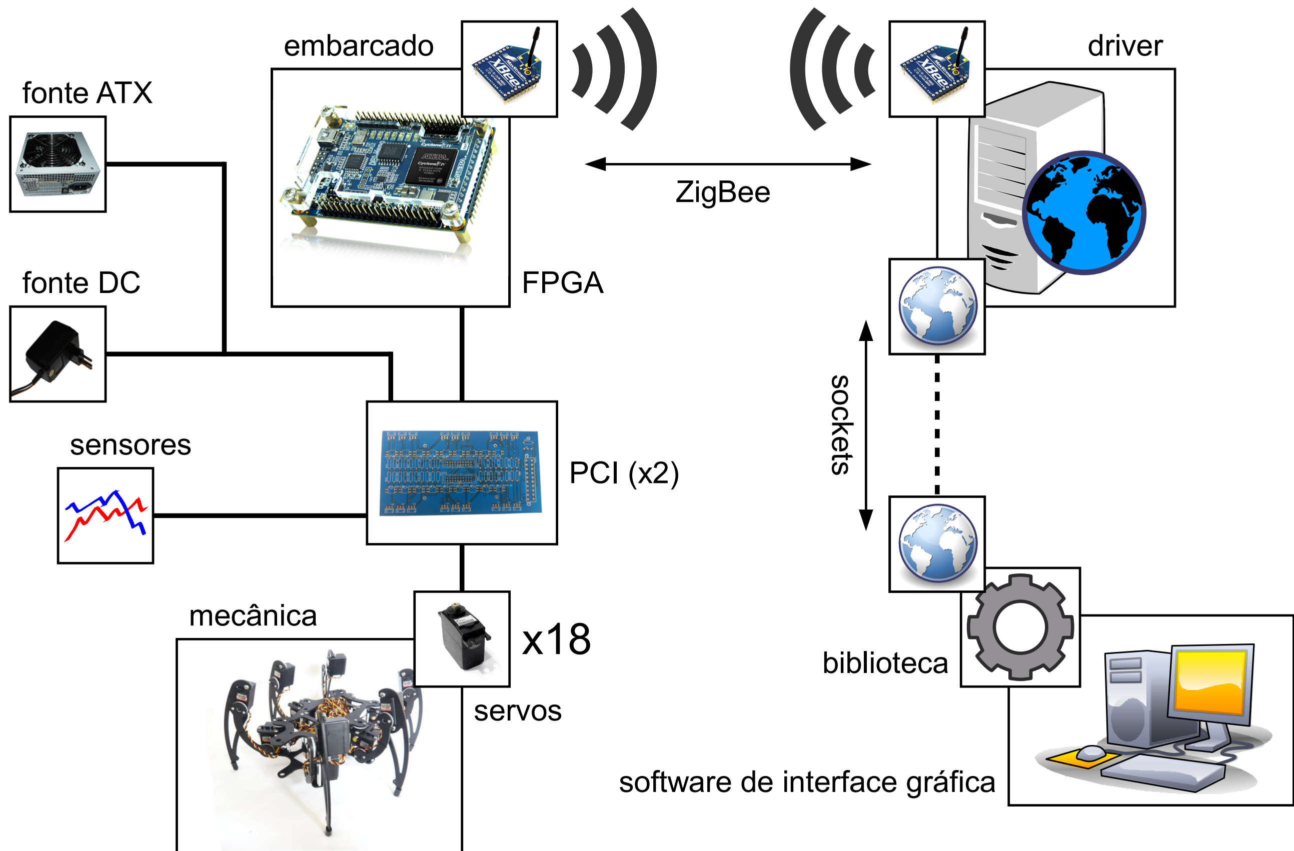


Figura 2: Diagrama de blocos do projeto.

Fonte: Autoria própria.

2.3.2 FPGA – Kit DE0-Nano

A FPGA é um dispositivo semicondutor que pode ser programado depois de fabricado[26]. Graças a essa capacidade, de maneira geral, qualquer circuito que pode ser construído por funções lógicas é implementável em uma FPGA. Atualmente os kits incluem também vários outros dispositivos como memórias RAM/Flash e *transceivers* de alta velocidade.

A FPGA oferece paralelismo real, ou seja, possui seções do hardware destinadas especificamente a cada finalidade sendo executadas de forma independentes[27]. Dessa forma, não há competição entre diversas tarefas por recursos compartilhados de

hardware. Sem mencionar os periféricos de um kit, o chip da FPGA é composto por três unidades básicas, indicadas na figura 3. As interconexões programáveis são utilizadas para implementar um circuito digital composto de blocos lógicos, que se comunicam com dispositivos externos através de blocos de I/O.

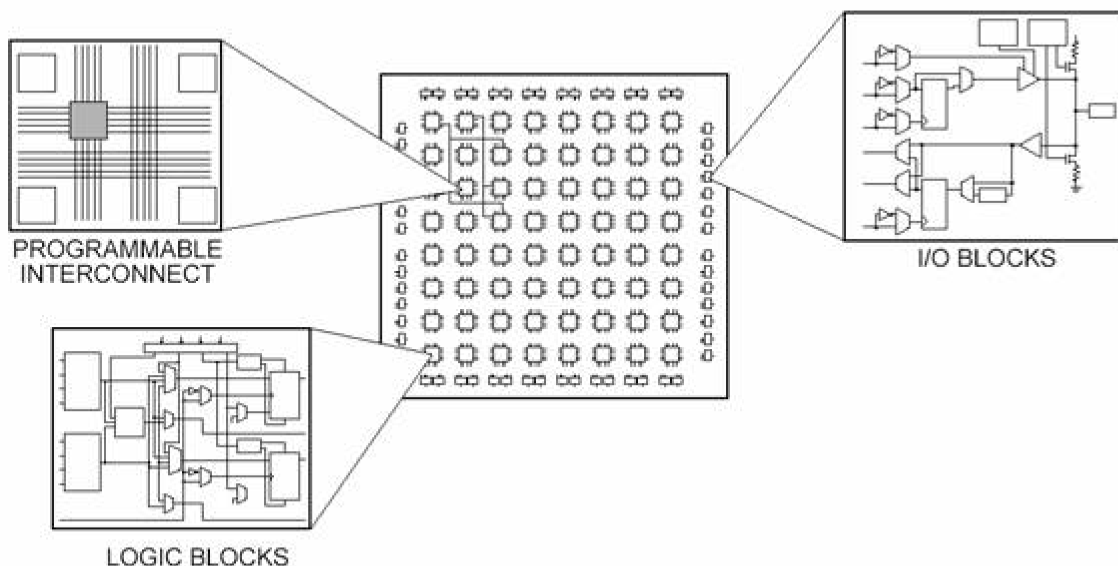


Figura 3: "The Different Parts of an FPGA".
Fonte: <http://www.ni.com/cms/images/devzone/tut/swvvifhq55851.jpg>

A escolha por uma FPGA como unidade central se fez pelo paralelismo exigido pelo projeto. Por exemplo, são controlados 18 motores simultaneamente e é feita a leitura/interpretação de sensores e comunicação com a estação base. O kit escolhido para ser utilizado é o DE0-Nano[14], projetado e produzido pela empresa Terasic e que faz uso do chip FPGA Cyclone IV, da Altera. Os fatores que levaram a escolha desse kit foram o tamanho compacto, bastante adequado a um robô; o modelo de FPGA com número suficiente de elementos lógicos; o acelerômetro e ADC incluídos no kit e a grande quantidade de pinos de I/O. A familiaridade com o software Quartus[16] também pendeu a favor da escolha de um kit que fizesse uso de tecnologia da Altera. A figura 4 mostra o kit utilizado. Fica evidente a pequena dimensão do mesmo.

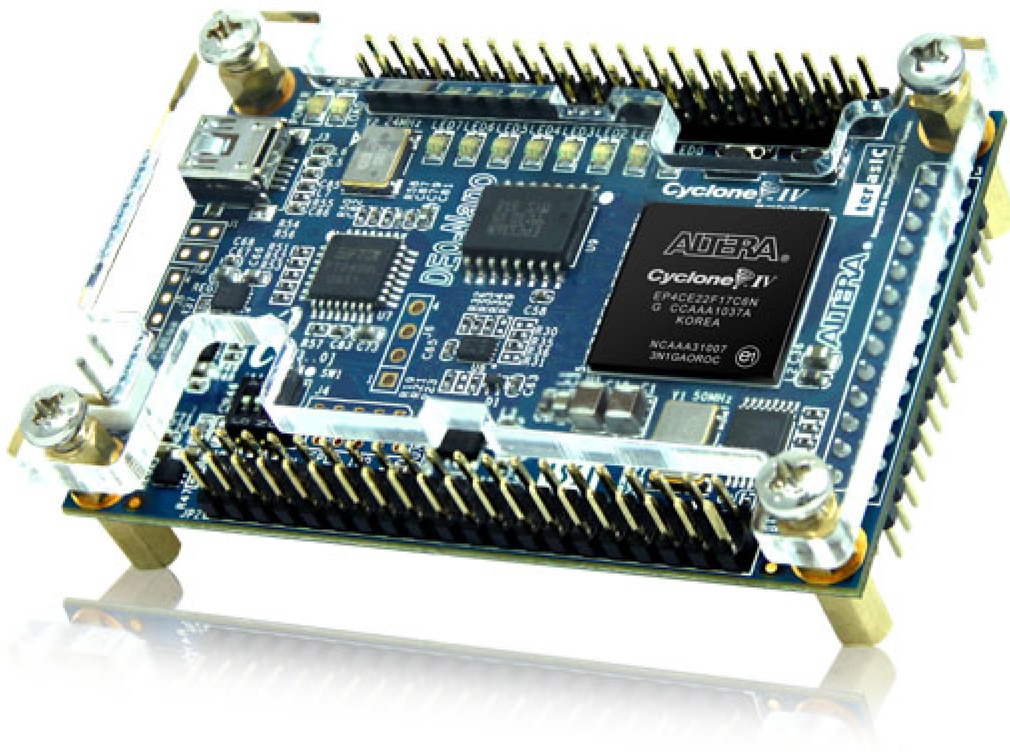


Figura 4: Kit DE0-Nano.

Fonte: <http://www.altera.com/education/univ/images/boards/de0_nano.jpg>

2.3.3 Processador embarcado – Nios II

Um sistema NIOS II é um microcontrolador embarcado, pois possui um processador, conjunto de periféricos e memória todos no mesmo chip. Nesse sistema está incluído um dos 3 possíveis cores NIOS II (*economy*, *standard* ou *fast*), periféricos selecionados de uma vasta gama de opções (que inclui, por exemplo, comunicação serial, processamento de sinais e interfaces como memórias de diferentes tipos, além de possuir um set de instrução próprio). O processador em si possui arquitetura RISC com *word* de 32 bits, instruções de ponto flutuante, entre outras capacidades. Pode ser programado utilizando o NIOS II SBT, baseado em Eclipse, nas linguagens C ou C++. O desempenho é elevado e pode chegar a 250 DMIPS[28].

A grande vantagem do NIOS é ser um processador *softcore* (embarcado). Sua arquitetura é disponibilizada pela Altera e ele pode ser montado de inúmeras maneiras dentro do próprio design da FPGA, completamente integrado ao resto do circuito desenvolvido. Na figura 5 é possível ter uma visão macro do processador e suas

características. Vale ressaltar a possibilidade de *debug* via JTAG, que facilita e muito o desenvolvimento.

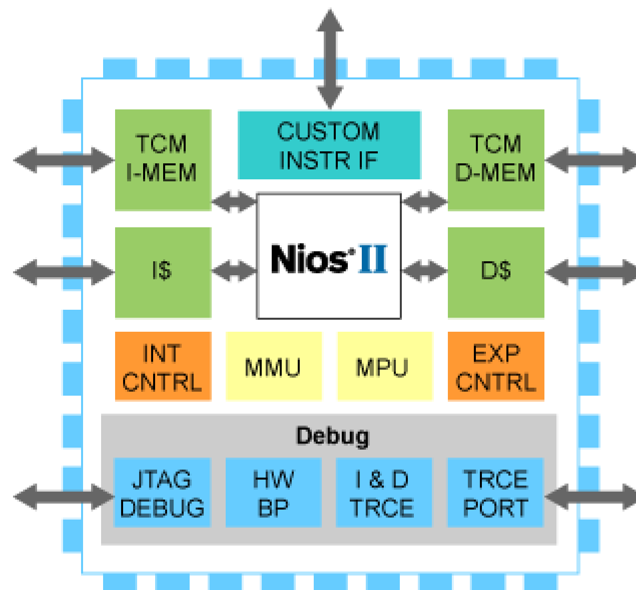


Figura 5: Arquitetura macro do NIOS II.

Fonte: <http://www.altera.com/devices/processor/nios2/images/nios-ii-features.gif>

2.3.4 RTOS – MicroC (µC/OS-II)

Um RTOS ("*real-time operating system*", sistema operacional de tempo real) é um tipo de sistema operacional especializado em operações de tempo real. A principal característica de um RTOS é seu determinismo, ou seja, a previsibilidade de duração das tarefas que são executadas. Apesar de esta ser a característica determinante, também é desejável que seja eficiente e tenha pequeno footprint, tanto em termos de memória de programa quanto em termos de RAM[29]. Sistemas deste tipo são bastante utilizados em aplicações embarcadas, como é o caso do robô hexápode.

O RTOS utilizado é o µC/OS-II, da empresa Micrium. A empresa caracteriza o µC/OS-II como "*real-time deterministic multitasking kernel for microprocessors, microcontrollers and DSPs*" (kernel de tempo real determinístico multi-tarefa para microprocessadores, microcontroladores e processadores digitais de sinais)[30]. O desenvolvimento do sistema operacional foi feito completamente em ANSI-C e inclui muito do que se pode esperar de um RTOS, como semáforos, flags de eventos, filas, gerenciamento de tarefas e tempo.

A existência de *templates* prontos para iniciar o desenvolvimento de *firmware* com µC/OS-II na IDE de programação do Nios[31] pesou bastante a favor no momento da escolha de RTOS. A IDE utilizada para programação do *firmware* é uma versão modificada do Eclipse.

2.3.5 ZigBee – Xbee S1

A tecnologia ZigBee é descrita como a "internet das coisas", segundo a ZigBee Alliance. ZigBee combina baixo custo, baixo consumo de energia e capacidade de controlar redes[32].

Existem muitas implementações diferentes da tecnologia variando de empresa para empresa, mas em comum há a operação na frequência de 2,4 GHz (global, mas variando em alguns continentes) e o padrão de rádio IEEE 802.15.4. Na frequência padrão, a taxa de transmissão pode chegar a 250kbps e o alcance pode chegar a 1600m dependendo da potência dos módulos e das condições do ambiente.

Os módulos utilizados no trabalho são da empresa Digi, cuja implementação da tecnologia ZigBee se chama Xbee[33]. O modelo de módulo utilizado é o Xbee 802.15.4 S1, similar ao da figura 6, que opera na frequência de 2,4GHz, transmissão de 250kbps e alcance de até 90m.

Figura 6: Módulo XBee S1.
Fonte: <http://www.seeedstudio.com/depot/bmz_cache/6/6dbfcfc022db58a32e670912a03095f6.image.300x225.jpg>

Além dos módulos, são utilizadas uma Xbee USB Adapter Board[34] e uma Xbee Adapter Board[35], ambas da Parallax. A primeira auxilia na configuração dos módulos via PC através do software X-CTU, também da Digi[36]. A segunda tem como objetivo adaptar a

pinagem dos módulos para que seja possível conectá-los a placas com espaçamento DIP, incluindo *protoboards,* durante o processo de testes.

2.3.6 Acelerômetro – ADXL345

Acelerômetros como o próprio nome sugere, são utilizados para medir aceleração. O acelerômetro como sensor pode ser utilizado tanto para medir aceleração estática, como a da gravidade, bem como pra medir acelerações dinâmicas, quando há movimentação do dispositivo[37]. No caso do projeto podem ser úteis em diversas frentes, como na obtenção da altura do corpo do robô e na detecção de inclinações indesejáveis do mesmo, bem como as acelerações ocorridas durante a movimentação.

O acelerômetro utilizado é um ADXL345, da Analog Devices. Acelerações nos 3 eixos são medidas com esse dispositivo, com resolução de até 13 bits. Os dados adquiridos são transmitidos de forma digital utilizando interface SPI ou I2C[38]. Vale ressaltar que o kit de FPGA utilizado já possui um sensor desses em seu PCB, facilitando bastante sua utilização. O design do kit, contudo, restringe a comunicação a ser somente SPI.

2.3.7 Magnetômetro – HMC5883

Magnetômetros são instrumentos utilizados para medir a força e algumas vezes a direção de um campo magnético, em muitos casos sendo esse campo o próprio campo magnético da terra[39]. Uma aplicação possível, como no caso do projeto, é utilizar o magnetômetro como uma bússola de baixo custo.

O magnetômetro utilizado é um HMC5883, da Honeywell. Forças magnéticas são medidas nos 3 eixos e digitalizadas por um ADC de 12 bits. A comunicação com o sensor é feita somente por I2C[40]. Para o funcionamento adequado é necessário calibrar o sensor em escala e *offset*. Quanto a escala, o próprio sensor já fornece a possibilidade de gerar um campo magnético arbitrário para coletar as diferenças entre os 3 eixos e normalizá-las. Já quanto a *offset*, é necessário verificar as respostas do sensor e corrigi-las manualmente.

2.3.8 Estrutura mecânica – MSR-H01

Devido a inexperiência da equipe na área de mecânica, mudou-se a ideia inicial de desenvolver a estrutura mecânica do robô. No princípio havia se pensado em fazer um esboço e repassá-lo a um engenheiro mecânico, devidamente capacitado a oficializar o

projeto mecânico. Com o projeto em mãos, seria necessário contratar os serviços de uma oficina para cortar as peças nos tamanhos adequados, bem como adquirir quantidade de material suficiente para essa finalidade. Apenas o corte de peças personalizadas, sem considerar o custo do projeto mecânico e da matéria prima, custa cerca de R$ 400,00. A partir do momento que se percebeu que essa solução não era muito vantajosa financeiramente, no tempo necessário para conclusão e acima de tudo no conhecimento agregado, decidiu-se por adquirir uma estrutura pronta.

Dentre várias opções, a escolhida foi o MSR-H01 da Micromagic Systems, que custa £139.99 (dados de 22 de janeiro de 2013)[41], similar à figura 7. A empresa Micromagic Systems desenvolve chassis de robôs hexápodes, junto às especificações dos motores que devem ser utilizados. O kit contém 26 componentes de alumínio (patas e corpo), além de todos os parafusos necessários para montagem.

Figura 7: MSR-H01 Hexapod Kit.
Fonte: <http://www.hexapodrobot.com/MSRL_image.asp?File=images/MSR-H01/MSR-H01_wake_ISO_s>

2.3.9 Servomotores – BMS-620MG & Corona DS329MG

Existia a possibilidade de comprar o robô hexápode completo na loja da Micromagic Systems, e também de comprar um pacote incluindo os servos, baterias e eletrônica de controle. Excetuando-se a compra conjunta dos servos, os outros produtos foram prontamente descartados, pois um dos focos do projeto é desenvolver a própria parte eletrônica.

Os servos recomendados pela loja para esse robô são os Hitec HS-225MG e HS-645MG[41], sendo o primeiro para a junta da coxa e o segundo para as juntas do fêmur e da

tíbia. Esses motores foram considerados caros e optou-se por procurar alternativas de características semelhantes e preço inferior. Para substituir o HS-225MG foi escolhido o servo Corona DS329MG e para substituir o HS-645MG escolheu-se o BMS-620MG. As tabelas 2 e 3 mostram as diferenças entre os servos. Na figura 8 pode ser visto o BMS-620MG e na figura 9 pode ser visto o Corona DS329MG.

Tabela 2: Comparação entre servos HS-645MG e BMS-620MG

	HS-645MG	**BMS-620MG**
Peso (g)	55,2	50
Dimensões (mm)	40.6x19.8x37.8	40.5x20x42
Torque (kgf.cm) @ 4.8V	7,7	9,1
Torque (kgf.cm) @ 6V	9,6	10,6

Tabela 3: Comparação entre servos HS-225MG e Corona DS329MG

	HS-225MG	**Corona DS329MG**
Peso (g)	31	32
Dimensões (mm)	32.3x16.8x31	32.5x17x34.5
Torque (kgf.cm) @ 4.8V	3.9	3.8
Torque (kgf.cm) @ 6V	4.8	4.5

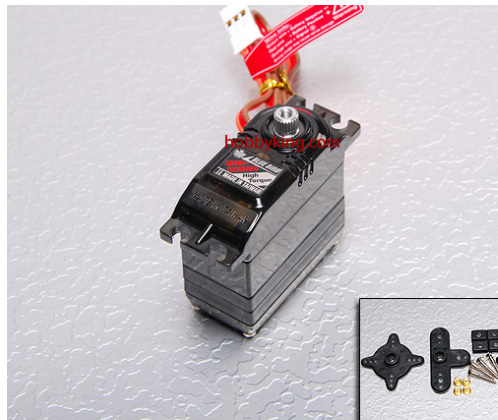


Figura 8: Servo BMS-620MG.

Fonte: <http://www.hobbyking.com/hobbyking/store/__8776__ bms_620mg_high_torque_servo_metal_gear_9_1kg_15sec_50g.html >

Figura 9: Servo Corona DS329MG

Fonte: <http://www.hobbyking.com/hobbyking/store/__19958___ Corona_DS329MG_Digital_Metal_Gear_Servo_3_8kg_32g_0_11s.html >

De acordo com [42], virtualmente todo equipamento eletrônico requer uma fonte de alimentação de corrente contínua, seja ela uma bateria ou fonte comum. Servos não fogem a regra e também necessitam de uma fonte desse tipo.

Segundo as especificações dos servos em [43] e [44], as correntes de operação do BMS-620MG e do DS329MG são respectivamente 740mA e 400mA em 6V. A tensão de 6V é comum em baterias para hobbystas, mas incomum para a maioria das fontes. Em 5V (pouco acima dos 4,8V das especificações) os servos já fornecem bom torque e portanto escolheu-se 5V como a tensão de alimentação.

2.3.10 Fonte de alimentação dos servos – Fonte ATX 450W

Cada uma das 6 patas possui 2xBMS-620MG e 1xDS329MG, o que dá uma estimativa de 1880mA em operação para cada pata e um total de 11,28A considerando-se as 6 patas. Especificou-se no projeto que a alimentação poderia ser feita por cabo, o que viabiliza a escolha de uma fonte comum em detrimento de uma bateria com capacidade suficiente e que necessitaria um estudo mais apurado. Pelo fato de já se possuir o componente, escolheu-se uma fonte ATX de 450W, em pico, para alimentar a parte de potência. A fonte escolhida fornece até 22A na linha 5V, o que dá uma margem bastante segura de operação considerando as estimativas prévias.

2.3.11 Estação Base

A estação base é qualquer computador que possua os seguintes requisitos: ao menos uma porta USB, funcione no sistema operacional Windows® XP (ou superior), consiga executar uma máquina virtual Java e possua um mouse, um teclado e monitor.

O computador da estação base, através de um software que implemente um protocolo simples a ser desenvolvido, deve enviar comandos ao robô através do módulo ZigBee ligado à porta USB. O módulo ZigBee realizará a comunicação com a FPGA, que deve, através do mesmo protocolo, interpretar os comandos e executar a ação correspondente ao comando enviando sinais para os motores ou enviando uma leitura para o computador.

2.4 Parte teórica

2.4.1 Cinemática inversa

Segundo Craig[45], cinemática é a ciência do movimento que ignora as forças que causam o movimento. Dentro da ciência da cinemática, estuda-se posição, velocidade, aceleração e as derivações das variáveis de posição. No campo da robótica, a descrição cinemática de um manipulador consiste em saber a posição da ferramenta a partir de um sistema de coordenadas base, como pode ser visto na figura 10.

Cada junta do robô tem um sistema de coordenadas correspondente. Os sistemas de coordenadas são mapeados de forma relativa ao anterior. A partir do momento que se tem todos os sistemas de coordenadas do robô é possível obter o posicionamento de um sistema com relação a qualquer outro. Matematicamente a definição de um sistema de coordenadas a partir de outro é feita através de uma matriz de transformação linear, como a da figura 11. Essa matriz, homogênea, engloba um rotacional, para representar as diferenças de ângulo entre os eixos de cada sistema, e um vetor posição, que vai da origem de um sistema a origem do outro. A matriz serve como operador para transformar vetores de um sistema em outro.

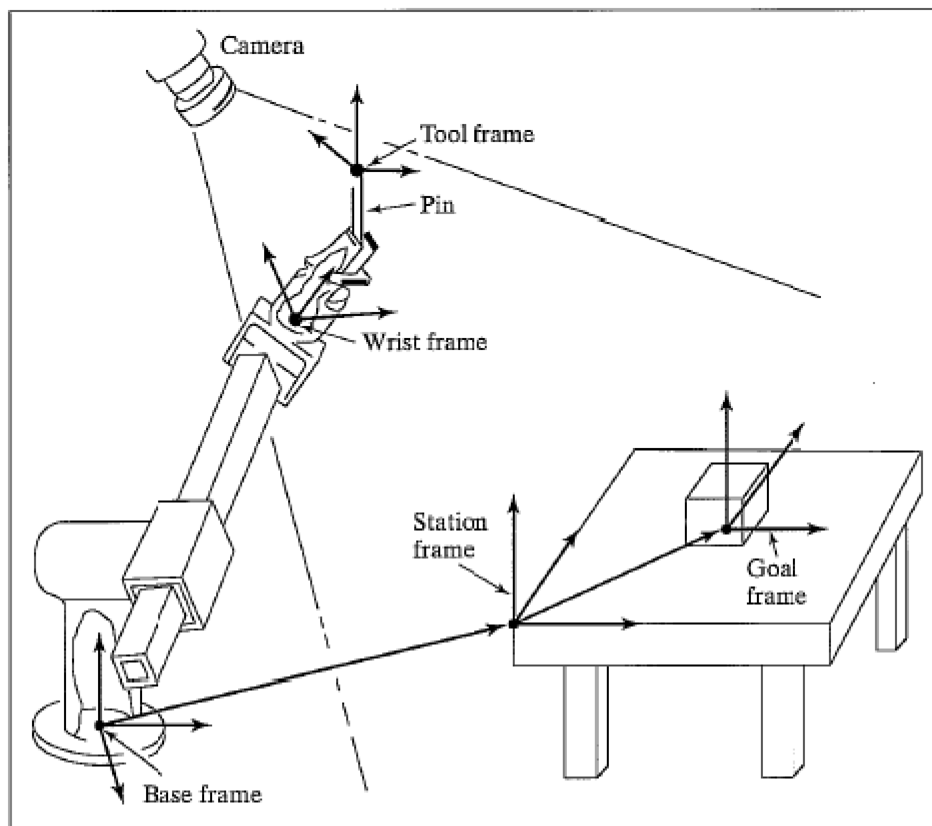


Figura 10: Exemplo de sistemas de coordenadas em robôs.
Fonte: Imagem retirada de [45].

$$\begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & {}^{A}_{B}R & & {}^{A}P_{BORG} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix}.$$

Figura 11: Transformação de um vetor posição P do sistema A para o sistema B, usando matriz de transformação.
Fonte: Imagem retirada de [45]

Ter conhecimento das posições relativas de sistemas de coordenadas é uma ferramenta importante, mas por si só não resolve o problema. Distâncias entre eixos, por exemplo, são constantes pois podem ser medidas, mas a angulação de cada junta é sempre uma incógnita no sistema ao calcular a posição e a rotação da ferramenta. Para resolver essa questão existem duas abordagens distintas, utilizando a cinemática direta ou cinemática inversa. Com a cinemática direta calculam-se a posição e orientação da

ferramenta a partir dos ângulos das juntas do manipulador. Já com a cinemática inversa calculam-se os ângulos das juntas do manipulador a partir da posição e da orientação da ferramenta.

Segundo Buss[46] na introdução de seu artigo, a cinemática inversa é comumente utilizada quando se trata de movimentação de braços robóticos ou animação de esqueletos virtuais. A opção por essa abordagem se torna clara por ser muito mais simples fornecer a posição final desejada do que os ângulos de juntas que levam a essa posição. O professor Max Fischer[47] cita em seu material três possibilidades de solução da cinemática inversa. O problema pode ser resolvido de forma incremental, analítica ou geométrica.

A solução incremental é relativamente simples de ser obtida. Partindo-se da equação (1) onde "J" é o jacobiano, "x" é o vetor posição da ferramenta e "q" é um vetor contendo os ângulos de cada junta. A partir dessa fórmula é possível obter equação (2). Partindo-se de uma posição inicial para a qual tanto "x" e "q" são conhecidos, é possível chegar as coordenadas desejadas somando pequenos incrementos "dq" e "dx". No caso da solução analítica, a solução é obtida manipulando-se algebricamente as matrizes de transformação de sistemas de coordenadas. A solução geométrica necessita de uma análise gráfica do robô para encontrar relações entre os ângulos e a posição desejada.

$$J(q)=\frac{dq}{dx} \quad (1)$$

$$dx=J^{-1}(q)\,dq \quad (2)$$

Todas as soluções são válidas, têm suas vantagens e desvantagens. Deve-se analisar qual o tipo de robô que se está construindo, qual é a complexidade e, a partir disso, definir qual método é o mais adequado.

2.4.2 "Gait"

De acordo com o dicionário, *gait* refere-se ao modo de andar. No caso de um hexápode, existe a possibilidade de utilizar 4 tipos distintos de *gait*[48]. Para facilitar a visualização dos movimentos, adota-se a convenção apresentada na figura 12.

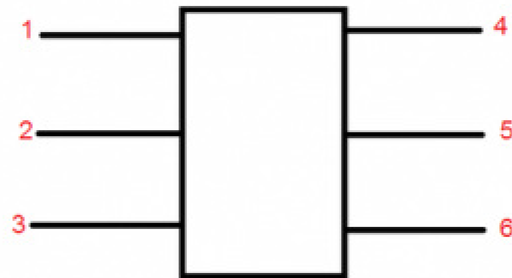


Figura 12: Convenção de numeração das patas.

Fonte: <http://hexapodrobots.weebly.com/uploads/8/5/3/4/8534558/1316341425.jpg>

Dentre as possíveis, é possível utilizar uma combinação 4+2, também conhecida como “quadruped gait”. Nessa combinação alternam-se 2 patas movendo-se para frente pelo ar, enquanto as outras 4 deslocam-se no chão em sentido contrário. Ao todo são necessárias três etapas para completar um ciclo de movimentação, já que são 3 pares de patas. É possível utilizar também 5+1, ou “one by one gait”. Nessa combinação uma pata move-se pelo ar, enquanto as outras 5 deslocam-se no chão em sentido contrário.

A terceira possibilidade é usar uma combinação 3+3, o “tripod gait”, em que se alternam 3 patas no ar e 3 patas no chão. Essa é a situação mais estável do ponto de vista estático e dinâmico, além de ser a mais conhecida quando se trata de modos de andar de hexápodes[49].

Uma última opção possível é o “free gait”. Nesse caso, o mais complexo, não há gait definido. Baseado no que o robô tem de percepção do ambiente, principalmente quanto ao tipo de terreno, o robô escolhe qual é o gait mais adequado para transpor a situação.

A figura 13 permite uma visualização simplificada do mecanismo de movimentação. Nela são mostradas as situações de pata para pata ao longo do tempo, representadas com 1 se estão no ar e 0 se estão no chão.

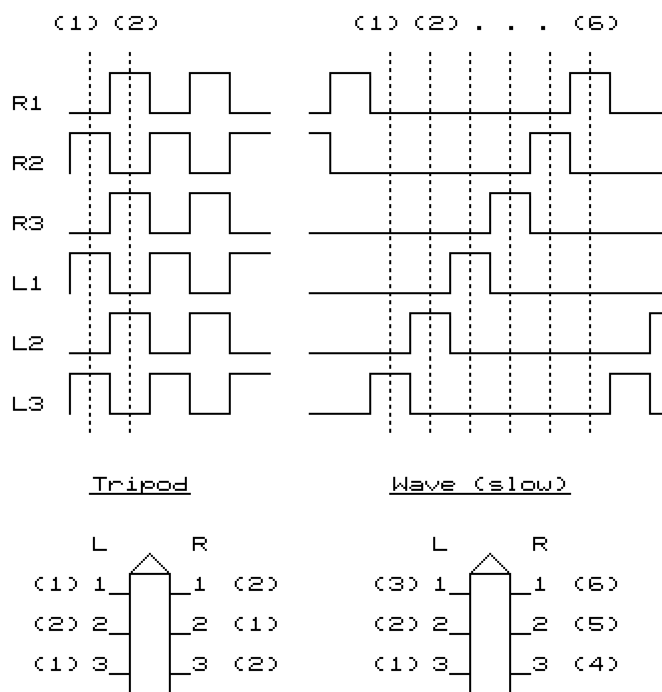


Figura 13: Esquemas de movimentação.

Fonte: <http://www.oricomtech.com/projects/cynthia2.gif>

2.4.3 Desenvolvimento de um servomotor próprio

Uma das ideias inicias do projeto era desenvolver um servomotor próprio a partir de um motor DC comum. Como descrito na página do 2 de [50], algumas etapas necessárias para essa realização, além da escolha de um motor DC adequado, seriam projetar um sistema de controle, calibrar a malha de controle com realimentação, projetar e construir a mecânica de redução, sem deixar de mencionar o uso de uma eletrônica de potência.

Com o objetivo de evitar boa parte das tarefas que não são da competência da área de computação e simultaneamente acatando sugestão do orientador, fez-se a opção por utilizar um servomecanismo já pronto e reprojetar sua eletrônica, o que inclui refazer a malha de controle e ter de adicionar uma eletrônica de potência para conduzir o motor. As questões de cunho mecânico puderam ser ignoradas dessa forma. A figura 14 contém um diagrama macro do servomotor próprio.

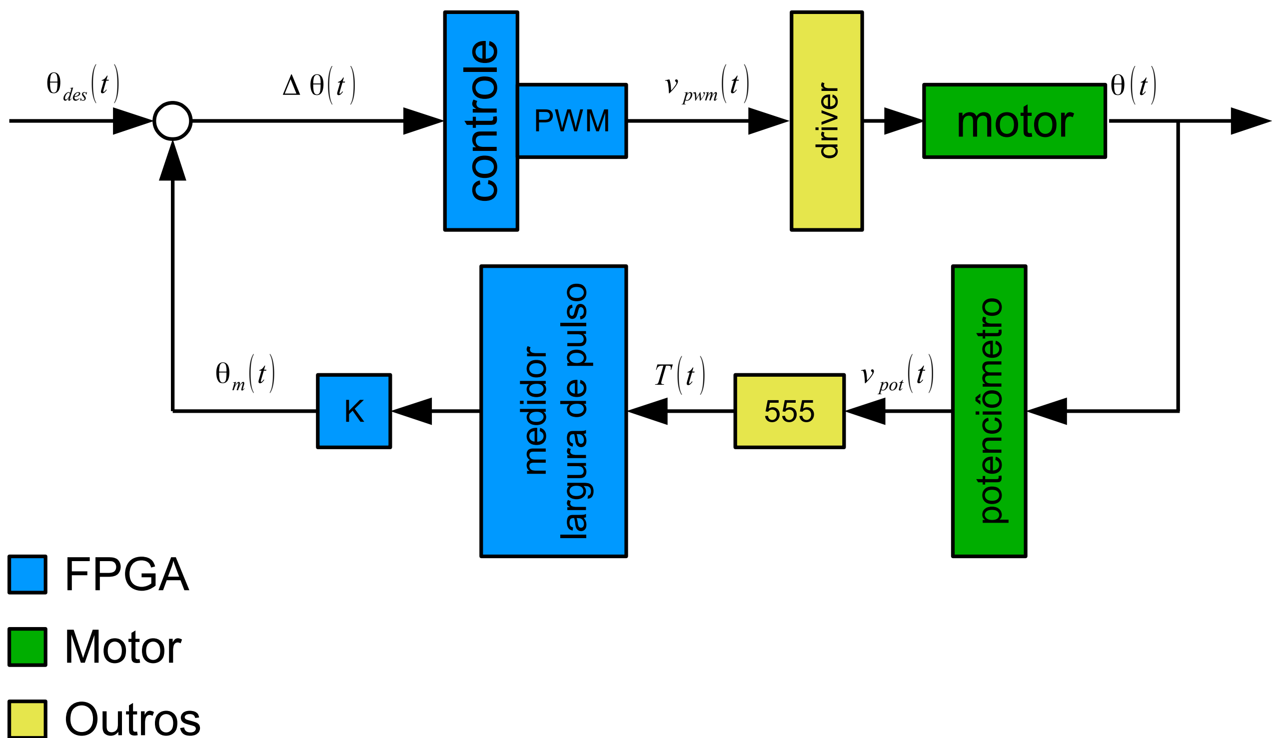


Figura 14: Diagrama do sistema de controle do servomotor próprio.

Fonte: Autoria própria

Onde:

- θ_{des} : ângulo desejado
- θ_m : ângulo medido
- θ : ângulo real
- v_{pwm} : tensão (média) do PWM
- v_{pot} : tensão de saída do potenciômetro
- T : largura do pulso gerado pelo timer 555

As conexões mecânicas e caixa de redução ficam implícitas no bloco do motor. Os componentes externos necessários são um *driver* para o motor, já que a saída da FPGA não fornece corrente e tensões suficientes, e um *timer*, no caso o comum LM555. Seriam necessários ao todo 18 motores semelhantes a esse e o kit de FPGA selecionado possui um ADC com 8 canais, insuficiente para realizar a leitura de todos os potenciômetros. Para contornar esse problema foi utilizado um temporizador para cada motor gerando um sinal digital de frequência fixa e com largura de pulso variável dependendo da resistência do potenciômetro. Dessa maneira faz-se uma versão simplificada de um ADC.

2.4.4 I2C

O barramento I2C foi projetado pela Philips no começo dos anos 80 para permitir comunicação simples entre componentes de uma mesma placa. A partir de 2006 a divisão de semicondutores da Philips faz parte da NXP[51]. As velocidades suportadas vão desde o padrão inicial *standard* de 100kbps até *high speed* de 3.4Mbps[52]. A comunicação é realizada por somente 2 pinos, SDA e SCL, correspondentes respectivamente às linhas de Serial Data e Serial Clock. Na figura 15 é possível um exemplo de implementação I2C.

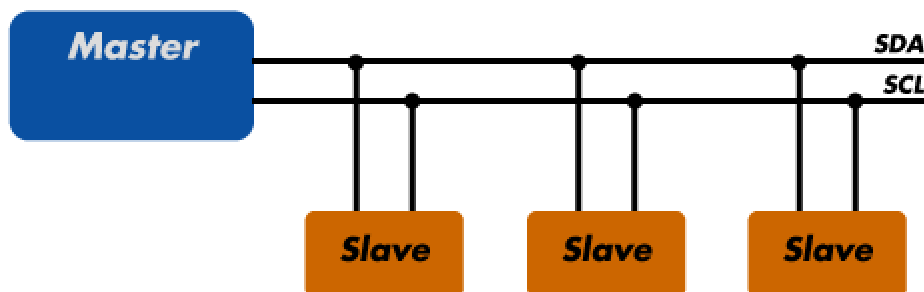


Figura 15: "Sample I2C Implementation."

Fonte: <http://www.totalphase.com/image/articles/i2c-diagram.png>

Um barramento I2C contém sempre no mínimo um *master*, responsável por iniciar as trocas de dados, e até 127 *slaves*, cada qual com seu endereço de 7 ou 10 bits. Os *slaves* não podem por si só enviar dados no barramento e portanto, sempre esperam pela iniciativa de um *master* para que haja comunicação[53]. O fato de utilizar apenas 2 fios para comunicar-se com inúmeros dispositivos é a maior vantagem desse tipo de comunicação. No entanto, isso causa um grande *overhead* ao ter de escrever o endereço do *slave* no barramento toda vez que se referir a ele e aguardar por ACKs, que confirmam o recebimento de mensagens. Esse problema pode ser visualizado no exemplo a seguir e na figura 16 a ele referente.

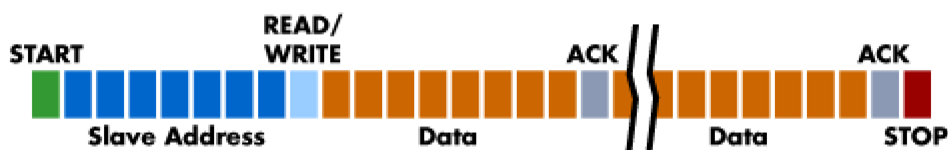


Figura 16: "I2C Communication Protocol".

Fonte: <http://www.totalphase.com/image/articles/i2c-protocol.png>

A comunicação se dá pela seguinte sequência de eventos, como na seção "Theory of Operation" de [52]:

1. *Master* coloca o barramento em *Start*. A partir desse momento todos os *slaves* aguardam.
2. Em seguida um dos *slaves* é endereçado, seguido por uma *flag* de escrita ou leitura.
3. O *slave* endereçado envia um ACK ao *master* confirmando a comunicação.
4. A comunicação procede alternando 1 byte enviado pelo *master* e 1 bit de ACK enviado pelo *slave*.
5. Quando o *master* achar conveniente, envia 1 bit de *Stop* e encerra a comunicação.

2.4.5 Alocação dinâmica, alocação estática e *object pooling*

A alocação dinâmica de memória é um elemento fundamental em sistemas computacionais desde a década de 1960[54]. É comum empregar alocação dinâmica quando se utilizam estruturas de dados complexas, pois ela dá ao programador grande poder expressivo[55]. Através dela é possível criar objetos quando necessário e eliminá-los quando deixam de ser utilizados, então é especialmente adequada quando certas entidades (objetos) têm um tempo de vida útil menor que a aplicação.

Apesar de amplamente utilizada, a alocação dinâmica pode ter comportamento não-determinista, e seu mau uso pode levar a vazamentos de memória, inconsistência de dados e exaustão de memória[56]. Certos autores consideram a alocação dinâmica um problema insolúvel[54]. Ainda assim, linguagens de alto nível amplamente utilizadas como C#, VB.Net e Java empregam alocação quase integralmente dinâmica[15,57].

O oposto de alocação dinâmica é a alocação estática, na qual a criação dos objetos ocorre em tempo de compilação (a inicialização ocorre em tempo de execução). Com alocação estática, não são criados ou apagados objetos na *heap* durante a execução[58], mas é possível criar objetos de escopo local na pilha. Este paradigma é muito utilizado em sistemas de tempo real. Nele, todos os objetos têm um escopo e um tempo de vida bem conhecidos, evitando o não-determinismo e o *overhead* da alocação dinâmica[59]. Por outro lado, é preciso criar explicitamente todos os objetos antes de utilizá-los e não à medida que são necessários[58], o que a torna menos adequada para implementar estruturas de dados mais complexas[55].

Para combinar as vantagens de alocação dinâmica e alocação estática, existe o paradigma de “object pooling”. Tal paradigma é utilizado amplamente (especialmente em jogos) com a finalidade de otimizar código[60]. Ele envolve a utilização de *pools* (piscinas) de objetos pré-alocados, das quais são obtidos objetos quando normalmente seriam alocados novos, e à qual eles são devolvidos quando normalmente seriam apagados[61]. *Pools* de objetos podem ser implementadas utilizando alocação dinâmica ou alocação estática.

Em *pools* de objetos implementadas utilizando alocação dinâmica, os objetos são criados na própria *heap*, então ainda podem ser criados novos objetos e apagados os antigos, se for necessário. Seu uso minimiza, mas não evita os problemas decorrentes de usar alocação dinâmica, já que a *heap* continua sendo utilizada. Em linguagens como Java e C#, somente este método pode ser utilizado para implementar *pools* de objetos. Como continua havendo alocação na *heap*, ele continua não sendo adequado para sistemas embarcados.

Já em *pools* de objetos implementadas por alocação estática, os objetos são todos declarados estaticamente e de alguma forma marcados como “utilizados” e “livres” à medida que são obtidos ou devolvidos à pool. Seu emprego tem uma desvantagem em relação à alocação dinâmica convencional: todos os objetos ocupam espaço da memória mesmo quando não são utilizados. Em contrapartida, o custo computacional de criação de objetos é consideravelmente reduzido. Em resumo, o emprego das *pools* implementadas por alocação estática tem a vantagem de preservar a semântica da alocação dinâmica, em que há controle sobre o tempo de vida dos objetos sem perder as vantagens (e as desvantagens) da alocação estática.

2.4.6 Garbage collection e reference counting

Programas desenvolvidos em linguagens não-gerenciadas podem conter erros em relação à alocação dinâmica de memória[62], que podem causar dois tipos distintos de problemas: vazamentos de memória (*memory leaks*) e ponteiros soltos (*dangling pointers*). Um vazamento de memória ocorre quando um bloco de memória não é apagado quando deixa de ser utilizado, de forma que o bloco nunca é liberado. *Dangling pointers* são ponteiros não-inicializados ou de objetos que já foram apagados, de forma que apontam para regiões inválidas de memória[63].

Para evitar estes problemas, é costumeiro empregar convenções rígidas quanto à ordem de alocação e apagamento de objetos, ou utilizar algum sistema para

gerenciamento automático de memória. Sistemas mais avançados podem acompanhar *garbage collectors*, que são ferramentas responsáveis por analisar a memória e determinar quais objetos não são mais utilizados e podem ser apagados. O programa deve parar de ser executado parcialmente ou totalmente (dependendo da implementação) enquanto um *garbage collector* estiver analisando a memória, do contrário a análise pode ficar inconsistente. As máquinas virtuais Java (JVMs) utilizam-se de *garbage collection* para gerenciamento da memória. Existem *garbage collectors* para as linguagens C e C++, mas são bastante limitados, e seu emprego requer que algumas normas de programação sejam seguidas.

Uma alternativa mais rudimentar e leve aos *garbage collectors* é o mecanismo de contagem de referências. Como o nome sugere, este mecanismo envolve manter um contador indicando em quantos lugares cada objeto é referenciado. O contador atingir valor zero é uma indicação de que o objeto não é mais referenciado, e então pode ser apagado. A contagem de referências não funciona bem quando há referências circulares, pois já que o contador de alguns objetos não atinge zero, eles nunca são apagados. O problema das referências circulares pode ser resolvido através de referências fracas nos lugares certos do código. Uma referência fraca é ignorada pelo contador de referências, mas é diferente de um ponteiro convencional. Um ponteiro convencional se tornaria um *dangling pointer* caso o objeto fosse apagado, enquanto as referências fracas passam a apontar para nulo quando isto ocorre.

2.5 Convenções de programação

A utilização de uma convenção consistente de programação melhora o processo de desenvolvimento e manutenção de código[64], pois facilita seu entendimento[65] e ajuda a evitar alguns erros comuns de programação[66]. Os quatro artefatos de software do projeto incluem códigos em linguagem Java, C++ e alguns trechos em linguagem C. Devido à complexidade do projeto, criou-se e adotou-se uma série de convenções de programação.

Já existem diversas convenções para as três linguagens, as quais diferem grandemente de uma linguagem para a outra. Para facilitar o desenvolvimento, dentro do razoável, buscou-se a utilização de uma convenção consistente de programação nas três linguagens. A convenção adotada é de autoria própria, mas incorpora características encontradas nas Java Coding Conventions[65], nos GNU Coding Standards[67] e no Google C++ Style Guide[68].

As regras são divididas em três grupos. O primeiro grupo contém regras de nomeação de funções, classes, entre outros. O segundo grupo contém regras gerais referentes à codificação em si. Finalmente, o terceiro grupo determina como o código deve ser formatado em termos de espaçamento, identação, entre outros.

2.5.1 Regras de nomeação

1. Pacotes em Java e namespaces em C++ devem ser escritos inteiramente com letras minúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente deve iniciar com letra maiúscula, com as demais letras minúsculas (camelCase).
2. Classes em Java e C++, estruturas em C e C++, interfaces em Java e enumerações em Java, C e C++ devem começar com letras maiúsculas, com as demais letras minúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente também deve iniciar com letra maiúscula, com as demais letras minúsculas (PascalCase).
3. Valores de enumeração em Java, C++ e C devem começar com letras maiúsculas, com as demais letras minúsculas Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente também deve iniciar com letra maiúscula, com as demais letras minúsculas (PascalCase).
4. Constantes em Java, C e C++ devem ser escritas inteiramente com letras maiúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente será precedida por um "underline" e também deve ser escrita inteiramente com letras maiúsculas (GNU standard).
5. Métodos em Java e C++ e funções em C++ devem ser escritos inteiramente com letras minúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente deve iniciar com letra maiúscula, com as demais letras minúsculas (camelCase).
6. Funções em C devem ser escritas inteiramente com letras minúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente será precedida por um "underline" e também deve ser escrita inteiramente com letras minúsculas (GNU standard).
7. Variáveis e parâmetros em Java e C++ devem ser escritos inteiramente com letras minúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra

subsequente deve iniciar com letra maiúscula, com as demais letras minúsculas (camelCase).

8. Variáveis e parâmetros em C devem ser escritas inteiramente com letras minúsculas. Se os nomes forem compostos de mais de uma palavra, cada palavra subsequente será precedida por um “underline” e também deve ser escrita inteiramente com letras minúsculas (GNU standard).
9. Atributos de classes e objetos em Java e C++ devem utilizar a mesma convenção para nomenclatura de variáveis, mas precedidos de “m_” (“meu”).

2.5.2 Regras de codificação

1. Todos os blocos (funções, laços, condicionais, etc) devem ser envolvidos por chaves, mesmo que sejam de uma única linha.
2. Todas as invocações de métodos de instância da própria classe devem ser empregar explicitamente a palavra-chave “this” para diferenciar da invocação de funções livres e métodos estáticos.
3. Todas as invocações de métodos estáticos da própria classe devem empregar explicitamente o nome da classe para diferenciar da invocação de funções livres e métodos de instância.
4. Em C e C++, as inclusões de arquivos de cabeçalho devem ser sempre em relação ao diretório raiz do projeto. Caminhos relativos são mais passíveis de erros e dificultam a localização dos arquivos.
5. Em Java, C++ e C deve-se evitar a chamada de métodos específicos da arquitetura a partir do código de alto nível, de forma ao código ser portável.

2.5.3 Regras de estilo

1. Em C e C++, o colchete de abertura de bloco deve estar sempre na linha seguinte da declaração do bloco.
2. Em Java, o colchete de abertura de bloco deve estar sempre na mesma linha da declaração do bloco.
3. O conteúdo de todo bloco deve sofrer identação. Essa regra inclui os *namespaces* em C++.

2.6 Considerações

Há diversos robôs hexápodes disponíveis no mercado, mas não foi encontrado nenhum que combine projetos de software, hardware e mecânica abertos ou bem documentados e alto grau de sofisticação. Dessa forma, o hexápode descrito neste projeto representa um grande avanço no sentido de um robô hexápode aberto.

Além disso, o projeto representa uma boa plataforma de aplicação dos conhecimentos adquiridos no curso de engenharia de computação. Desde conhecimentos de matemática, geometria e física, aplicados na cinemática inversa até conhecimentos de eletrônica e programação aplicados na FPGA, o robô envolve diversas áreas da engenharia.

3 Desenvolvimento

3.1 Hardware na FPGA

A figura 48 apresenta um diagrama de blocos resumido, o qual contém o hardware que se encontra na FPGA e os periféricos com os quais esses elementos de hardware se comunicam.

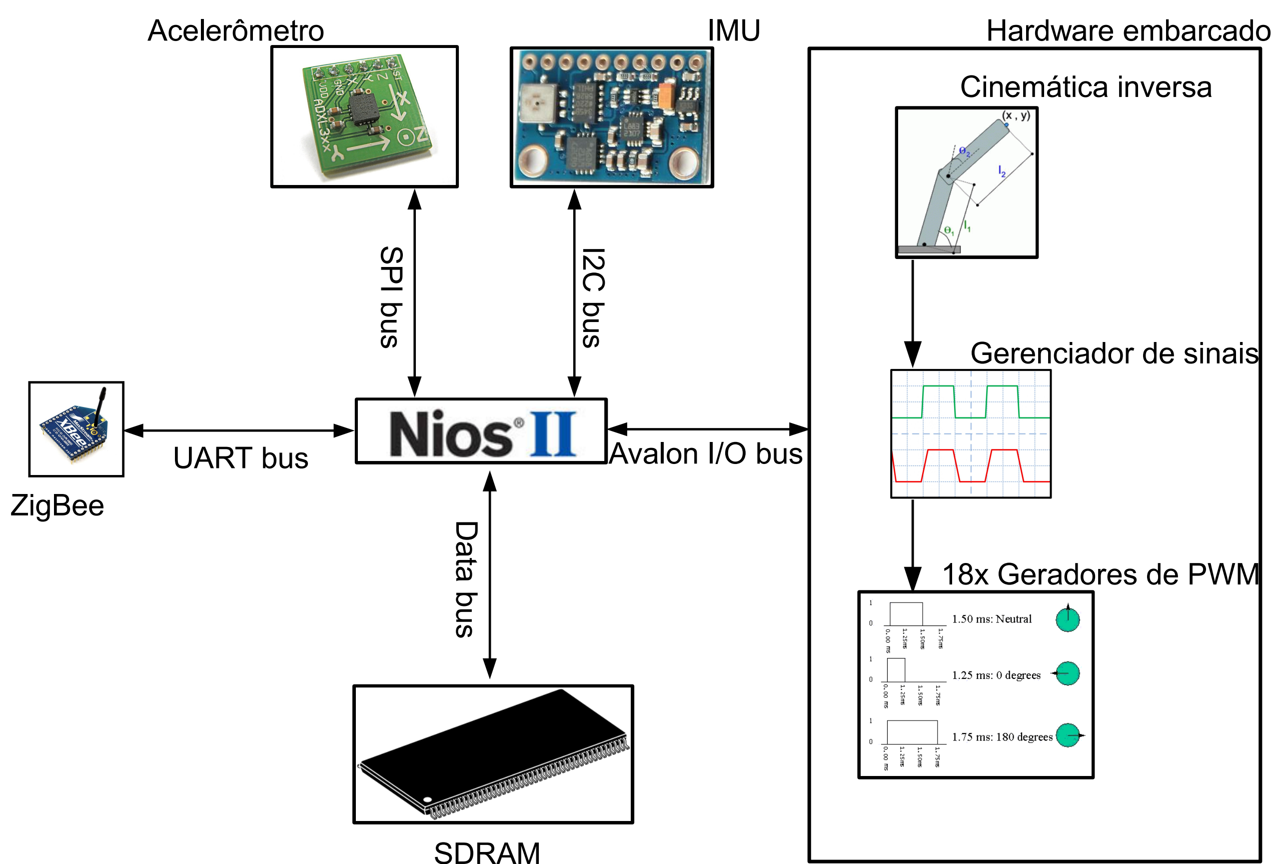


Figura 17: Diagrama de blocos do hardware embarcado e periféricos.

Fonte: Autoria própria.

3.1.1 NIOS – Microcontrolador Embarcado

O bloco Nios inclui o processador, o qual roda os módulos descritos na seção firmware, e outros periféricos de integração. Todo o sistema é montado com auxílio da ferramenta SOPC Builder[69], a começar pela inclusão de um processador Nios II. Das 3 opções disponíveis na tela de inclusão 18, escolheu-se a opção Nios II/e. É o mais

simples dos três, mas é o que requer menos unidades lógicas. Os cálculos mais pesados, como a cinemática inversa, foram implementados como blocos de hardware avulsos e dessa forma não há necessidade de alto poder de processamento do Nios.

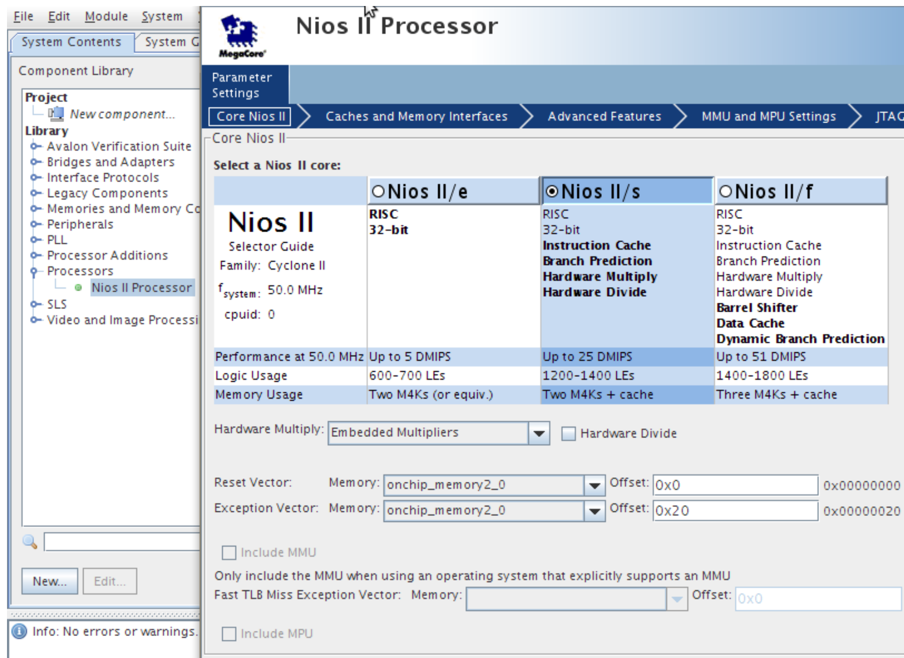


Figura 18: Opções de processador Nios disponíveis no SOPC.

Fonte: <http://fpga4u.epfl.ch/images/b/be/NiosIIs.jpg>

Além da escolha do processador na figura 18, deve-se mencionar a escolha das memórias que serão utilizadas pelo processador. No exemplo dado é utilizada memória *onchip*, composta por unidades lógicas da própria FPGA, o que é altamente benéfico do ponto de vista desempenho, mas também bastante limitado quando se fala em capacidade[70]. Somando-se o *firmware* desenvolvido para o Nios acompanhado de um RTOS e lógica desenvolvida em VHDL, é necessário mais memória. Seguindo sugestões da própria Altera e do recurso oferecido pelo kit FPGA utilizado, fez-se uso da SDRAM de 32MB disponível. Embora seja mais lenta e necessite de uma controladora dedicada para operá-la, oferece espaço mais do que suficiente para as rotinas necessárias no projeto.

A controladora de SDRAM tem algumas peculiaridades na configuração. Não existe configuração padrão do SOPC para o modelo de SDRAM. Para que funcione corretamente, deve-se criar uma configuração customizada e utilizar os parâmetros fornecidos no tutorial[71]. As duas telas de configuração da controladora ficam como as figuras 19 e 20. Além de configurar a controladora, a SDRAM precisa de um sinal de clock deslocado no tempo para que os dados estejam estáveis. A geração desse sinal é

possível utilizando uma PLL para o clock, com sinal de saída de formato igual ao da entrada deslocado -3ns para um clock de 50MHz. Como o clock utilizado é de 100MHz, utilizou-se o equivalente em graus para o deslocamento, cujo valor é -54º e independe da frequência. A tela de configuração da PLL pode ser vista na figura 21.

Figura 19: Configurações da controladora SDRAM, página 1.

Fonte: Autoria própria.

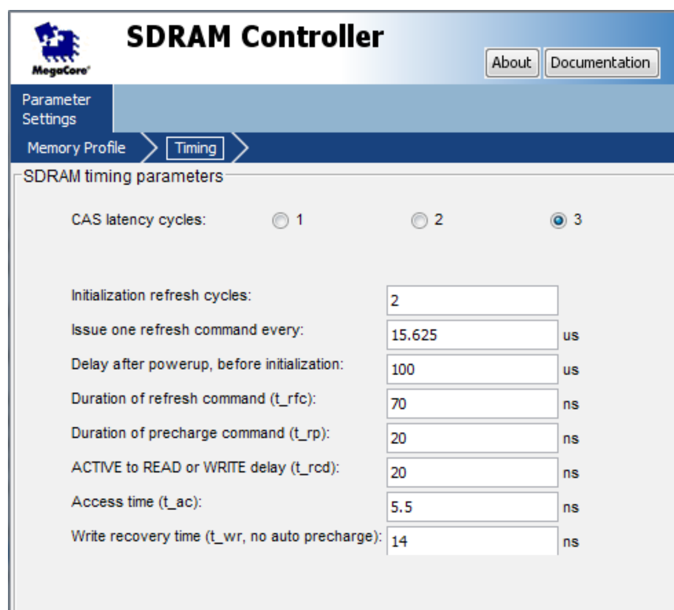


Figura 20: Configurações da controladora SDRAM, página 2.

Fonte: Autoria própria.

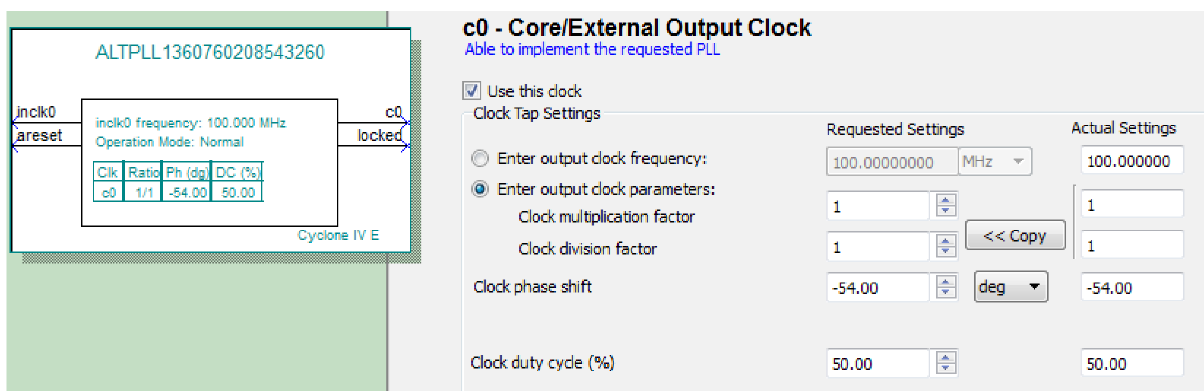


Figura 21: Configuração da PLL da SDRAM.

Fonte: Autoria própria.

O módulo Xbee conectado a FPGA é tratado como um dispositivo UART comum. Para estabelecer a comunicação com dispositivos desse tipo, é possível adicionar uma controladora UART que já faz parte do SOPC Builder. As configurações da controladora só precisam ser similares às configurações dos módulos Xbee para que tudo funcione corretamente. Como pode ser visto na figura 22, os únicos cuidados a serem tomados são não usar paridade e usar 8 bits de dados acompanhado de um *stop* bit. O *baud rate* utilizado é de 57600 bits por segundo. Embora o máximo seja 115200 bits, essa opção se tornou inviável devido a alta taxa de erros apresentada. Além dessa UART, é utilizada

uma JTAG UART para estabelecer comunicação entre o Nios e o PC para auxiliar no processo de *debug* durante o desenvolvimento.

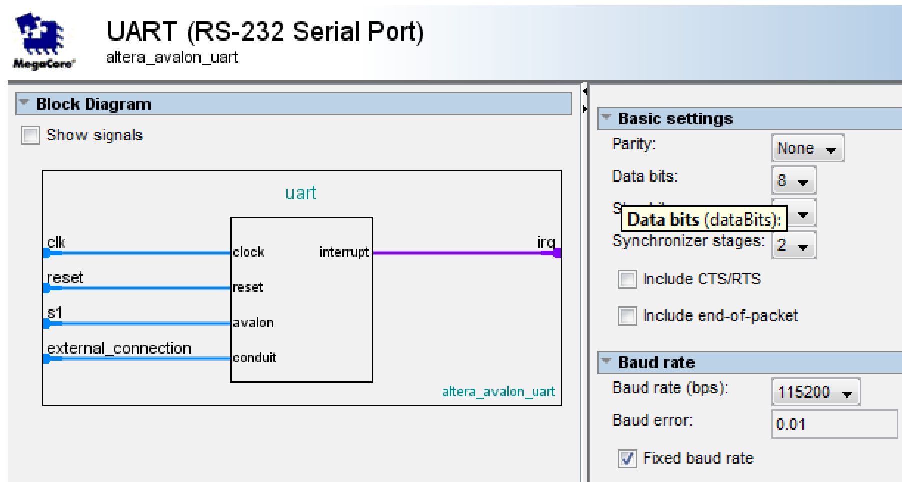


Figura 22: Configuração da UART.

Fonte: Autoria própria.

A comunicação entre Nios e sensores é feita pela controladora I2C descrita na seção 3.1.6. Ela não faz parte da biblioteca padrão do SOPC e deve ser adicionada. O pacote disponível em [72] já foi adaptado a interface Avalon e é completamente funcional após ser adicionado, sem que haja necessidade de configurações adicionais.

Os componentes restantes são dispositivos de I/O paralelo, utilizados para interfacear o Nios com o resto do sistema desenvolvido. A figura 23 mostra as controladoras utilizadas, todas com o prefixo “pio_bot” por interfacearem com o bloco responsável pela movimentação do robô. As três primeiras (pio_bot_x, pio_bot_y e pio_bot_z) possuem 16 bits cada e são responsáveis pela escrita de coordenadas cartesianas sinalizadas. Os 3 bits de “pio_bot_legselect” selecionam a qual pata as coordenadas correspondem e o bit “pio_bot_wrcoord” indica que os dados devem ser escritos. O bit “pio_bot_reset” é utilizado para garantir que o cálculo da cinemática inversa só seja efetuado quando o Nios garantir que as três coordenadas cartesianas são válidas e “pio_bot_endcalc” indica que o cálculo foi finalizado. Através do bit “pio_bot_updateflag” indica-se que as saídas, que são os sinais PWM direcionados aos servos, devem ser atualizadas.

Target

Device Family: Cyclone IV E

Clock Settings

Name	Source	MHz
clk_100	External	100.0
altpll_sdram_c0	altpll_sdram.c0	100.0

Use	Conn...	Name	Description	Clock	Base	End	IRQ	Tags
☑		⊟ cpu	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	clk_100				
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x04000800	0x04000fff		
☑		⊟ jtag_uart	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_100	0x04001180	0x04001187	0	
☑		⊟ pio_led	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x040010e0	0x040010ef		
☑		⊟ spi	SPI (3 Wire Serial)	[clk]				
		spi_control_port	Avalon Memory Mapped Slave	clk_100	0x04001080	0x0400109f	1	
☑		⊟ TERASIC_SPI_3WIRE_0	TERASIC_SPI_3WIRE	[clock_reset]				
		slave	Avalon Memory Mapped Slave	clk_100	0x04001000	0x0400103f		
☑		⊟ altpll_sdram	Avalon ALTPLL	altpll_sdram_...				
		pll_slave	Avalon Memory Mapped Slave	clk_100	0x040010f0	0x040010ff		
☑		⊟ sdram	SDRAM Controller	[clk]				
		s1	Avalon Memory Mapped Slave	altpll_sdra...	0x02000000	0x03ffffff		cpu.data_master
☑		⊟ timer_sys	Interval Timer	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x040010a0	0x040010bf	3	
☑		⊟ pio_bot_x	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001100	0x0400110f		
☑		⊟ pio_bot_y	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001110	0x0400111f		
☑		⊟ pio_bot_z	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001120	0x0400112f		
☑		⊟ pio_bot_reset	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001140	0x0400114f		
☑		⊟ pio_bot_endcalc	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001130	0x0400113f		
☑		⊟ pio_bot_legselect	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001150	0x0400115f		
☑		⊟ pio_bot_updateflag	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001160	0x0400116f		
☑		⊟ pio_bot_wrcoord	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001170	0x0400117f		
☑		⊟ I2C_Master	I2C Master (opencores.org)	[clock]				
		avalon_slave	Avalon Memory Mapped Slave	clk_100	0x040010c0	0x040010df	4	
☑		⊟ uart	FIFOed UART (RS-232 serial port)9.3.0	[s1_clock]				
		s1	Avalon Memory Mapped Slave	clk_100	0x04001040	0x0400107f	5	

Figura 23: Sistema SOPC completo.

Fonte: Autoria própria.

3.1.2 Cinemática inversa

Dentre as opções de cinemática inversa apresentadas anteriormente, optou-se por utilizar a solução geométrica. Tratando-se cada pata do robô como um sistema independente, fica relativamente simples resolver de maneira geométrica porque existem 3 juntas e 2 delas estão posicionadas no mesmo lugar, sem que haja um eixo entre elas. A figura 24 facilita a análise.

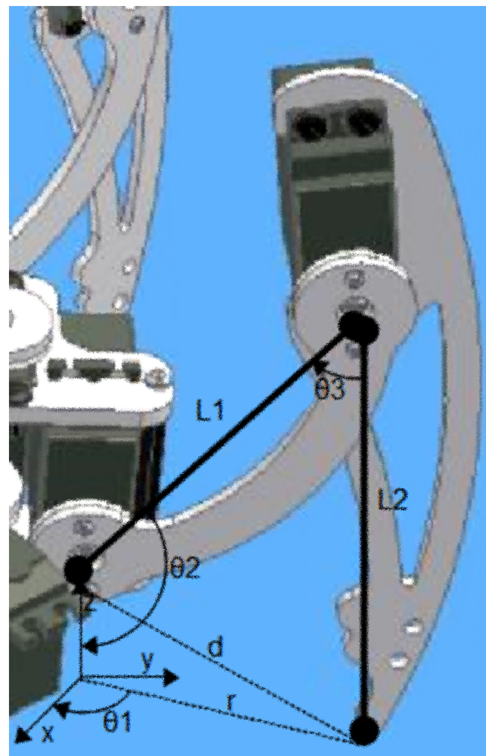


Figura 24: Geometria de uma pata em detalhe.

Fonte: Autoria própria.

As incógnitas são os três ângulos θ1, θ2 e θ3. θ1 está situado somente no plano xy e pode ser calculado através do arco tangente, como mostra a equação (3).

$$\theta 1 = \arctan(y/x) \quad (3)$$

θ2 necessita de uma decomposição para ser calculado mais facilmente. Destacando-se o triângulo de arestas z, d e r, o ângulo entre as arestas z e d é o primeiro ângulo parcial de θ2 (α), onde d e r são definidos pelas equações (4) e (5). Destacando-se o triângulo de arestas L1, L2 e d, o ângulo entre as arestas L1 e d é o segundo ângulo parcial que compõe θ2 (β). O primeiro ângulo parcial é definido pela equação (6), por arco-tangente, e o segundo pela equação (7), por lei dos cossenos. θ2 é então definido pela soma de α e β, equação (8).

$$r = \sqrt{x^2 + y^2} \quad (4)$$

$$d = \sqrt{r^2 + z^2} \quad (5)$$

$$\alpha = \arctan(r/z) \quad (6)$$

$$\beta = \arccos\left(\frac{L1^2 - L2^2 + d^2}{2\,L1\,d}\right) \quad (7)$$

$$\theta 2 = \alpha + \beta \quad (8)$$

O último ângulo θ3, pode ser calculado também utilizando lei dos cossenos. Tomando-se o triângulo formado pelas arestas L1, L2 e d como base se tem a equação (9).

$$\theta 3 = \arccos\left(\frac{L1^2 + L2^2 - d^2}{2\,L1\,L2}\right) \quad (9)$$

Definidas as equações dos ângulos das juntas, torna-se simples calcular quais os ângulos necessários para qualquer posição desejada para a extremidade da pata.

Com a finalidade de validar as equações encontradas antes da implementação prática do robô, utilizou-se o software MATLAB acompanhado do Robotics Toolbox[73]. O *toolbox* está sendo desenvolvido e melhorado há 15 anos, o que faz com que ele seja uma ferramenta bastante madura para auxiliar na simulação de robôs. Para o teste, primeiramente a pata robótica do hexápode teve de ser representada por uma estrutura robótica de operação similar de acordo com as possibilidades oferecidas pelo *toolbox*. As conexões e juntas foram descritas usando parâmetros segundo a notação Denavit-Hartenberg[74]. Em seguida foram definidas 3 localizações, as quais foram interpoladas linearmente para simular uma trajetória de uso. O comportamento observado, em imagens como a 25, era o esperado.

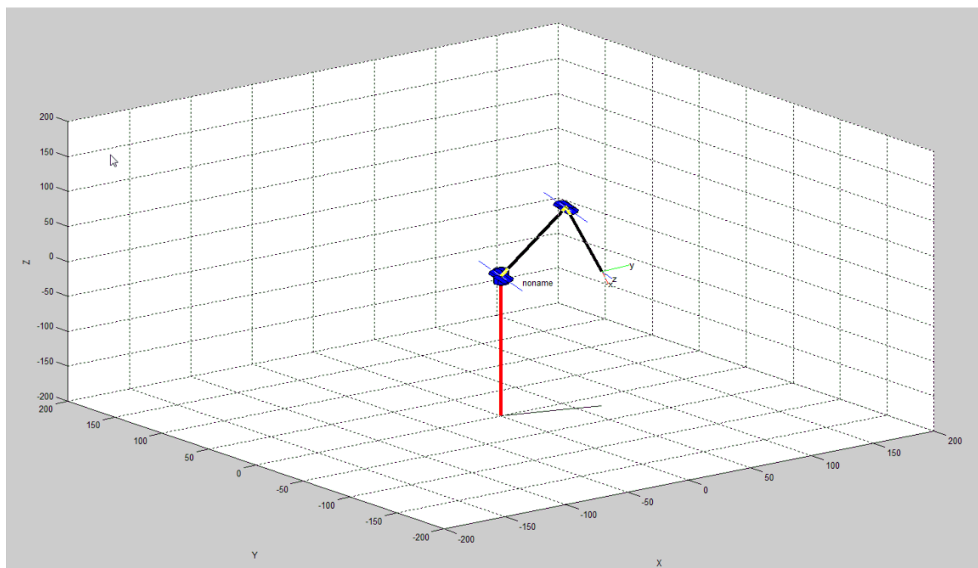


Figura 25: Screenshot durante simulação de trajetória.

Fonte: Autoria própria.

Após ter a simulação funcionando de forma satisfatória, o passo seguinte foi implementar a cinemática inversa de forma a ser utilizável no robô. Para que esse objetivo fosse completado, era possível tanto implementar a cinemática inversa em lógica reconfigurável diretamente ou em C, fazendo uso do Nios. Pelo foco do projeto ser o uso da FPGA, a primeira opção foi escolhida. Além da justificativa do foco, a matemática não ficou complexa a ponto de inviabilizar a implementação em VHDL. A operação de raiz quadrada foi retirada de[75] e apenas modificada para receber e retornar dados do tipo integer. No caso das operações de arco tangente e arco cosseno optou-se por utilizar valores tabelados, armazenados em memórias ROM. O bloco é externamente similar ao apresentado na figura 26.

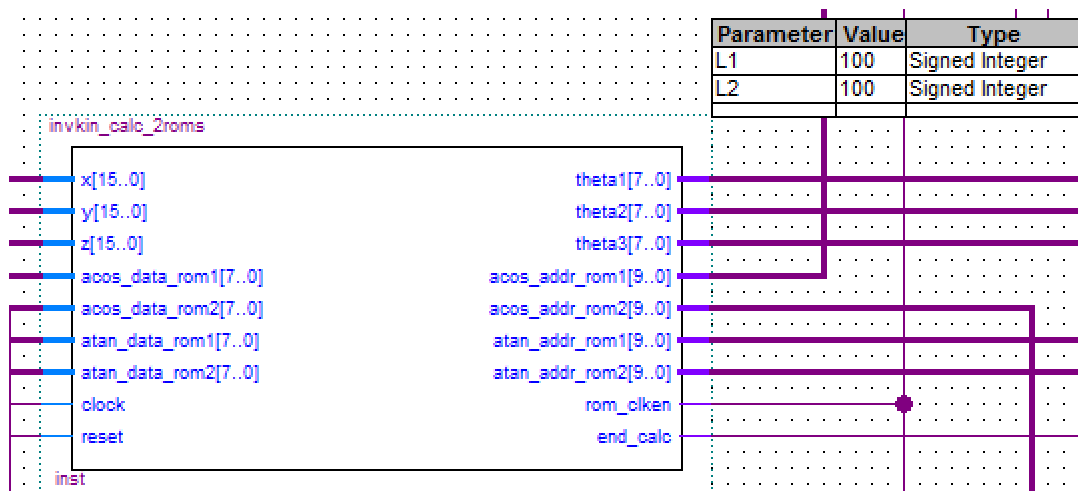


Figura 26: Bloco de cálculo da cinemática inversa.

Fonte: Autoria própria.

Pode-se notar que existem barramentos de dados e endereço redundantes. Uma ROM síncrona fornecida pela Altera necessita de 2 clocks para que haja uma leitura segura do dado desejado (1 clock para escrever o endereço, 1 clock para ler o barramento de dados). No sistema projetado há sobra de memória, então é conveniente ter memórias redundantes e diminuir o número de ciclos do cálculo. Vale ressaltar também que todos os dados são inteiros, sem ponto flutuante. Trabalhando-se com uma faixa de 180 graus de liberdade é tolerável ter +-1 grau de erro para diminuir consideravelmente a complexidade.

3.1.3 Conversor entre ângulo e largura de pulso

O cálculo da cinemática inversa, como apresentado anteriormente, recebe coordenadas cartesianas como parâmetros e fornece ângulos de juntas como saída. Para que um servo fique em uma determinada posição contudo, necessita-se de um sinal com frequência constante e largura de pulso variável.

Nessa etapa primeiramente deve-se encontrar a faixa de operação dos servos em termos de largura de pulso. Após várias informações equivocadas encontradas, o melhor resultado encontrado foi utilizando a faixa padrão dos servos de equipamentos de controle remoto, ou seja, entre 1 e 2ms. O servo fica centralizado com 1,5ms e atinge aproximadamente -40 e +40 graus nos dois extremos.

Como entrada, o bloco de conversão recebe um ângulo entre 0 e 180 graus. A faixa segura de operação dos servos é apenas metade disso, então é imposta uma limitação no hardware embarcado para que o ângulo utilizado na conversão seja no mínimo equivalente a -40º e não ultrapasse +40º. Limitado o ângulo, basta estabelecer uma relação linear de transformação entre as duas faixas. A relação fica então simples, como na equação (10), onde y é a largura do pulso e x o ângulo.

$$y=1000+(100*x)/18 \quad (10)$$

A aparência final do bloco pode ser vista na figura 27.

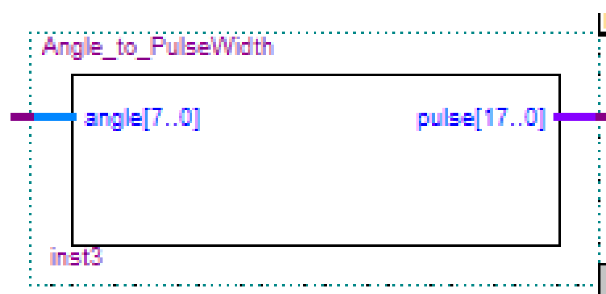


Figura 27: Bloco de conversão ângulo – largura de pulso.

Fonte: Autoria própria.

3.1.4 Gerador de sinal PWM

Os servos, como já dito anteriormente, têm sua posição controlada por um sinal PWM. Em português a sigla significa modulação por largura de pulso, e as propriedades básicas desse tipo de sinal podem ser adquiridas em [76]. A partir dos conceitos básicos, percebe-se que os parâmetros de um sinal PWM são frequência e largura de pulso. A

figura 28 mostra como esses parâmetros são interpretados no sinal e a figura 29, a aparência externa do bloco gerador de sinal.

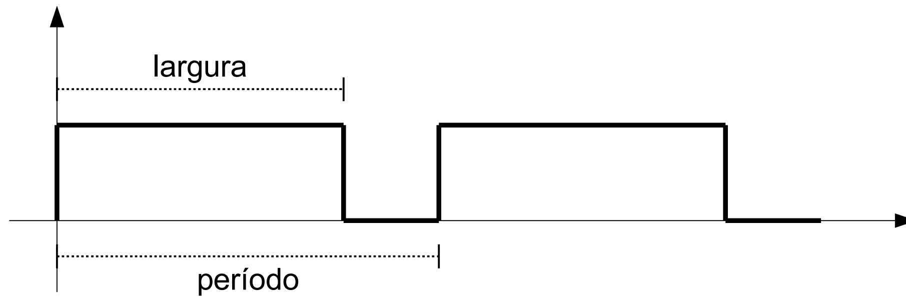
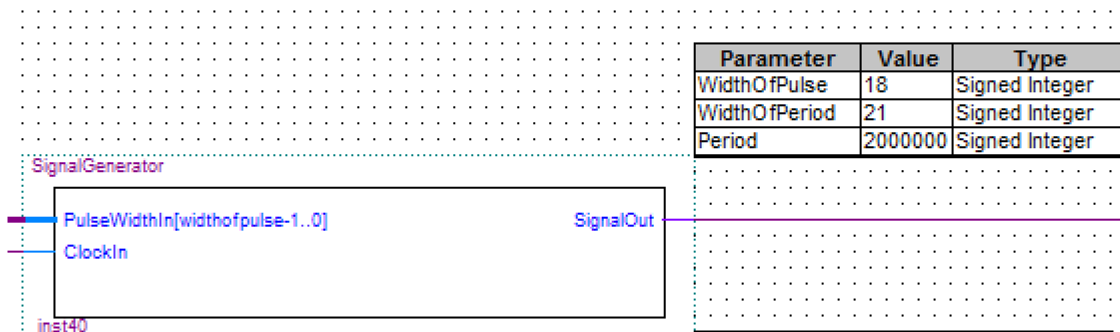


Figura 28: Sinal PWM.

Fonte: Autoria própria.



Parameter	Value	Type
WidthOfPulse	18	Signed Integer
WidthOfPeriod	21	Signed Integer
Period	2000000	Signed Integer

Figura 29: Bloco gerador de sinal PWM.

Fonte: Autoria própria.

A entrada "ClockIn" do bloco recebe um clock, no caso o geral do sistema, cuja finalidade é servir de referência para os cálculos de largura e período do sinal. No bloco acima o clock de entrada é de 100MHz e o período é 2 milhões de clocks, o que resulta em 20ms ou frequência de 50Hz. O BMS-620MG como servo analógico usa sinais PWM com frequência de 50Hz. O DS329MG, por sua vez, é digital e usa sinais com 200Hz de frequência.

A outra entrada, "PulseWidthIn", recebe a largura de pulso propriamente desejada. Vale ressaltar que o clock é a referência, logo a largura deve ser dada em número de clocks.

3.1.5 Gerenciador de sinais para os servomotores

Cada uma das 6 patas possui 3 servos para serem controlados, cada um com um ângulo objetivo definido segundo o processo de cinemática inversa. Em uma situação

ideal cada uma das patas teria seu próprio bloco de cinemática inversa e o cálculo seria executado em paralelo de maneira simultânea para todas elas. No entanto, isso não é possível pela falta de capacidade da FPGA utilizada. Fazendo uso de todos os blocos foi ultrapassado o limite de aproximadamente 22 mil elementos lógicos.

Com o objetivo de resolver esse problema, reduziu-se o número de blocos de cinemática inversa para apenas um e foi desenvolvido um bloco para armazenar os ângulos calculados e enviá-los às saídas de maneira sincronizada. A aparência externa do bloco pode ser vista na figura 30.

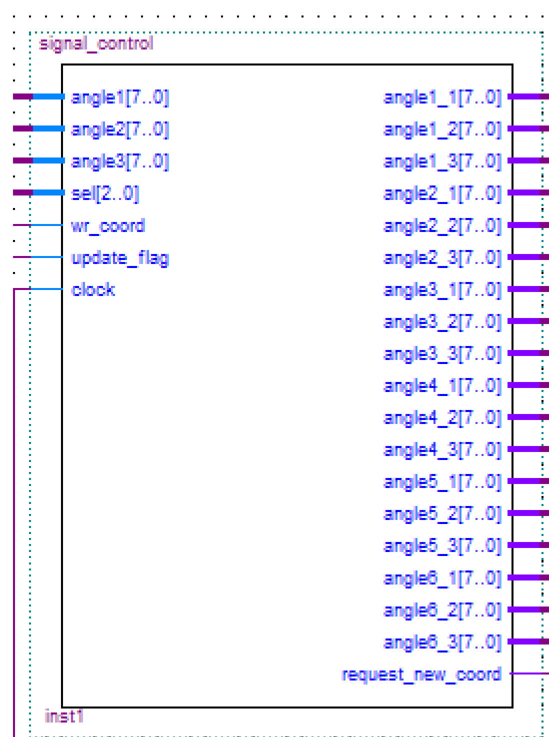


Figura 30: Bloco gerenciador de sinais

Fonte: Autoria própria.

Na parte esquerda do bloco angle1, angle2 e angle3 são entradas e recebem os ângulos dos servos de uma pata calculados pelo bloco de cinemática inversa. Os bits sel[2..0] são utilizados para selecionar uma das 6 patas, para qual se deseja armazenar os ângulos calculados. Para escrever as coordenadas a uma das patas além dos bits sel[2..0], deve-se manter o bit *wr_coord* para que a escrita seja devidamente realizada. Quando todas as escritas foram devidamente realizadas e as 6 patas possuem coordenadas relativas a um mesmo instante de tempo, o bit *update_flag* é setado para que as saídas *angleP_S* sejam atualizadas, onde P se refere a uma pata e S a um servo.

Após a atualização de todas as saídas, o bit *request_new_coord* é setado e indica o requerimento de novas coordenadas.

3.1.6 Comunicação I2C

Como apresentado anteriormente, todos os sensores utilizados usam comunicação por I2C e se comportam como *slaves*. Embora bastante completa, a biblioteca de componentes do SOPC Builder[77] não possui bloco de hardware como *master* I2C para ser integrado ao NIOS e realizar a comunicação com os sensores. Em situações como essa é conveniente buscar uma solução em sites como o OpenCores[78], que são comunidades de desenvolvimento de IPs (propriedade intelectual) disponibilizadas como código aberto (*open source*). A opção escolhida foi o *I2C controller core*[79], desenvolvido em Verilog por Richard Herveille. A figura 31 mostra a arquitetura interna do bloco.

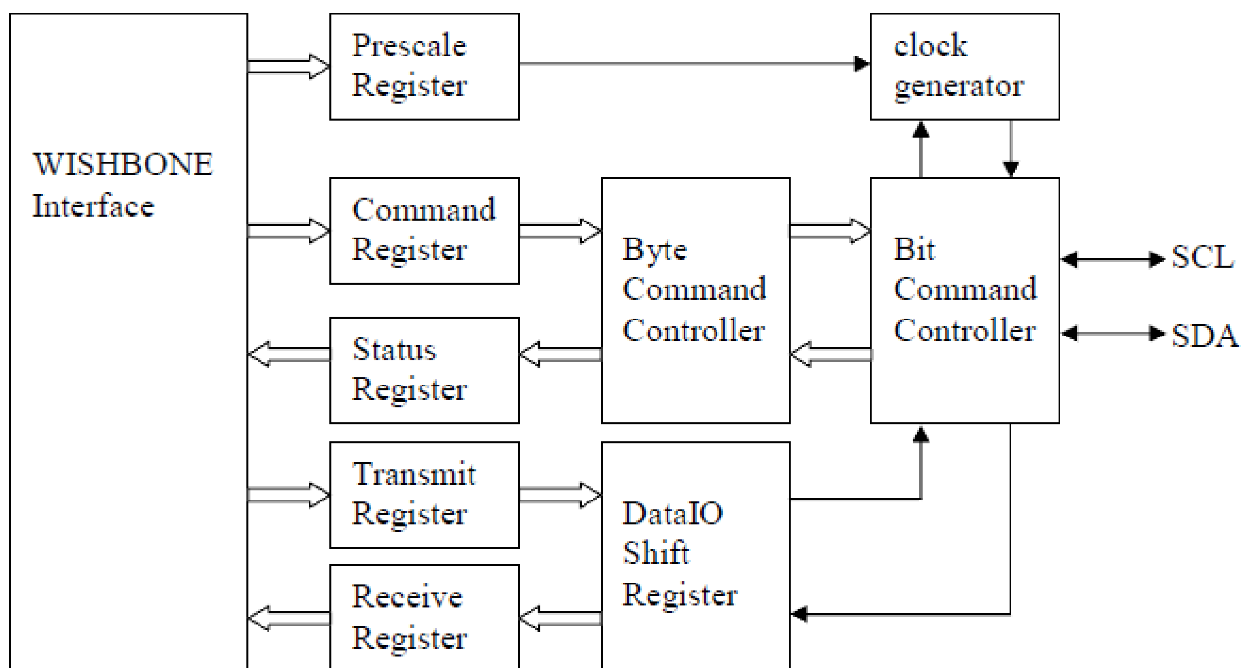


Figura 31: "Internal structure I2C Master Core".

Fonte: <http://opencores.org/project,i2c?do=svnget&project=i2c&file=%2Ftrunk%2Fdoc %2Fi2c_specs.pdf>

Do ponto de vista do usuário do bloco, somente os registradores podem ser acessados. Através do "Prescale Register" determina-se o clock que será utilizado na linha SCL. Com o "Command Register" determinam-se os próximos passos das SCL e SDA, como gerar sinais de *Start*, *Stop*, *Read*, *Write* e envio de ACK/NACK ao *slave*. O "Status Register" fornece informações das transmissões, como recebimento de ACK do

slave, canal I2C ocupado ou transmissão em andamento. Os últimos dois registradores "Transmit Register" e "Receive Register" contém respectivamente o próximo byte que se deseja enviar e o último byte válido recebido. Vale ressaltar que toda a comunicação com o bloco é feita através de bytes, sendo somente o "Prescale Register" dividido em 2 registradores de 1 byte.

O bloco I2C escolhido possui todos os recursos necessários, mas utiliza a interface open source Wishbone[80] para ser integrado ao resto do sistema. Um sistema SOPC com NIOS, no entanto, usa a interface proprietária Avalon[81] da Altera, incompatível diretamente com Wishbone. Para contornar esse problema é necessário um *wrapper* que seja conectado ao Avalon e repasse os dados recebidos na interface de forma compatível com a interface Wishbone, bem como o caminho inverso. Um *wrapper* com essa finalidade já foi desenvolvido e pode ser encontrado em [72].

3.1.6.1 Software de interfaceamento

Com o bloco I2C devidamente integrado ao resto do sistema SOPC e ao NIOS em termos de hardware, deve-se então estabelecer o contato entre eles do ponto de vista do *firmware*. Baseado nos exemplos da documentação fornecida, foram desenvolvidas funções simples para realizar algumas atividades necessárias quando se trata de comunicação I2C[82]. As funções tem o objetivo de reajustar o *clock*, enviar dados a um *slave*, escrever um registrador de um *slave* e ler o registrador de um *slave*. São elas:

- i2c_set_clock(system_clock, desired_scl)
  - Recebe como parâmetros o *clock* geral do sistema e o *clock* desejado na linha SCL. A partir dos parâmetros calcula-se o valor necessário de *prescale*. O byte mais alto do valor calculado e o mais baixo são escritos respectivamente nos registradores PRERhi e PRERlo.
- i2c_write_data(slave_address, data)
  - Escreve no TXR (Transmit Register) o endereço *slave_address* e envia ao *slave*, setando os bits STA (*Start*) e WR (*write*) do CR (*Command Register*). Aguarda pelo fim da transmissão e confirmação de recebimento pelo *slave*, enviando em seguida o parâmetro data pelo mesmo processo mas com o bit STO (*Stop*) em vez de STA. Ao enviar um STO o *master* requisita o fim da comunicação com *slave*. O *master* espera pelo fim da transmissão e confirmação de recebimento do dado pelo *slave*.

- i2c_write_register(slave_address, register_address, data)
    - Similar a função anterior, mas envia o dado para ser gravado em um dos registradores do *slave*. A única diferença é que há mais uma rodada de comunicação, depois do envio do *slave_address* e antes do envio de data. Nessa etapa adicional os bits STA e STO não são setados.
- i2c_read_register(slave_address, register_address)
    - Envia um endereço *slave_address* e um registrador de maneira similar ao processo de escrita, mas em vez de enviar um dado, envia novamente o endereço com os bits STA e WR setados. Nesse caso, setar o bit STA é uma requisição de reinicio. Na quarta rodada apenas são setados os bits RD (*Read*), ACK e STO do CR. Dessa forma o *master* libera o barramento para ser escrito pelo *slave* e após o fim da transmissão pede o fim da comunicação.

3.2 Hardware externo

3.2.1 Desenvolvimento dos servomotores próprios

Para a construção de um servomotor próprio utilizou-se de base um servo Motortech[83]. O servo foi aberto e desmontado, de forma a remover toda a eletrônica responsável pelo controle e amplificação dos sinais, como mostrado na figura 32. Restaram somente as engrenagens, o motor DC e o potenciômetro que serve como mecanismo de realimentação da posição.

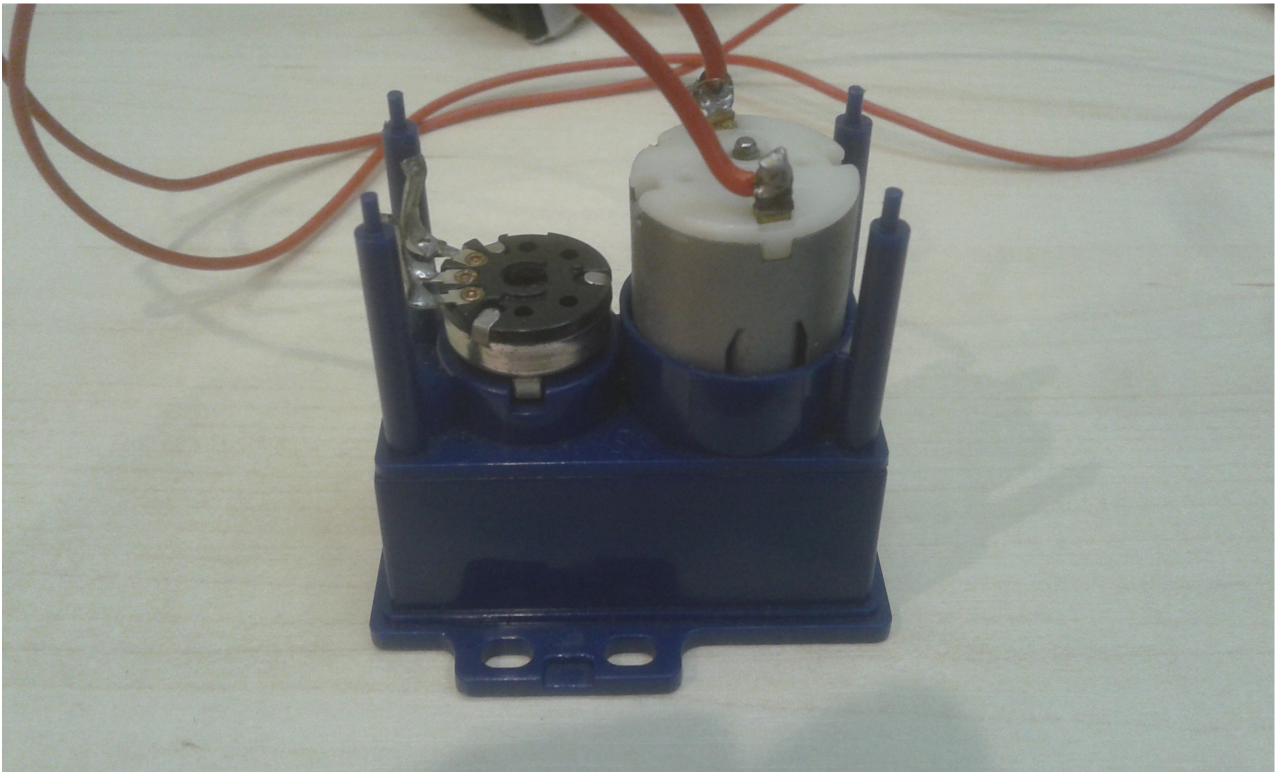


Figura 32: Servo desmontado.

Fonte: Autoria própria.

Dos componentes restantes, utilizam-se os terminais do potenciômetro em um circuito similar ao descrito na seção da parte teórica. A conexão do potenciômetro ao LM555 pode ser visualizada na figura 33, configuração **c** (astável). Os terminais do potenciômetro ficam conectados ao LM555 de forma que Ra é uma das parcelas do potenciômetro e Rb é a resistência restante. A frequência do sinal produzido é constante e a largura é variável. Esse sinal de saída produzido pelo LM555 é então conectado a FPGA, que, com base na largura do pulso, determina a posição (ângulo) do servo.

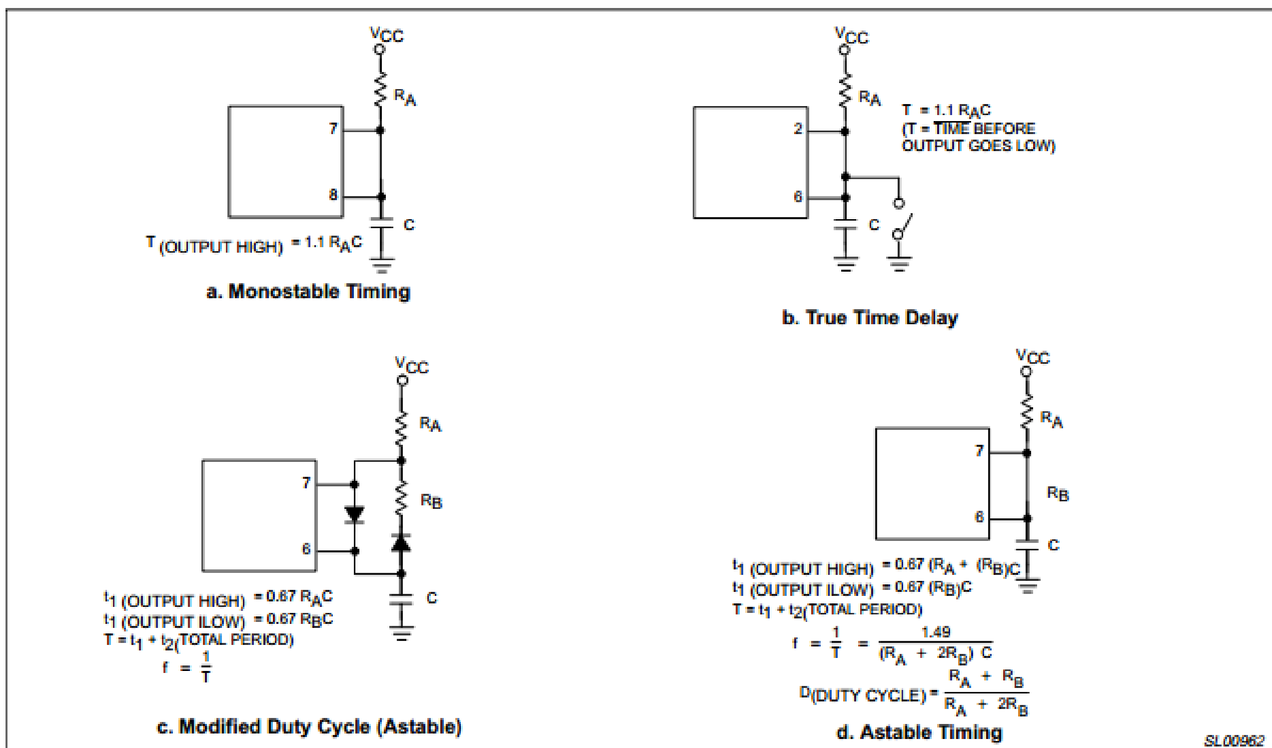


Figura 33: Configurações de LM555.

Fonte: <http://www.doctronics.co.uk/pdf_files/555an.pdf>

É necessário encontrar uma relação entre ângulo e largura do pulso produzido para realizar a devida interpretação na FPGA. Para realizar essa tarefa foram utilizados os instrumentos osciloscópio e transferidor. Manipulando-se o eixo do servo manualmente e verificando o sinal produzido com o osciloscópio chegou-se a uma relação linear, explicitada na equação 11, onde x é um pulso com valor entre 0 e 1000, e y é um ângulo entre 0 e 180 graus.

$$y=(x-60)/2.11 \quad (11)$$

Após a interpretação do sinal, a FPGA gera um sinal PWM calculado por um controlador proporcional da forma descrita na equação 12, onde P é uma constante pré-definida e *error* é a diferença entre ângulo desejado e ângulo real. O sinal PWM é invertido pois é ligado à eletrônica de potência através de opto acopladores em modo emissor comum. Finalmente o sinal de saída do opto acoplador passa pelo L298, que o amplifica, para então controlar o motor DC. O melhor resultado encontrado nos testes, com o objetivo de minimizar o *overshoot* e maximizar a velocidade de resposta foi com P = 2.

$$P_{out} = P * error ; \quad (12)$$

3.2.2 Placas de Circuito Impresso

As Placas de Circuito Impresso (PCI) produzidas para o projeto tem como principais objetivos distribuir a energia e realizar a conexão entre os dispositivos (XBee, servos e sensores) e a FPGA. A ideia original era produzir apenas uma PCI contendo todos esses elementos, entretanto, após uma análise e um projeto inicial verificou-se que não seria possível produzir uma única placa com tamanho adequado para ser encaixada a estrutura mecânica do Hexápode.

Por isso foram elaboradas 2 placas, uma responsável por distribuir as conexões dos servomotores, intitulada PCI dos servos. E outra contendo todas as conexões da FPGA, dos sensores e do XBee chamada de PCI da FPGA. Quanto a alimentação do circuito foram utilizadas duas fontes independentes, sendo uma responsável por alimentar a PCI dos servos e a outra para alimentar a PCI da FPGA. Mais detalhes são explicados na seção Fonte de alimentação.

Apesar de serem duas placas isoladas elas se conectam, através de barras de pinos paralelas, para passar os sinais do FPGA para os motores formando uma PCI de 2 níveis. Essa abordagem foi adotada para evitar a utilização de conectores com cabos e transformar o projeto eletrônico em partes modulares para facilitar a montagem e a manutenção do circuito. Além disso a FPGA é encaixada na sua PCI da mesma maneira, facilitando a sua remoção e também evitando possíveis problemas de mal contato por conta de cabos. A seguir serão explicados os projetos da duas placas de maneira isolada, analisando primeiramente o esquemático do circuito e depois o projeto da placa.

3.2.2.1 PCI da FPGA

O projeto da PCI da FPGA sofreu poucas alterações depois da sua concepção inicial e possuí um projeto relativamente simples. Para facilitar o entendimento do circuito completo, que pode ser visto na figura 34, o esquemático será explicado em 4 partes distintas.

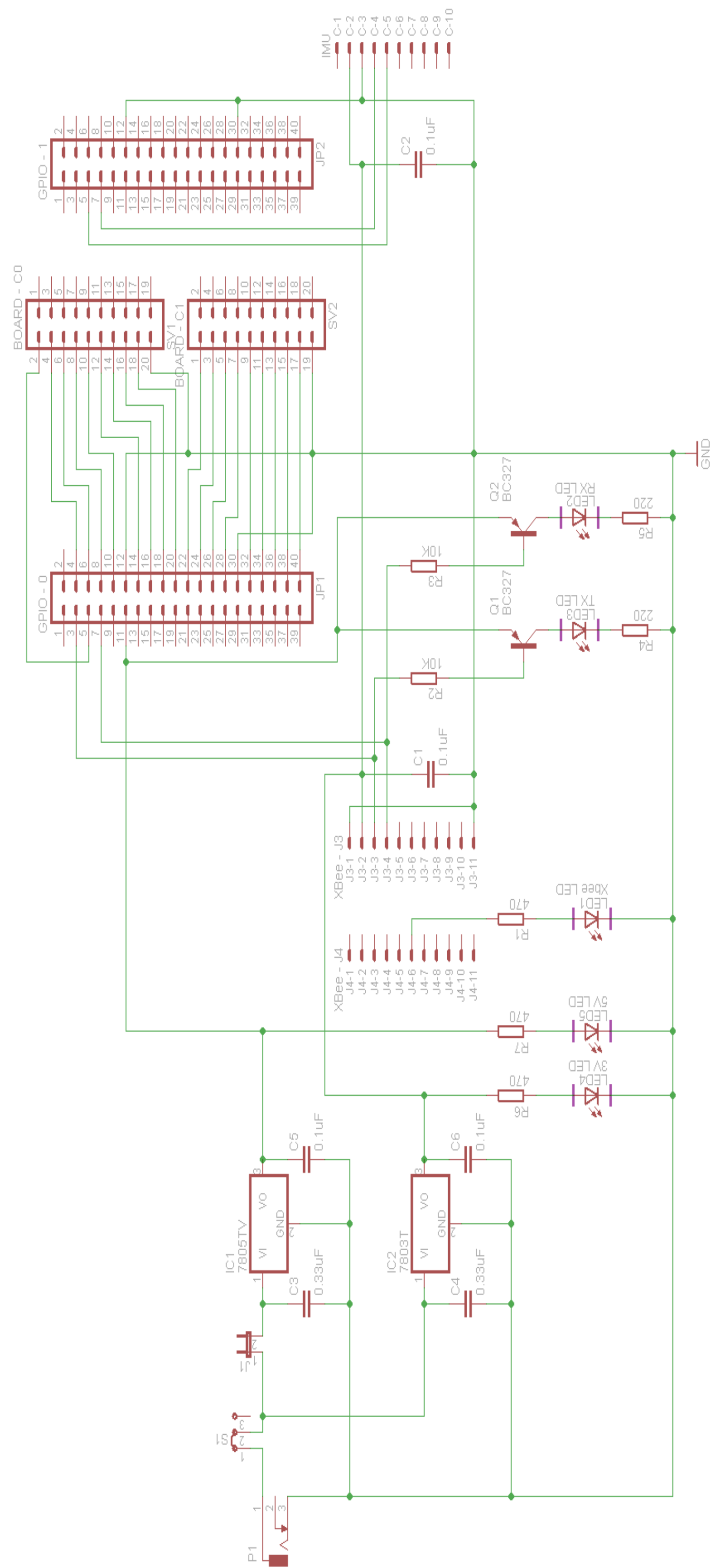


Figura 34: Esquemático do circuito da PCI da FPGA.

Fonte: Autoria própria.

A primeira parte realiza a conexão entre a FPGA e a placa dos servomotores. Essas conexões são responsáveis por passar os sinais de PWM da FPGA para os motores. A figura 35 apresenta a configuração dos pinos de saída da FPGA e são idênticos aos conectores GPIO-0 e GPIO-1 apresentados na figura 34. Na configuração do projeto o GPIO-0 é configurado como canal de saída para os pinos GPIO_00 a GPIO_33 e possui os pinos GPIO_0_IN0 e GPIO_0_IN1 como pinos de entrada. A tabela 4 e a tabela 5 apresentam as configurações dos pinos utilizados para conectar as duas PCIs, todos os pinos não relacionados nas tabelas não possuem nenhuma conexão.

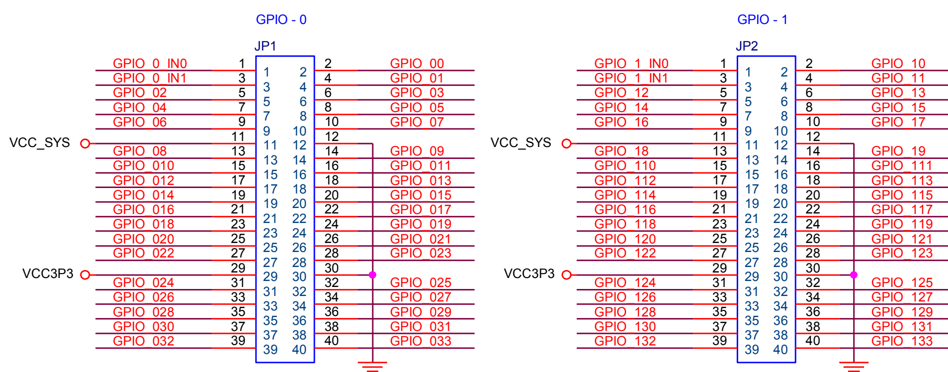


Figura 35: Pinos de I/O da FPGA (GPIO-0 e GPIO-1)

Fonte: Retirado de [14]

Tabela 4: Configuração dos pinos do conector BOARD-C0

Fonte: Autoria própria.

Board - C0 [Pin]	GPIO - 0 [Pin]	FPGA [Pin]	Função
2	5	GPIO_02	Sinal de PWM para o Servo 1
4	4	GPIO_01	Sinal de PWM para o Servo 2
6	6	GPIO_03	Sinal de PWM para o Servo 3
8	8	GPIO_05	Sinal de PWM para o Servo 4
10	10	GPIO_07	Sinal de PWM para o Servo 5
12	14	GPIO_09	Sinal de PWM para o Servo 6
14	16	GPIO_011	Sinal de PWM para o Servo 7
16	18	GPIO_013	Sinal de PWM para o Servo 8
18	20	GPIO_015	Sinal de PWM para o Servo 9
20	-	-	GND

Tabela 5: Configuração dos pinos do conector BOARD-C1

Fonte: Autoria própria.

Board - C1 [Pin]	GPIO - 0 [Pin]	FPGA [Pin]	Função
1	22	GPIO_017	Sinal de PWM para o Servo 10
3	24	GPIO_019	Sinal de PWM para o Servo 11
5	26	GPIO_021	Sinal de PWM para o Servo 12
7	28	GPIO_023	Sinal de PWM para o Servo 13
9	32	GPIO_025	Sinal de PWM para o Servo 14
11	34	GPIO_027	Sinal de PWM para o Servo 15
13	36	GPIO_029	Sinal de PWM para o Servo 16
15	38	GPIO_031	Sinal de PWM para o Servo 17
17	40	GPIO_033	Sinal de PWM para o Servo 18
19	-	-	GND

A outra parte da placa é a que conecta a FPGA com o módulo contendo os sensores (acelerômetro, giroscópio e magnetômetro) utilizados no hexápode, chamado de IMU. A IMU é conectada ao barramento GPIO–1 e alimentada com uma tensão de 3.3V. A figura 36 apresenta os pinos da IMU e a tabela 6 as configurações dos pinos utilizados. Novamente, os pinos não apresentados não são utilizados.

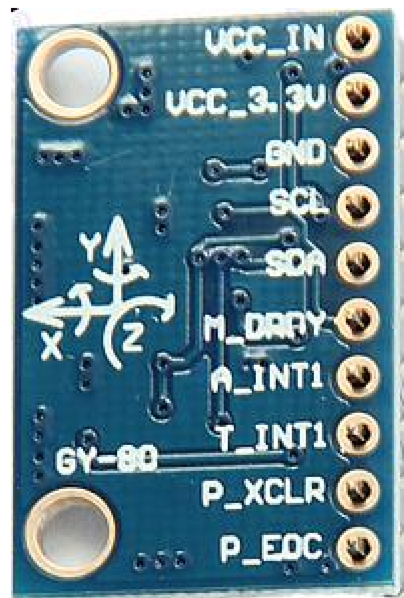


Figura 36: Pinos da IMU.

Fonte: Autoria própria.

Tabela 6: Configuração dos pinos da IMU

Fonte: Autoria própria.

IMU [Pin]	GPIO - 1 [Pin]	FPGA [Pin]	Função
C-2	-	-	VCC 3.3V
C-3	-	-	GND
C-4	7	GPIO_14	SCL (clock) do I2C
C-5	5	GPIO_12	SDA (dados) do I2C

A conexão com o XBee é a terceira parte da PCI da FPGA. Ela consiste basicamente da conexão dos pinos de RX e TX do XBee com o GPIO-0 da FPGA. Além disso existe um circuito simples com dois transistores e dois LEDS para indicar o envio e recebimento de dados através do RX e TX, e um LED indicador de funcionamento do XBee. A figura 37 apresenta a placa para adaptar o módulo XBee com a configuração DIP. Os barramentos J3 e J4 apresentados são os mesmos que aparecem no esquemático do circuito da PCI da FPGA da figura 34. A tabela 7 apresenta a configuração dos pinos do módulo XBee utilizados na PCI, aqueles não apresentados não foram utilizados.

Por fim, a última parte dessa PCI é referente ao circuito de alimentação onde o IC1 e IC2 da figura 34 representam respectivamente um regulador de tensão de 5V e 3.3V cujas saídas estão conectadas a 2 LEDs indicando o funcionamento dos reguladores.

A FPGA é alimentada com 5V e o XBee e a IMU com 3.3V e possuem um capacitor de desacoplamento para filtrar os possíveis ruídos na alimentação. Por fim existe uma chave para ligar e desligar o conector de energia (S1) e um *jumper* (J1) para permitir que a FPGA seja conectada e alimentada diretamente pelo computador através da USB, facilitando etapas de testes.

Após terminado o circuito foi realizado o projeto da PCI no Eagle para distribuir corretamente todos os elementos na placa. As maiores dificuldades encontradas nessa etapa foram ajustar corretamente todos os elementos em uma placa com tamanho adequado para encaixar corretamente na estrutura mecânica do hexápode e ajustar os espaçamentos para encaixar a FPGA na PCI. Durante a distribuição dos componentes também foi necessário afastar a placa dos sensores da linha de alimentação para evitar uma possível interferência. Como a placa não necessita de trilhas com uma corrente elétrica superior a 1.5A, foi possível utilizar sem nenhum problema as configurações padrões utilizadas no Eagle e fazer o roteamento manual das trilhas. O Diagrama da PCI pode ser visto abaixo na figura 38. As figuras 39 e 40 apresentam o resultado final da PCI da FPGA.

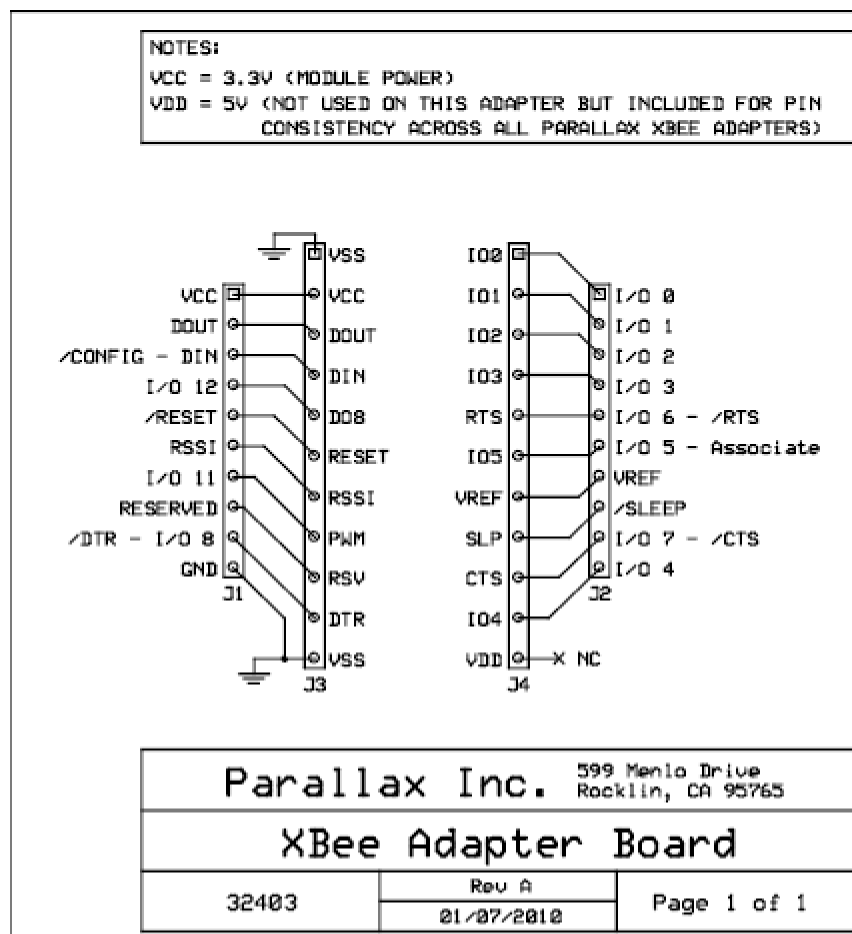


Figura 37: Pinos do Módulo XBee.

Fonte: Retirada de [35]

Tabela 7: Configuração dos pinos do Módulo XBee na PCI

Fonte: Autoria própria.

XBee [Pin]	GPIO - 0 [Pin]	FPGA [Pin]	Função
J3-1	-	-	GND
J3-2	-	-	VCC 3.3V
J3-3	3	GPIO_0_IN1	TX do XBee
J3-4	7	GPIO_04	RX do XBee
J3-11	-	-	GND
J4-6	-	-	Indicador de funcionamento do Xbee

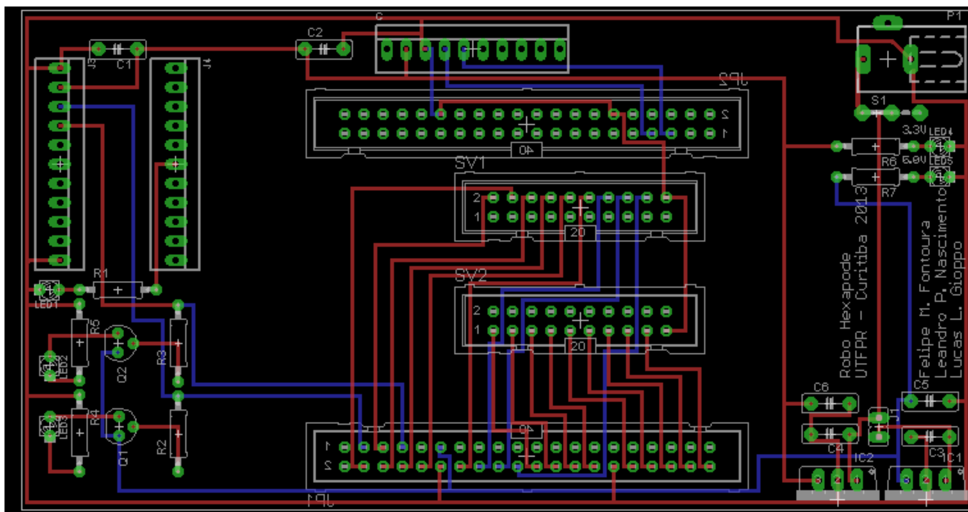


Figura 38: Diagrama da PCI da FPGA.

Fonte: Autoria própria.

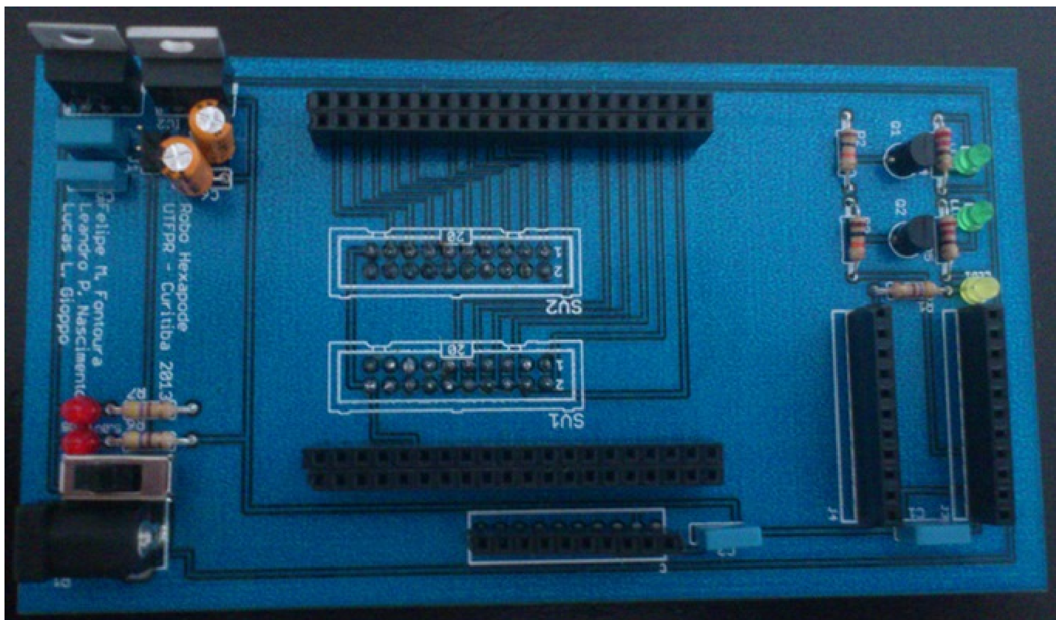


Figura 39: PCI da FPGA com todos os componentes soldados.

Fonte: Autoria própria.

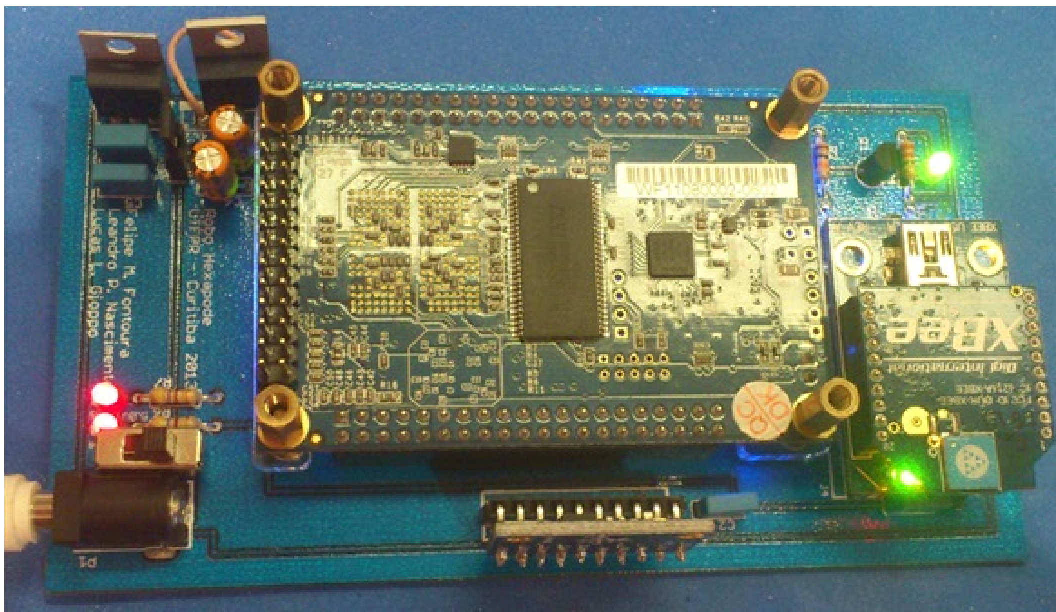


Figura 40: PCI da FPGA ligada com o Xbee, a FPGA e a IMU.

Fonte: Autoria própria.

3.2.2.2 PCI dos Servos

O circuito da PCI dos servos, apesar de maior, é mais simples do que o circuito da PCI anterior. Ele pode ser dividido em 2 partes, uma responsável por enviar os sinais de PWM provenientes da PCI da FPGA para os motores e outra por fornecer a alimentação para todos eles. O esquemático completo pode ser visto na figura 41.

Os conectores BOARD-C0 e BOARD-C1 são exatamente os mesmos utilizados na PCI da FPGA, pois são os responsáveis por conectar as duas placas e a configuração dos seus pinos pode ser vista nas tabelas 4 e 5. Cada um dos sinais de PWM, um por servo, é enviado para um opto acoplador que tem como função isolar a alimentação das duas placas e mudar o nível do PWM de 3.3V gerado pela FPGA para os 5.0V utilizados pelos motores. O sinal de saída dos opto acopladores, alimentados na saída por uma das linhas de 5V da fonte ATX que alimenta os motores, é enviado para cada um dos 18 servos utilizados no hexápode.

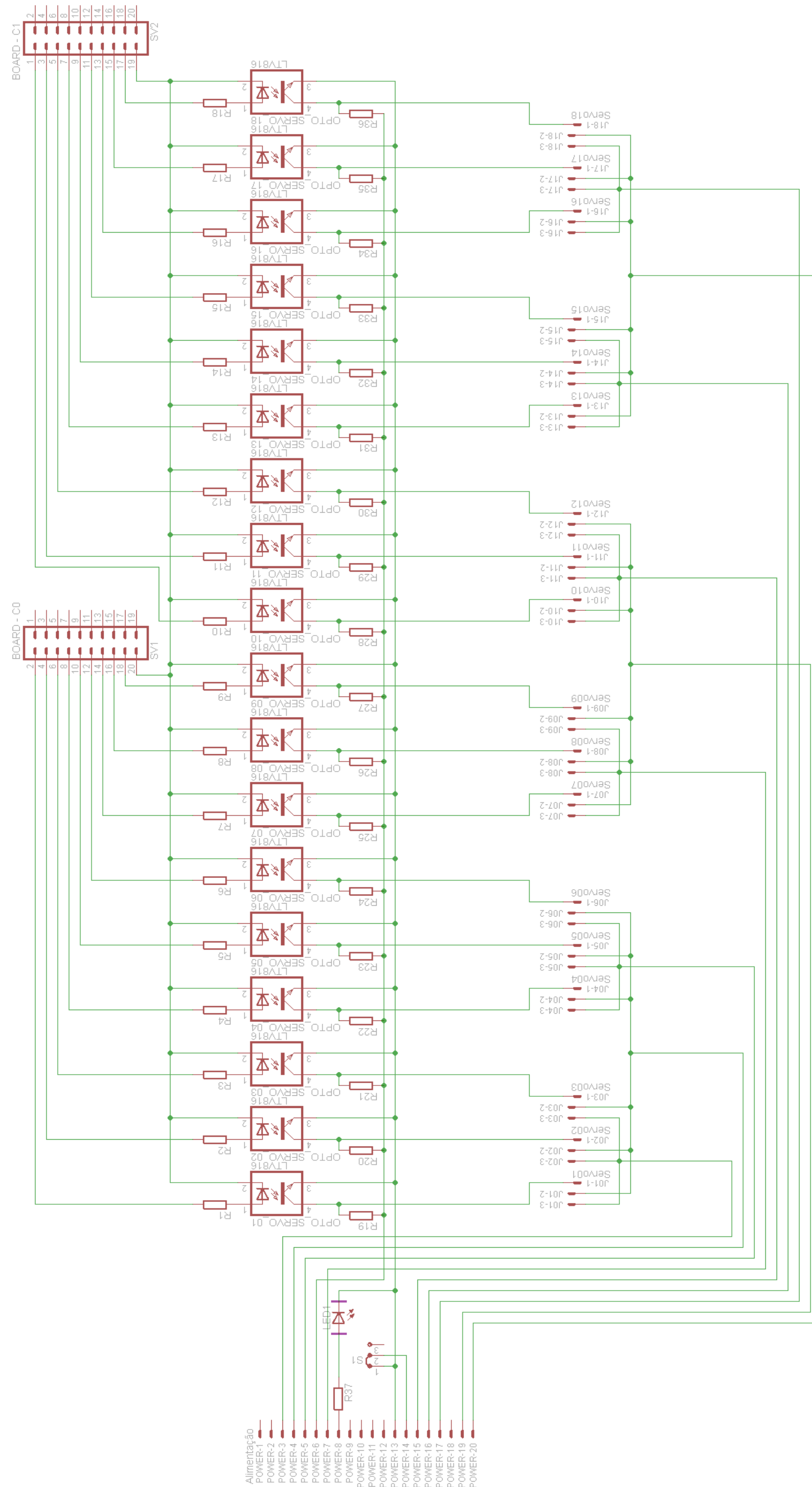


Figura 41: Esquemático do circuito da PCI dos servos.

Fonte: Autoria própria.

Para reduzir a corrente das trilhas nessa placa de circuito foram utilizadas as quatro conexões de 5.0V e as sete GND provenientes da fonte ATX. Na tabela 8 são apresentados como foram distribuídos os canais de alimentação para os motores e opto acopladores utilizados na PCI. Por fim, o circuito possui um LED ligado na pino Power Ok indicando se a fonte está ligada e uma chave conectando o GND ao Power Supply On para ligar a alimentação da fonte diretamente da PCI dos servomotores.

Tabela 8: Configuração dos pinos da Fonte ATX utilizados na PCI dos Servomotores Fonte: Autoria própria.

Alimentação [Pin]	Função
POWER-3	GND dos Servos 01 a 03
POWER-4	5.0V dos Servos 01 a 06
POWER-5	GND dos Servos 04 a 06
POWER-6	5.0V utlizado nos Optoacopladores
POWER-7	GND dos Servos 07 a 09
POWER-8	Power Ok conectado ao LED indicador da alimentação
POWER-13	GND utilizado nos Optoacopladores e Power Supply On
POWER-14	Power Supply On da Fonte ATX
POWER-15	GND dos Servos 10 a 12
POWER-16	GND dos Servos 13 a 15
POWER-17	GND dos Servos 16 a 18
POWER-19	5.0V dos Servos 07 a 12
POWER-20	5.0V dos Servos 13 a 18

Apesar do circuito ser mais simples do que o utilizado na placa da FPGA, o diagrama da placa foi bem mais trabalhoso pois foi necessário realizar algum estudo prévio e ajustar algumas configurações da PCI para suportar trilhas com corrente próxima a 5.5A. O diagrama final do projeto da PCI dos servos pode ser visto na figura 42. A placa precisou ser projetada para encaixar na PCI da FPGA sem gerar problema com os conectores utilizados nos servomotores, através dos conectores no centro da placa. Outra coisa que gerou um trabalho adicional foi o fato dessa placa necessitar de 4 furos para ser encaixada na estrutura mecânica do hexápode.

Todos os componentes da placa precisaram ser ajustados para poder permitir as trilhas largas (90mil) utilizadas na alimentação dos 5V, para fornecer energia para cada par de patas (6 servos). Trilhas um pouco mais finas (40mil) foram utilizadas para as linhas de terra (uma trilha para cada pata). Além disso, a placa exigiu uma camada de cobre mais densa (1.4oz) para suportar toda a corrente dos motores sem aquecer muito o circuito. Para calcular a espessura e largura das trilhas para suportar a corrente de

operação foram utilizadas as ferramentas e as fórmulas presentes em [84]. A figura 43 apresenta a PCI dos servomotores antes de terem sido soldados os componentes e a figura 44 a versão final da PCI com todos os componentes e parafusada a estrutura mecânica do hexápode.

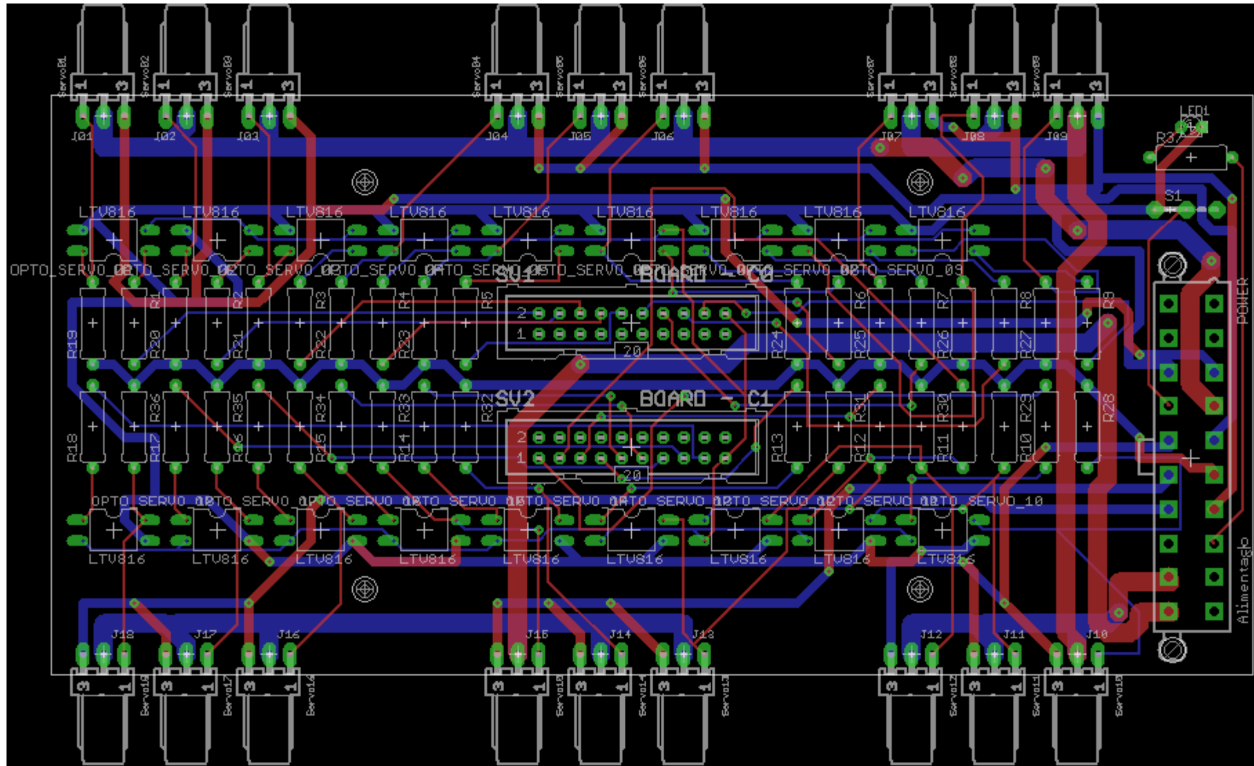


Figura 42: Diagrama da PCI dos servos.

Fonte: Autoria própria.

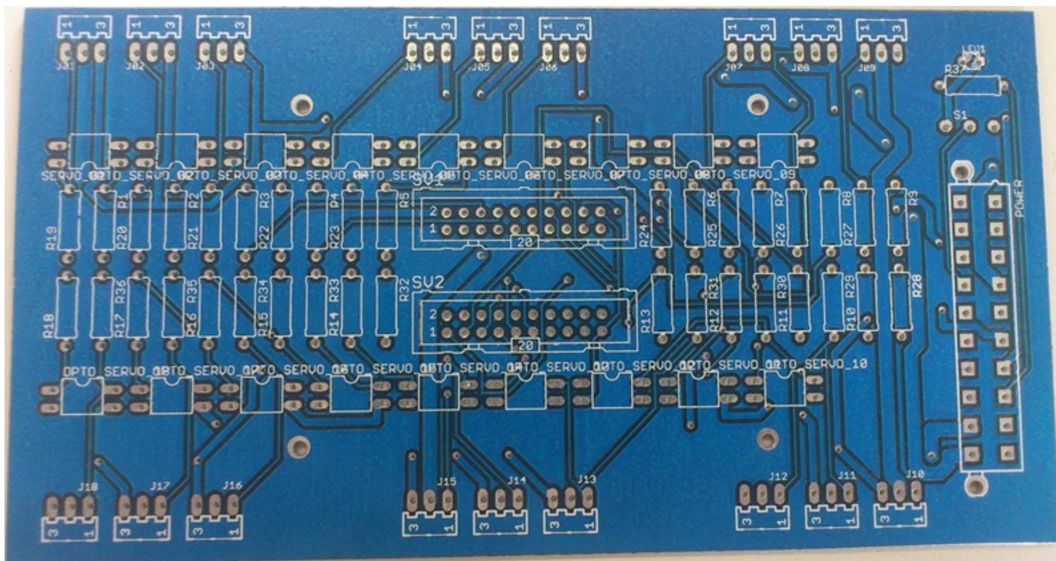


Figura 43: PCI dos servos sem os componentes soldados.

Fonte: Autoria própria.

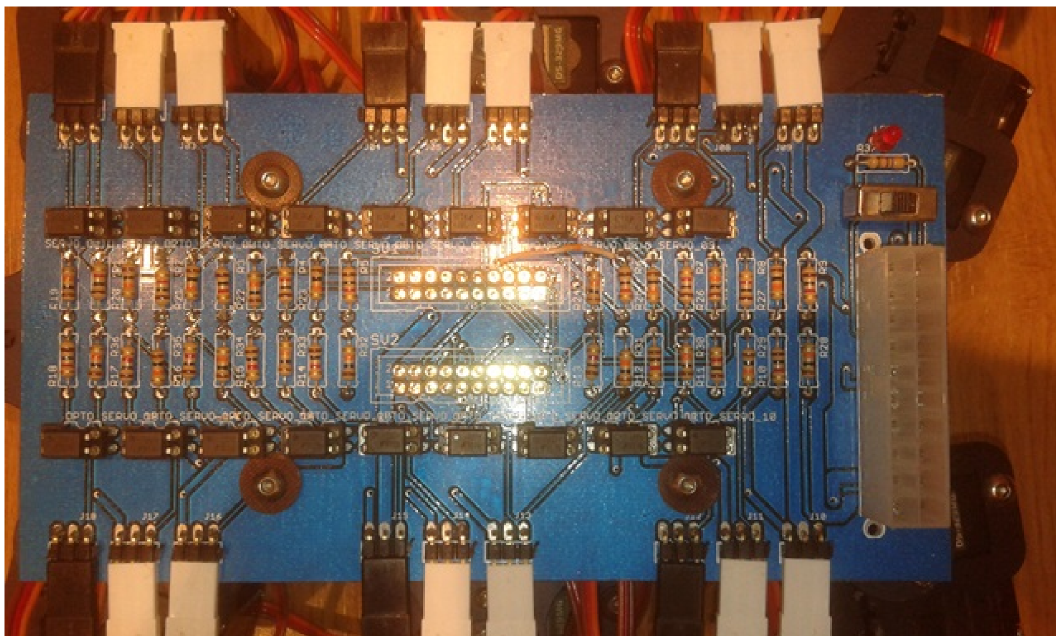


Figura 44: Versão final da PCI dos servos parafusada ao hexápode.

Fonte: Autoria própria.

3.2.3 Fonte de alimentação

A fonte de alimentação escolhida para ser utilizada como alimentação dos servomotores no projeto foi uma fonte genérica de computador do tipo ATX com potência máxima, em pico, de 450W. Como os dois tipos de servos utilizados no projeto, BMS-620MG e Corona DS329, trabalham entre 4.8V e 6.0V[43,44] foi utilizado o canal de 5V da fonte para fornecer a energia a eles.

Uma fonte do tipo ATX possuí diversos tipos de conectores para alimentar os diferentes dispositivos existentes dentro de um computador: placa-mãe, HD, leitor de DVD, placa de vídeo, ventiladores, dentre outros. Entretanto, somente o conector principal, que fornece a energia para a placa-mãe, foi utilizado no projeto, figura 45. Existem 2 variações desse tipo de conector: uma é um conector Molex de 20 pinos e a outra é um conector Molex de 24 pinos. A principal diferença entre eles é a simples adição de algumas linhas adicionais de alimentação, entretanto, eles mantém uma compatibilidade entre seus 20 pinos comuns, cuja configuração pode ser vista na tabela 9.

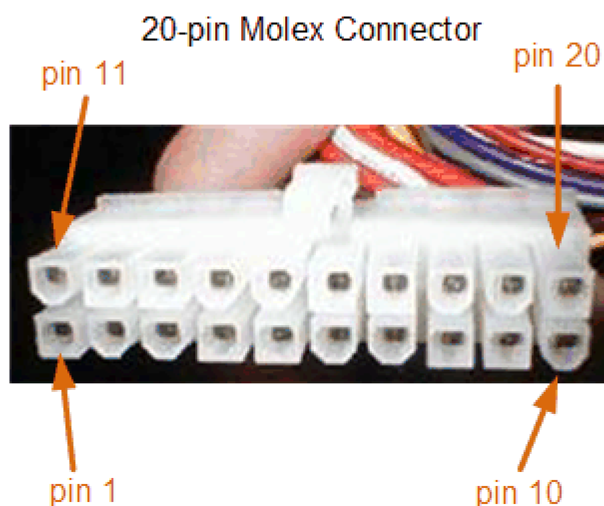


Figura 45: Conector Molex de 20 pinos.

Fonte: Autoria própria.

Por ser o conector com o maior número de linhas de 5V e GND, esse foi o conector utilizado para fornecer a energia da fonte para o hexápode. Porém, como a extensão do cabo que sai da fonte é pequena, foi necessário construir um cabo extensor para ligá-lo

ao robô. Foi utilizado no projeto a linha de 5V (pinos 4, 6, 19 e 20), a linha do GND (pinos 3, 5, 7, 13, 15, 16, 17), o Power Ok (pino 8) e o Power Supply On (pino 14).

Os pinos de GND e 5V foram utilizados para alimentar os motores e os opto acopladores, aos quais foram conectados os pinos de 5V em conjunto com um ou mais pinos de GND (terra), de maneira a isolar a alimentação dentro da PCI dos motores e reduzir a corrente atuante em cada trilha da placa. Vale ressaltar que apesar das diferentes linhas de GND e de 5V estarem isoladas na placa de alimentação, elas estão conectadas dentro da fonte ATX. As conexões podem ser vistas com mais detalhes na sessão da PCI dos motores.

Além das linhas de alimentação padrão utilizadas nos motores foram utilizados os pinos de Power Ok e Power Supply On. O primeiro serviu apenas para alimentar um LED da placa dos motores para indicar que a fonte está ligada. O outro foi conectado a uma chave para ligar ou desligar a alimentação da fonte, acionada quando o Power Supply On está em nível lógico baixo (0 V)[85].

A alimentação da FPGA e dos sensores provem de uma outra fonte de alimentação, que pode ser qualquer fonte DC com tensão entre 6V e 12V que consiga fornecer ao menos 1A de corrente.

Tabela 9: Pinos de saída do conector de 20 pinos da fonte ATX.

Fonte: Retirada de [85]

Pin No	Name	Colour	Description
1	3.3V	Orange	+3.3 VDC
2	3.3V	Orange	+3.3 VDC
3	COMMON	Black	Ground
4	5V	Red	+5 VDC
5	COMMON	Black	Ground
6	5V	Red	+5 VDC
7	COMMON	Black	Ground
8	Pwr_Ok	Gray	Power Ok (+5 VDC when power is Ok)
9	+5VSB	Purple	+5 VDC Standby Voltage
10	12V	Yellow	+12 VDC
11	3.3V	Orange	+3.3 VDC
12	-12V	Blue	-12 VDC
13	COMMON	Black	Ground
14	Pwr_ON	Green	Power Supply On (active low)
15	COMMON	Black	Ground
16	COMMON	Black	Ground
17	COMMON	Black	Ground
18	-5V	White	-5 VDC
19	5V	Red	+5 VDC
20	5V	Red	+5 VDC

3.3 Montagem do hexápode

O processo de montagem do hexápode consistiu basicamente em integrar a estrutura mecânica do robô (MSR-H01) aos motores, encaixar as PCIs na parte superior e fazer um cabo extensor para alimentação da fonte. Apesar do processo ser simples, ele foi um pouco mais trabalhoso do que o esperado e alguns ajustes foram necessários para que tudo funcionasse de maneira adequada.

A primeira etapa desenvolvida foi a montagem da estrutura mecânica com os motores utilizados seguindo o guia de montagem [86], fornecido pelo fabricante, da parte mecânica do hexápode. Foram seguidos praticamente todos os passos descritos no guia, exceto aqueles que envolviam o encaixe da placa de controle e bateria pois esses itens não foram comprados e utilizados no projeto. Entretanto, apesar de possuir medidas semelhantes ao Hitec HS-645MG utilizado no guia, foi necessário limar cerca de um milímetro as laterais das extremidades do BMS-620MG para que fosse possível encaixá-lo entre as peças de número MSR-H01-ASSY01 e MSR-H01-ASSY02 conforme as etapas 3 e 4 descritas no guia. Além disso foi necessário perfurar a parte de plástico para encaixar corretamente os motores na peça MSR-H01-FEMUR apresentada na etapa 6. O motor Corona DS329MG não apresentou nenhuma dificuldade durante a montagem por substituir o HS-225MG. A figura 46 apresenta as peças do Kit MSR-H01 após realizado o encaixe dos motores.

Figura 46: Peças do MSR-H01 com alguns servos encaixados

Fonte: Autoria própria.

Uma das etapas mais importantes durante a montagem do hexápode consiste em centralizar todos os motores e encaixá-los de modo que eles fiquem o mais alinhado possível e preferencialmente com o mesmo grau de liberdade dos dois lados. Entretanto é difícil garantir que todos os motores irão se manter na mesma posição durante o encaixe das peças, por isso foi necessária uma calibragem individual dos motores após o processo de montagem. Para realizar este ajuste foi enviado um comando para centralizar os motores e ajustado um *offset* na faixa de operação do bloco de PWM de cada um deles. Com isso foi possível garantir que, na posição central dos motores, a variação das medidas entre eles fosse próxima a um milímetro no pior caso. A figura 47 mostra o kit MSR-H01 após a montagem completa das peças.

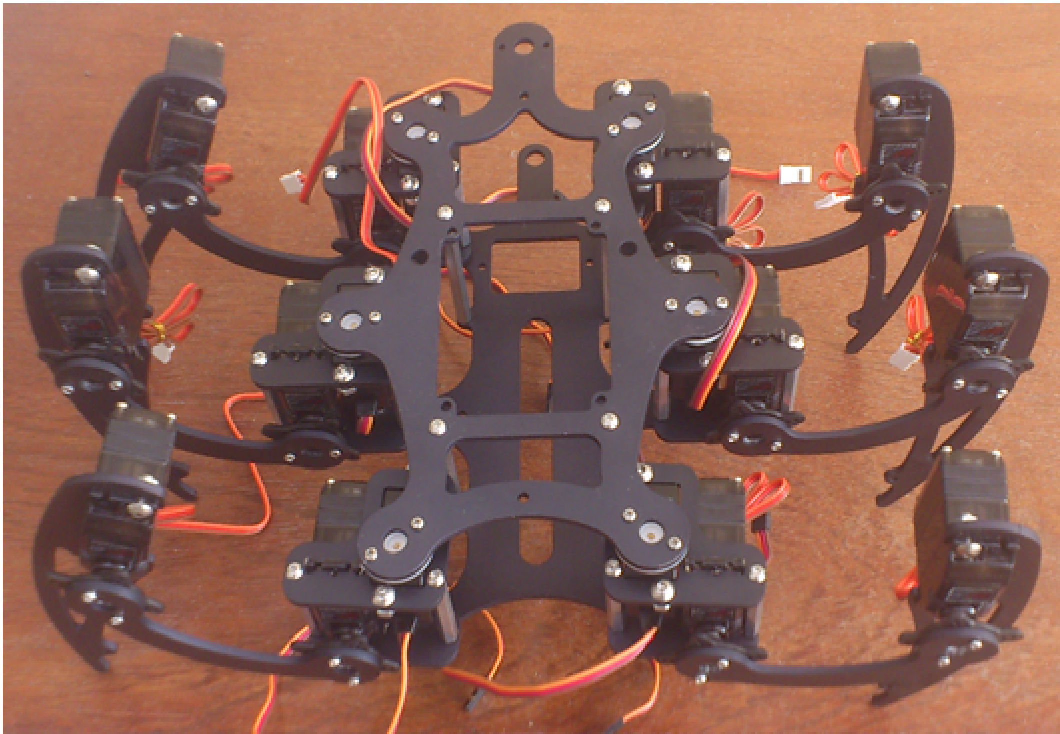


Figura 47: Kit MSR-H01 montado com todos os servos encaixados

Fonte: Autoria própria.

Após a montagem da estrutura mecânica as PCIs foram soldadas e a placa dos servos foi parafusada na parte superior do hexápode. Para isso foram aproveitadas as peças MSR-P002 da etapa 9 do guia de montagem[86], mas ao invés de parafusar na parte de baixo elas foram encaixadas na parte superior do robô, pois as placas utilizadas no projeto tinham dimensões que impossibilitaram colocá-las na parte de baixo. A placa de alta potência, utilizada para alimentar os servos, foi parafusada diretamente na estrutura e a placa de controle contendo a FPGA foi encaixada nela. A IMU com os sensores, o XBee e a FPGA então foram encaixados por cima da placa de controle concluindo a montagem do hexápode. A figura 48 apresenta o hexápode após terem sido encaixadas as duas PCIs, entretanto a PCI com o Xbee e os sensores não tinha sido finalizada.

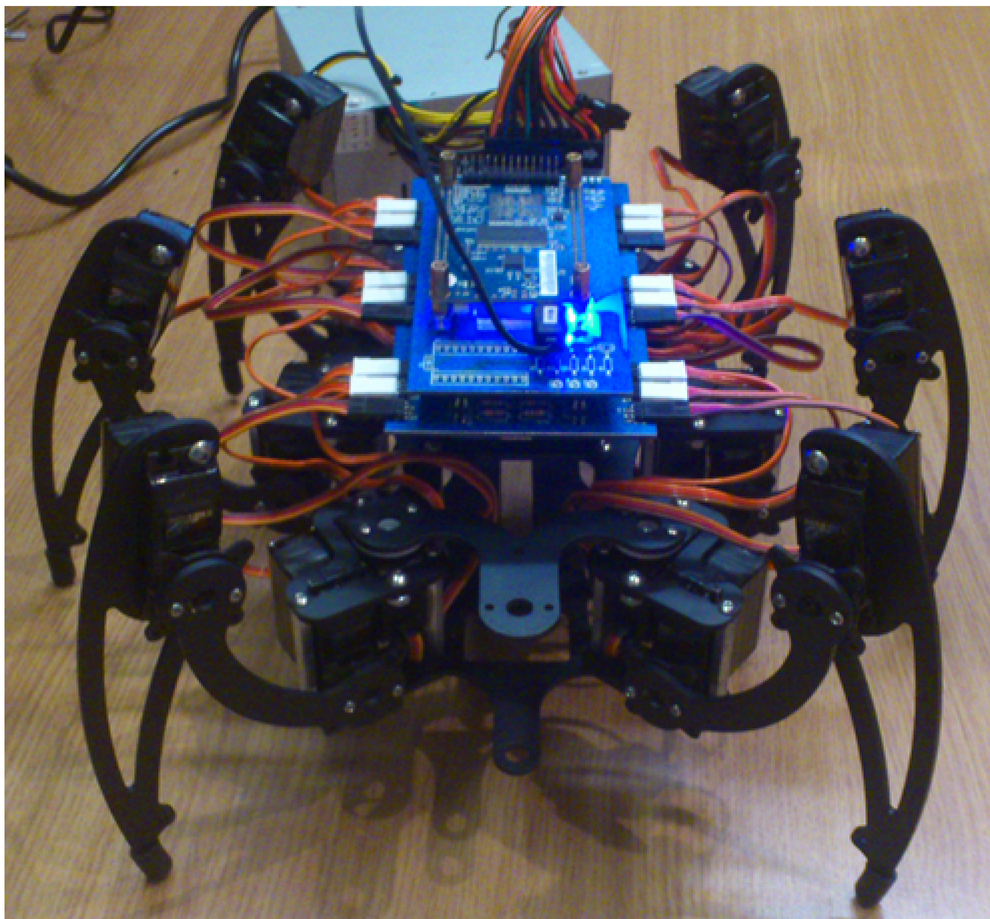


Figura 48: Hexápode com as PCIs e com a FPGA mas sem a PCI superior ter sido soldada.

Fonte: Autoria própria.

O cabo de alimentação, cordão umbilical, do robô com comprimento igual a um metro e meio foi elaborado também durante a etapa de montagem. Ao trabalhar com baixa corrente e com um sinal de 5V, geralmente, não é preciso se preocupar muito com a resistência e consequentemente com a queda de tensão gerada por fios com poucos metros de comprimento. Entretanto, em um circuito com cabos de 1,5m onde podem passar até 5A de corrente é preciso se preocupar com a espessura dos cabos para evitar uma queda de tensão indesejada. Para verificar a resistência gerada de acordo com a espessura dos cabos foi utilizada a tabela de condutores em [87].

Com isso foram utilizados cabos de 18 AWG para os sinais de terra e cabos duplos de 18 AWG para a linha de 5V da fonte de alimentação. Como os motores trabalham de 4.8V a 6V e foram medidos 5.07V na saída da fonte, a queda de tensão não deve ultrapassar 0.22V para manter um faixa de segurança de pelo menos 0.05V. Como cada

par de patas utiliza uma linha de 5V e cada pata uma linha de GND, conforme explicado na seção “Fonte de alimentação“, é preciso calcular apenas a queda de tensão provocada para um par de patas. O circuito resultante da resistência dos fios do “cordão umbilical” em série com as patas pode ser visto abaixo na figura 49.

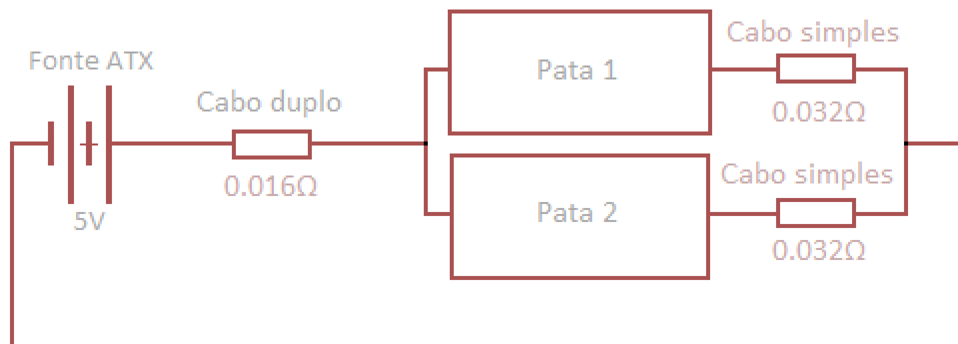


Figura 49: Circuito com as resistências geradas pelo "cordão umbilical".

Fonte: Autoria própria.

Para calcular a tensão resultante em cada fio basta utilizar a Lei de Ohm e substituir os valores pela resistência gerada por cada cabo. A corrente máxima no cabo utilizado na entrada de um par de pata é de 5A e na saída de cada pata é 2.5A. Portanto a queda de tensão provocada por conta dos cabos é de aproximadamente 0.16V, subtraindo esse valor dos 5.07V fornecidos pela fonte é possível encontrar a tensão final, de 4.9V, aplicada nos motores.

O cordão umbilical encerrou a etapa de montagem do Hexápode, processo que acabou não apresentando nenhum tipo de problema grave apesar das modificações necessárias. Uma visão da versão final do hexápode pode ser vista na figura 50.

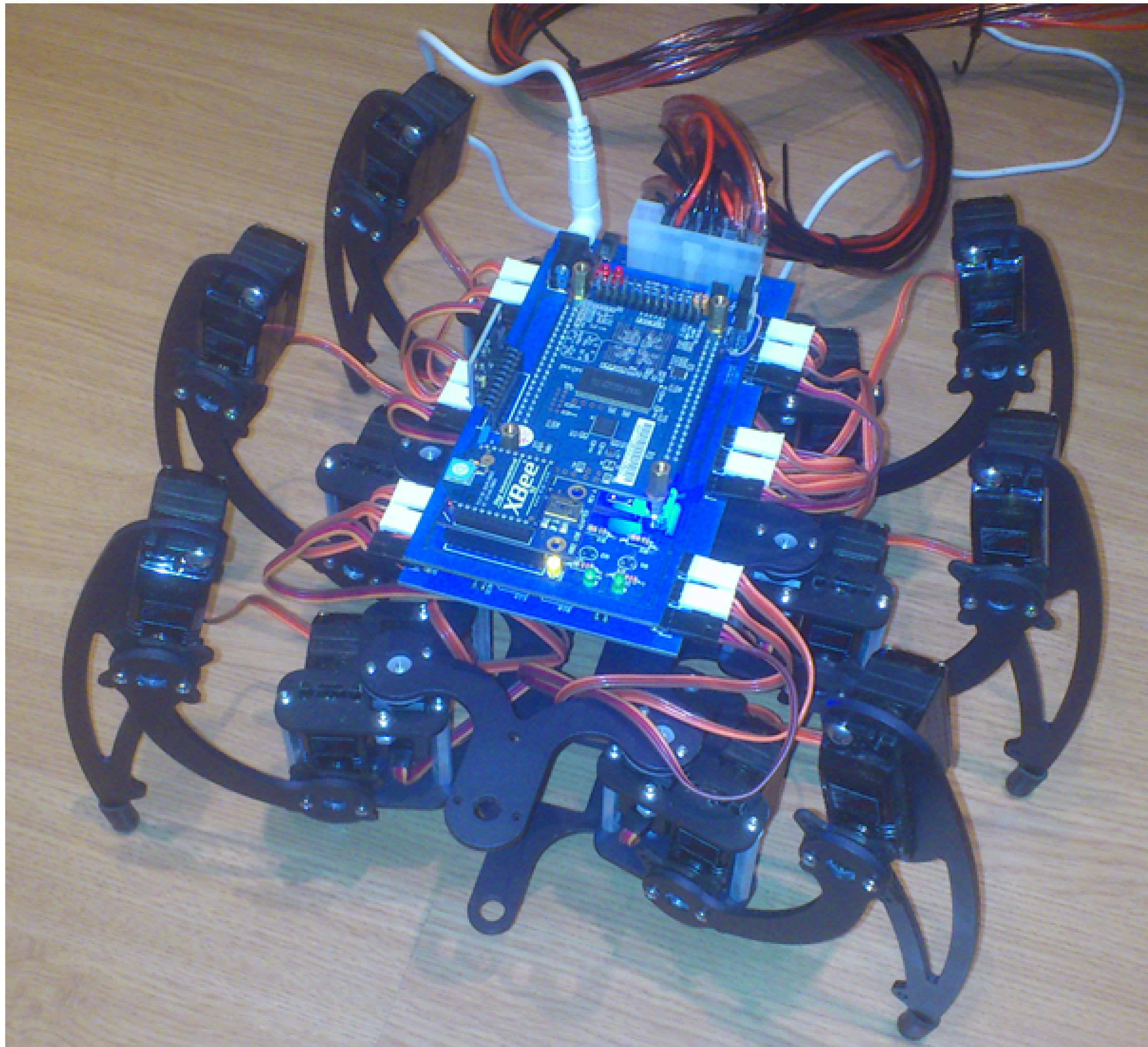


Figura 50: Hexápode completo.

Fonte: Autoria própria.

3.4 Protocolos de comunicação

Os mecanismos de comunicação entre o robô e o *driver* de estação base e comunicação entre o *driver* e o cliente de comando são baseados na troca de mensagens, como comumente utilizado em sistemas distribuídos[88]. Denomina-se de "protocolo de baixo nível" o sistema de troca de mensagens construído sobre os canais de comunicação.

A comunicação entre o robô e a estação base ocorre através de um canal serial ZigBee, enquanto a comunicação entre a estação base e o cliente de comando do robô ocorre através de *sockets* TCP. O sistema de troca de mensagens foi projetado e implementado sobre um fluxo de dados genérico, que pode representar tanto um canal de

UART quanto um *socket*. A vantagem dessa implementação genérica é que o mesmo código pode ser utilizado em ambos os casos, reduzindo a complexidade total do software. Outro lado positivo de tal design é que, em versão futuras do sistema, o mesmo protocolo poderia ser utilizado sobre canais diferentes de comunicação como, por exemplo, *sockets* UDP.

Os protocolos de alto nível são construídos sobre o protocolo de baixo nível. Cada um dos protocolos de alto nível suporta alguns tipos de mensagens. Há um protocolo entre o robô e o *driver* e outro entre o *driver* e o cliente de comando. Em ambos os casos, eles foram elaborados para incluir três tipos fundamentais de mensagens: instruir o robô a realizar um certo movimento, obter as leituras dos sensores do robô e notificar que um movimento foi terminado.

3.4.1 Protocolo de baixo nível

O protocolo de baixo nível é baseado no conceito de mensagens, que são pacotes indivisíveis de dados que transmitem algum tipo de informação[88]. Cada mensagem tem um tipo, um identificador único e um corpo. O tipo de uma mensagem define que tipo de informação é carregada em seu corpo e de que forma ela está representada, enquanto o identificador é um número que identifica unicamente a mensagem. É necessário que ambos os lados da comunicação utilizem intervalos distintos para geração de seus identificadores, de forma a evitar colisões.

3.4.1.1 Mensagens

Apesar de ambos os canais utilizados (ZigBee e TCP) terem mecanismos próprios de verificação e validação dos dados, que são transparentes para o programador[89,90], experimentos mostraram que há perda ocasional de bytes no canal ZigBee. Assim sendo, foi necessário adotar um mecanismo próprio de verificação e validação dos dados, de forma que uma mensagem incompleta seja identificada como inválida e descartada. A opção adotada foi empregar dois *checksums* simples e um terceiro valor para verificar consistência, todos enviados como parte da mensagem. Um dos *checksums* envolve a soma de todos os bytes do cabeçalho, enquanto o outro envolve a soma de todos os bytes do corpo da mensagem. O terceiro valor é o resultado de uma operação de ou-exclusivo entre os dois *checksums*, para garantir que o cabeçalho esteja consistente.

O formato de cada mensagem é apresentado na figura 51. A *magic word* tem quatro bytes e usada para identificar o começo de uma mensagem. Segue o cabeçalho, que

apresenta o tipo da mensagem, seu identificador, seu tamanho e os três campos de verificação (*checksums* e o campo de ou-exclusivo), todos com dois bytes de tamanho. Após todo o cabeçalho, é enviado o corpo da mensagem, que tem o número de bytes indicado no tamanho. O corpo da mensagem é a parte que carrega de fato a informação. A "*magic word*" é configurável, e é definida na especificação de cada protocolo de alto nível.

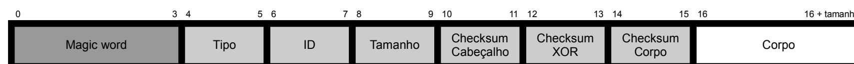


Figura 51: Formato de uma mensagem do protocolo de baixo nível.

Fonte: Autoria própria.

O envio de bytes pelo fluxo de dados é feito de forma bloqueante, mas o recebimento é feito de forma bloqueante com *timeout*. Desta forma, evitam-se problemas decorrentes de perda de bytes ou desligamento inesperado de um dos entes comunicantes, tornando o sistema mais tolerante a falhas e, portanto, mais robusto.

3.4.1.2 Controle de fluxo

Junto ao sistema de envio de mensagens é construído um sistema de controle de fluxo que gerencia requisições e respostas. Para cada mensagem de requisição enviada, espera-se receber uma resposta, que confirma o recebimento pelo outro lado da comunicação. O sistema de controle de fluxo é assíncrono, envolvendo o disparo de eventos ao recebimento de requisições e respostas, ou quando a resposta a uma mensagem não é recebida dentro do tempo esperado.

Estabeleceu-se uma convenção para o controle de fluxo: cada mensagem de resposta deve ter o mesmo identificador da requisição correspondente, mas um tipo diferente. Ainda convencionou-se que mensagens cujos tipos são números pares são de requisição e mensagens cujos tipos são números ímpares são de resposta.

Quando é perdido um byte na comunicação é possível que o próprio cabeçalho da mensagem tenha sido corrompido, então é preciso descartar a mensagem inteira. Como o cabeçalho pode ter sido corrompido, o tamanho da mensagem é desconhecido, já que tal informação é carregada no cabeçalho. Ainda assim, é preciso garantir que sejam descartados apenas os bytes daquela mensagem, recebendo com sucesso a mensagem seguinte. Só é possível estabelecer tal compromisso se forem utilizadas temporizações consistentes tanto para os tempos limite de transmissão de bytes quanto para os de

espera por respostas. Tais temporizações são condicionadas à latência e à velocidade do meio de transmissão sobre o qual o protocolo for utilizado, então cada protocolo de alto nível define temporizações próprias.

3.4.2 Protocolos de alto nível

Em ambos os protocolos de alto nível, as principais funções das mensagens são enviar comandos para o robô, ler seus sensores e notificar fim de movimento. A maneira que são feitos o envio de comandos e a leitura de sensores foi concebida de forma a permitir a inclusão de novas ações e sensores a uma futura versão do robô, sem necessitar uma alteração no protocolo. Então, em ambos os casos, as respostas às requisições podem indicar que a operação foi realizada com sucesso ou que o comando/sensor requisitado não é suportado.

As ações que o robô é capaz de realizar são denominadas “movimentos”. Cada movimento tem um tipo e deve receber um certo número de parâmetros, que caracterizam como ele será realizado. O tipo é um número inteiro de 2 bytes (16 bits), de forma que o robô pode suportar pouco mais de 65 mil movimentos distintos. Existe ainda um campo do movimento que é definido no alto nível, o “identificador da ação”. Esse identificador não tem utilidade nenhuma para o robô em si, e não deve afetar de forma alguma o movimento. Sua única função é facilitar a identificação de qual ação foi terminada quando uma notificação é recebida. O identificador é um número inteiro de 4 bytes (32 bits), podendo assumir mais de 4 bilhões de valores possíveis. A lista de movimentos que o robô pode realizar e seus respectivos parâmetros pode ser encontrada no apêndice A.

Cada movimento tem um “tamanho” e um “valor”, responsável por indicar quanto do movimento já foi realizado. O tamanho do movimento geralmente é determinado pelos parâmetros que ele recebe. Por exemplo, em um movimento de andar em frente o tamanho é o número total de passos e o valor é o número de passos andados até certo instante. O comportamento esperado é que o valor seja inicialmente zero e que se aproxime do tamanho à medida que o movimento vai sendo realizado. Quando o movimento terminar com sucesso, espera-se que seu valor seja igual ao seu tamanho.

Cada sensor ligado ao robô tem um identificador, denominado “tipo”. O tipo de sensor é um número inteiro de 16 bits, de forma que o protocolo permite a leitura de até 65 mil tipos diferentes de sensores. A lista de sensores suportados pelo robô e o formato das respectivas leituras pode ser encontrada no apêndice B.

3.4.2.1 Protocolo entre robô e driver

Como é ocasional a perda de bytes no canal de comunicação entre o robô e o *driver*, o protocolo de alto nível utilizado neste meio foi elaborado de forma a evitar resultados indesejados em caso de mensagens serem retransmitidas. As mensagens deste protocolo de alto nível foram concebidas para realizarem operações simples. Escolheu-se 0x000FF0FF (1044735 em decimal) como *magic word* deste protocolo, cujas mensagens devem seguir o formato apresentado na figura 52.

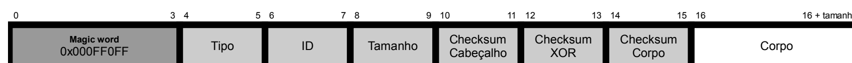


Figura 52: Formato geral de mensagens do protocolo entre o robô e o driver.

Fonte: Autoria própria.

As características do canal serial entre o robô e o *driver* são: *baud rate* de 57600, um *stop* bit e sem paridade. O tempo limite para recebimento de cada byte de uma mensagem foi definido em 4 milissegundos, e o limite para recebimento da resposta a uma requisição, em 128 milissegundos. Para evitar que atrasos no tratamento das mensagens dificultem a comunicação, esses valores foram definidos bastante acima do valor limite - idealmente, a transmissão pode atingir mais de 5 kB/s a 57600 bauds/s. Nesta seção, apresenta-se uma visão geral do protocolo, enquanto a documentação detalhada de todas as mensagens deste protocolo e seus respectivos parâmetros pode ser encontrada no apêndice C.

Neste protocolo, a operação de comandar o robô a realizar alguma ação é dividida em duas etapas. Primeiramente, define-se qual o movimento seguinte e em seguida, determina-se que o robô deve começar a executá-lo. Quando o movimento do robô é iniciado o campo interno que armazena a ação seguinte é limpo. Esta forma foi escolhida, em detrimento de usar uma única mensagem de “movimentar”, por garantir que o mesmo movimento não é iniciado duas vezes em caso de retransmissão, pois a mensagem de iniciar movimento não tem efeito quando não há ação seguinte.

A mensagem de definir movimento tem tipo 0x0022, e seu corpo inclui o tipo de ação, seu identificador e seus parâmetros. Seu formato pode ser visualizado na figura 53. Se o movimento for suportado, a mensagem de resposta tem tipo 0x0023, do contrário, tem tipo 0x0021. O corpo da resposta é vazio em ambos os casos. A mensagem de iniciar movimento tem tipo 0x0024 e não recebe parâmetros.

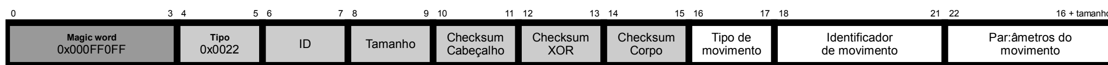


Figura 53: Formato da mensagem de definir movimento para o robô.

Fonte: Autoria própria.

A mensagem de leitura dos sensores recebe como parâmetro um código identificador do sensor. Caso o sensor não seja suportado, a resposta a essa mensagem é vazia. Do contrário, os dados lidos do sensor são enviados no corpo da mensagem de resposta. A requisição de leitura de sensores tem tipo 0x0040 e segue o formato apresentado na figura 54. A leitura do sensor retornada na mensagem de resposta não é necessariamente realizada no momento em que a resposta foi enviada, ela pode ter ocorrido em até um segundo antes.

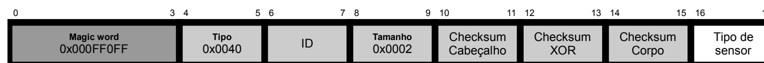


Figura 54: Formato da mensagem de ler sensor para o robô.

Fonte: Autoria própria.

O robô pode enviar para o *driver* mensagens de notificação, indicando que um movimento foi terminado com sucesso ou que foi abortado. O movimento é abortado quando uma mensagem de parada é enviada para o robô enquanto ele estiver realizando um movimento. A notificação de movimento terminado retorna, também, o tamanho do movimento. A notificação de movimento abortado indica o valor do movimento no instante em que foi interrompido. O formato dessas mensagens de notificação é apresentado nas figuras 55 e 56.

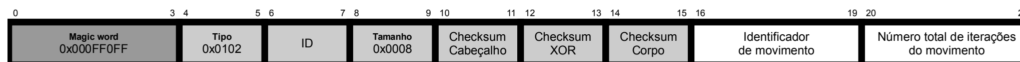


Figura 55: Formato da mensagem de movimento terminado com sucesso.

Fonte: Autoria própria.

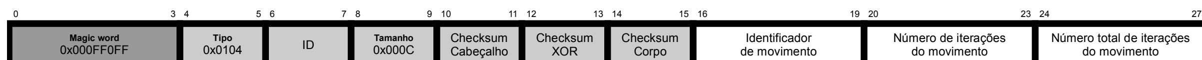


Figura 56: Formato da mensagem de movimento abortado.

Fonte: Autoria própria.

3.4.2.2 Protocolo entre cliente e driver

Como o canal TCP garante a transmissão de todos os dados de uma ponta a outra, o protocolo de alto nível entre o cliente e o *driver* não requer o mesmo cuidado com a perda de bytes, mas, em contrapartida, requer um cuidado maior com latência na transmissão. Dessa forma, os tempos limite de recebimento de bytes e de respostas neste protocolo são maiores do que os tempos limite no protocolo entre o *driver* o robô. O tempo limite para recebimento de um byte é de 500 ms e o tempo limite de recebimento de respostas a mensagens varia entre 1,5 a 5 segundos, dependendo do tipo de mensagem. Essas temporizações são possivelmente exageradas, já que a latência em redes locais (LANs) geralmente não passa de 10 ms e em rede pública (internet) oscilam entre 100 ms e 500 ms[91]. A porta na qual o *driver* espera por conexões é 0xEDDA (60890 em decimal).

A *magic word* do protocolo entre cliente e *driver* é 0x00BA0BAA (12192682 em decimal). Nesta seção, apresenta-se uma visão geral do protocolo, enquanto a documentação detalhada de todas as mensagens deste protocolo e seus respectivos parâmetros pode ser encontrada no anexo D.

O *driver* inicialmente não está conectado ao robô, então é preciso enviar-lhe um comando para conecta-lo. Existem duas mensagens associadas a esta operação, uma cuja função é retornar a lista de portas (seriais) disponíveis no computador no qual o *driver* é executado, e outra cuja função é comandar o estabelecimento da conexão com o robô através de alguma porta. A mensagem de enumerar a portas tem código 0x8000, com o corpo vazio. A resposta a esta mensagem tem o formato apresentado na figura 57. Seu corpo é a lista de nomes das portas, separadas por carácter “newline” (0x0a), e codificadas usando *charset* ASCII. A mensagem de abrir conexão com o robô através de uma porta tem código 0x8002 e seu formato é apresentado na figura 58. O corpo da resposta a esta mensagem é um byte com um código indicando o resultado da operação de abertura.

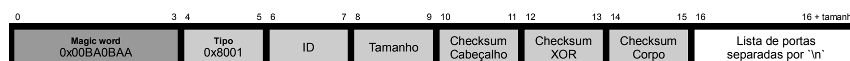


Figura 57: Formato da mensagem de enumerar portas do cliente.

Fonte: Autoria própria.

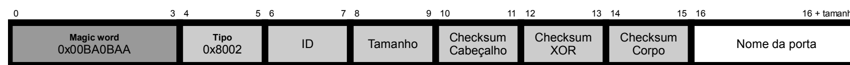


Figura 58: Formato da mensagem de abrir porta do cliente.

Fonte: Autoria própria.

Uma vez que a conexão com o robô esteja aberta passa a ser possível enviar comandos para o robô. Não é possível fechar a conexão com o robô sem fechar a conexão com o *driver*. A operação de comandar o robô a realizar alguma ação, diferentemente do que acontece no protocolo entre o *driver* e o robô, não é dividida em duas etapas. O *driver* recebe o comando de movimentar em uma única etapa, e faz a negociação em duas etapas com o robô. O formato da mensagem associada a essa operação é o mesmo da requisição de definir movimento no protocolo entre o *driver* e o robô, mas tem código 0x8080, tal qual apresentado na figura 59. A resposta desta requisição tem código 0x8081 e seu corpo contém um código indicando o resultado da operação.

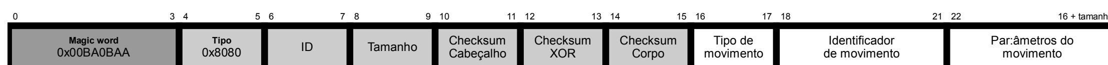


Figura 59: Formato da mensagem de movimentar do cliente.

Fonte: Autoria própria.

Da mesma forma, a mensagem de leitura de sensores também tem formato semelhante à utilizada no protocolo entre o robô e o *driver*, mas com código 0x80C0. A resposta, ao contrário do que acontece no outro protocolo, inclui um código indicando o resultado da operação. O formato da mensagem de requisição é apresentado na figura 60.

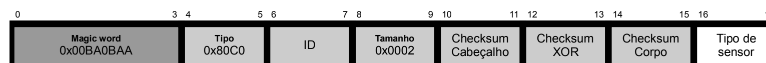


Figura 60: Formato da mensagem de ler sensor do cliente.

Fonte: Autoria própria.

As notificações enviadas pelo *driver* ao cliente também seguem um formato semelhante às utilizadas no protocolo entre o robô e o *driver*, mudando apenas seus identificadores. Ambas estão representadas nas figuras 61 e 62.

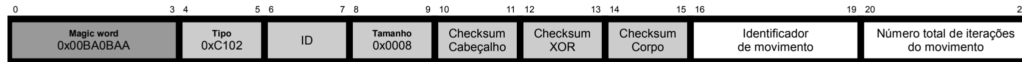


Figura 61: Formato da mensagem de movimento terminado com sucesso.

Fonte: Autoria própria.

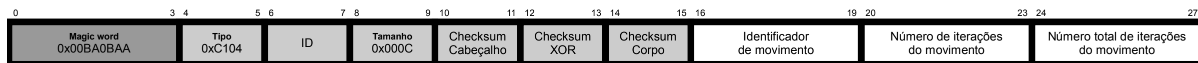


Figura 62: Formato da mensagem de movimento abortado.

Fonte: Autoria própria.

3.5 Software

Há três artefatos de software principais do projeto: o *firmware,* isto é, o software embarcado, o software do *driver* de comunicação com o robô e a biblioteca de comunicação com o *driver*. Além disso, foi criado um quarto software com uma interface gráfica de exemplo, que utiliza a biblioteca de comunicação. Todo o código foi desenvolvido utilizando orientação a objetos e de forma modular, para reduzir o esforço necessário se futuramente houver o interesse de incluir novas funcionalidades. O *firmware* foi desenvolvido em linguagem C++, enquanto o *driver* de comunicação com o robô, a biblioteca e o software de interface gráfica foram desenvolvidos em linguagem Java, versão 6 ou superior.

3.5.1 Projeto de software de interface gráfica

3.5.1.1 Requisitos funcionais

SWRF1. O software deverá comunicar-se com o robô através da biblioteca de comunicação.

SWRF2. O software devera permitir que o usuário transmita comandos de locomoção ao robô.

SWRF3. O software deverá exibir, quando possível, informações dos sensores do robô.

3.5.1.2 Requisitos não-funcionais

SWRNF1. O software deverá ser utilizado através de mouse e teclado

SWRNF2. O software deverá ter partes de seu código-fonte, quando cabível, distribuídas sob licença BSD.

SWRNF3. O software da interface gráfica deverá ser executado em computadores com no mínimo sistema operacional Windows XP.

SWRNF4. O software não precisa contemplar todos os comandos que o robô pode executar.

3.5.1.3 Casos de uso

O software de interface gráfica inclui dois atores: o próprio software e o usuário. O robô não foi incluso pois não interage diretamente com nenhum dos demais atores. Há apenas dois casos de uso do software de interface, referentes às ações de enviar um comando para o robô e ler seus sensores. O diagrama de casos de uso é apresentado na figura 63.

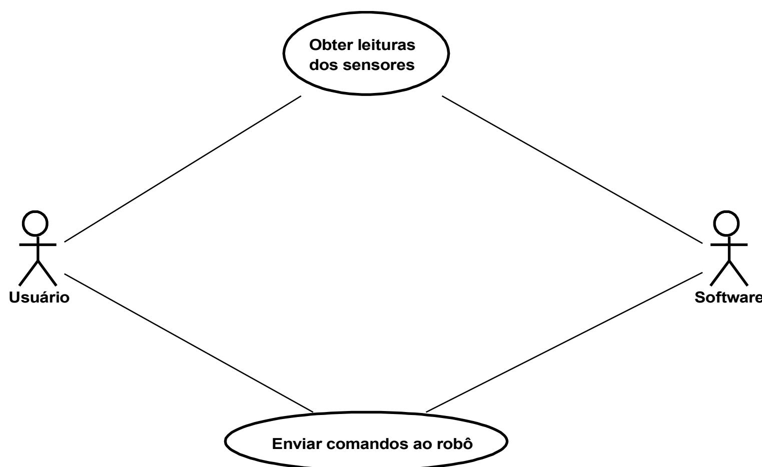


Figura 63: Diagrama de casos de uso.

Fonte: Autoria própria.

3.5.1.3.1 Enviar comandos ao robô

O caso de uso "enviar comandos ao robô" é referente à transmissão de comandos ao robô, através do software. O caso de uso inclui a espera pelo término do movimento ou sua parada precoce pelo usuário.

- Atores:

◇ Usuário.

◇ Software.

● Fluxo de eventos:

◇ Fluxo básico:

1. Usuário seleciona uma aba na tela principal, associada ao comando.
2. Usuário digita, se necessário, parâmetros do comando.
3. Usuário pressiona o botão de iniciar movimento.
4. O software tenta enviar o comando através da biblioteca, se os parâmetros forem válidos,
5. O software exibe uma mensagem indicando que o movimento iniciou, se a biblioteca responder indicando com sucesso.
6. O software retorna à tela principal, assim que a biblioteca enviar notificação de movimento terminado.

◇ Fluxo alternativo:

1. Usuário seleciona uma aba na tela principal, associada ao comando.
2. Usuário digita, se necessário, parâmetros do comando.
3. Usuário pressiona o botão de iniciar movimento.
4. O software tenta enviar o comando através da biblioteca, se os parâmetros forem válidos,
5. O software exibe uma mensagem indicando que o movimento iniciou, se a biblioteca responder indicando com sucesso.
6. Usuário pressiona o botão de parar movimento.
7. O software retorna à tela principal, assim que a biblioteca enviar notificação de movimento abortado.

◇ Fluxo alternativo

1. Usuário seleciona uma aba na tela principal, associada ao comando.
2. Usuário digita, se necessário, parâmetros do comando.
3. Usuário pressiona o botão de iniciar movimento.
4. O software não faz nada, se os parâmetros forem inválidos.

◇ Fluxo alternativo

1. Usuário seleciona uma aba na tela principal, associada ao comando.
2. Usuário digita, se necessário, parâmetros do comando.
3. Usuário pressiona o botão de iniciar movimento.

4. O software tenta enviar o comando através da biblioteca, se os parâmetros forem válidos,
5. O software exibe uma mensagem indicando que o movimento não iniciou, se a biblioteca responder indicando falha.

- Pré-condições:
  - ◇ O software está em execução.
  - ◇ A biblioteca de comunicação do software está conectada a um driver, que já abriu a conexão com um robô.
- Pós-condições
  - ◇ Robô executa a ação desejada, ou a ação não é executada por algum motivo.

3.5.1.3.2 Obter leituras dos sensores

O caso de uso “obter leituras dos sensores” é referente à obtenção das leituras de um ou mais sensores do robô, através do software.

- Atores:
  - ◇ Usuário.
  - ◇ Software.
- Fluxo de eventos:
  - ◇ Fluxo básico
    1. Usuário pressiona o botão de atualizar leitura de um dos sensores, na tela principal.
    2. Os dados exibidos são atualizados, se a biblioteca responder indicando sucesso..
  - ◇ Fluxo alternativo:
    1. Usuário pressiona o botão de atualizar leitura de um dos sensores, na tela principal.
    2. Os dados exibidos não são atualizados, se a biblioteca responder indicando falha..
- Pré condições
  - ◇ O software está em execução.
  - ◇ A biblioteca de comunicação do software está conectada a um driver, que já abriu a conexão com um robô.
- Pós condições

◇ Os dados sobre o sensor na tela são atualizados.

3.5.2 Detalhes da implementação em C++

3.5.2.1 Alocação de memória

Os artefatos de software desenvolvidos em linguagens C++ e Java foram elaborados de forma a preservarem grande grau de similaridade mútua. Assim sendo, o código em linguagem C++ naturalmente empregaria alocação dinâmica de memória, como ocorre no software desenvolvido em Java. Entretanto o software em C++ é embarcado e em sistemas deste tipo evita-se utilizar alocação dinâmica[56]. Entretanto, para preservar a consistência entre as duas implementações do software, adotou-se o paradigma de “object pooling”.

O sistema de object pooling utilizado no software em C++ é de autoria própria. Nele, as pools de objetos são implementadas como listas encadeadas. De forma ao object pooling ser totalmente transparente para o programador, considerou-se a possibilidade de sobrescrever os operadores “new” e “delete”, mas esta opção foi descartada pois pode gerar uma série de complicações[92]. Assim, adotou-se uma convenção própria para diferenciar a obtenção e inserção de objetos em uma pool da criação e do apagamento de objetos. A convenção envolve o emprego de um método estático denominado “*create*” em cada classe, para obter instâncias a partir da pool, e outro método denominado “finalize” para devolvê-las à pool. A convenção de empregar um método estático denominado “create” em lugar do construtor foi utilizado também no software Java, para preservar a similaridade entre as duas partes do software. As pools de objetos, no código C++, estão implementadas na classe genérica “ObjectPool<Count,Type>” (na qual “Count” é o total de objetos e “Type” é a classe dos objetos) .

3.5.2.2 Gerência de memória

A linguagem C++ é não-gerenciada e permite emprego de alocação dinâmica de memória, então programas desenvolvidos nela podem apresentar problemas quanto a vazamentos de memória e *dangling pointers*. Para evitar tais problemas, e preservar a similaridade com o software desenvolvido em Java, empregou-se no software em C++ um sistema automático de gerenciamento de memória.

No software desenvolvido em C++ optou-se por empregar contagem de referências. Nele, há duas classes: “StrongReference<T>” e “WeakReference<T>” (utilizadas através das macros “_strong(T)” e “_weak(T)”), que implementam, respectivamente, referências fortes e fracas a objetos. Há ainda uma classe denominada “ControlBlock”, que representa o “bloco de controle” de um objeto, que contém seu contador de referências. Como se utilizou um mecanismo automático de gerência de memória, nenhum objeto é apagado diretamente a partir do código de alto nível, o que acaba por ser mais um ponto de similaridade entre o software em C++ e o em Java.

Para evitar referências circulares, em alguns locais do código C++ utilizaram-se referências fracas. Para preservar a similaridade entre as implementações, os trechos de código que empregam referências fracas em C++ também o fazem em Java. Nota-se, entretanto, que não é realmente necessário utilizar referências fracas para essa finalidade no código em Java, pois o *garbage collector* da JVM é capaz de detectar e tratar referências circulares.

3.5.3 Código de baixo nível

O código de alto nível é responsável pelo controle do robô, comunicação, entre outros. Como há quatro artefatos de software, em mais de uma plataforma, buscou-se desenvolver o código de alto nível de forma independente da arquitetura. Como consequência, diversas partes do código foram programadas uma única vez, mas utilizadas em mais de uma versão do software. De forma a tornar este procedimento possível, foi preciso que houvesse uma forma portável de realizar operações de baixo nível, como criação de *threads*, operações com semáforos, entre outras. Com esta finalidade, foi criada uma biblioteca de baixo nível orientada a objetos. Apesar de tal biblioteca ter uma implementação diferente em cada plataforma, sua especificação é única.

Na biblioteca de baixo nível, por possuir chamadas bloqueantes, contribui para que ocorram situações de *deadlock* ou de espera indeterminada. Para evitar tais problemas, as chamadas da biblioteca de baixo nível, quando conveniente, são não-bloqueantes ou bloqueantes com *timeout*. Um diagrama completo com todas as classes de baixo nível é apresentado na figura 64.

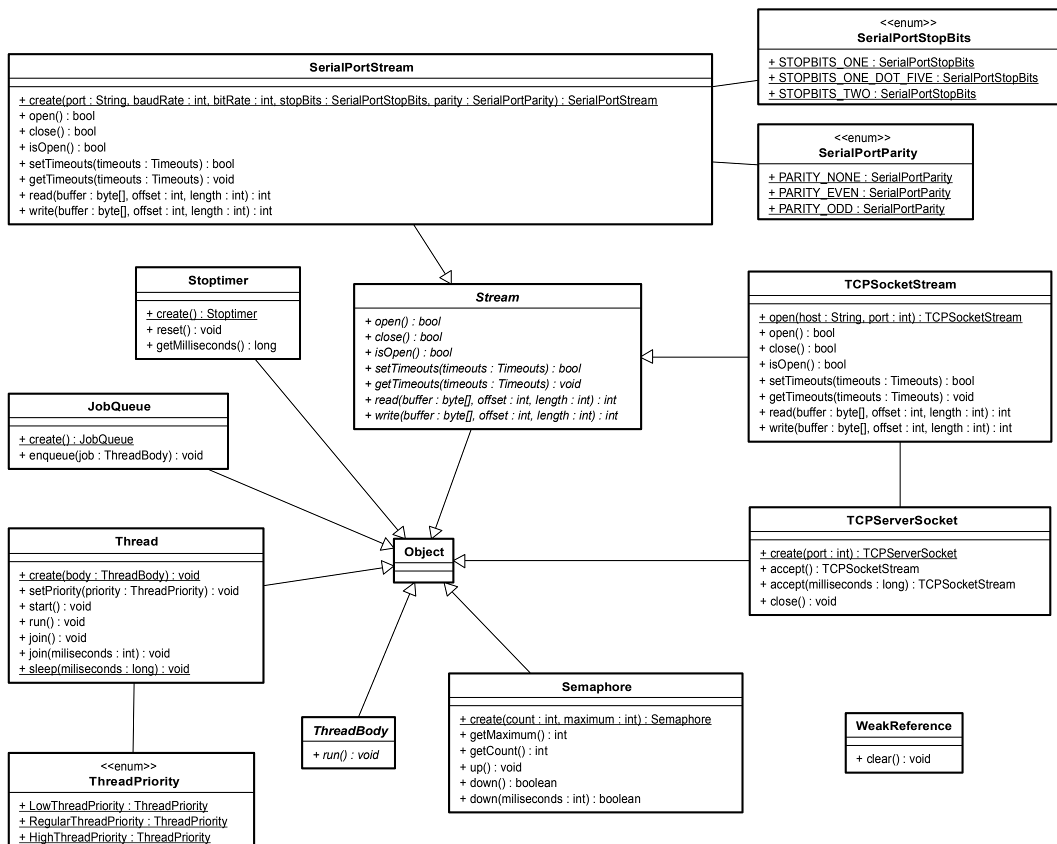


Figura 64: Diagrama de classes de baixo nível.

Fonte: Autoria própria.

As classes de baixo nível são divididas em três pacotes (namespaces em C++): “base”, “*stream*” e “*concurrent*”. O pacote “base” contém a classe “*Object*”, a qual todos os objetos estendem. No código em C++, as classes que representam referências fracas e *pools* de objetos também são parte do *namespace* “base”; tais classes não aparecem no diagrama de classes, pois são específicas da versão C++. Já os pacotes “stream” e “concurrent” contém, respectivamente, classes referentes a fluxos de dados (UART e sockets) e referentes a operações concorrentes, como semáforos, threads, temporizadores, entre outros.

A biblioteca foi implementada em Java e em C++. Na API da linguagem Java, existem algumas classes equivalentes ou muito semelhantes às definidas na biblioteca de baixo nível; neste caso, foram utilizadas as classes da API em detrimento das da biblioteca. A implementação em C++ foi feita para duas plataformas: primeiramente, foi feita para Windows XP em arquitetura x86, o que foi utilizado apenas para desenvolvimento, e depois foi feita para MicroC/OS-II sobre processador NIOS II, o que é utilizado no firmware.

A interação entre a biblioteca de baixo nível do código Java e a porta serial é feito através da biblioteca “nrjavaserial”, liberada sob licença LGPL[93]. A nrjavaserial é um fork da biblioteca RxTx, com a qual os membros da equipe já tinham familiaridade ao começo do projeto[11,25]. Em C++, é usado para interagir com a UART Avalon um driver desenvolvido pela própria Altera.

3.5.4 Código de alto nível

A implementação das classes de alto nível é independente de plataforma, exceto as classes que realizam movimentação e leitura de sensores no firmware. Assim, via de regra, as operações de baixo nível utilizadas são apenas aquelas implementadas pelas classes de baixo nível.

Algumas classes de alto nível foram projetadas para suas instâncias enviarem notificações para outros objetos quando detectam algum evento que requer tratamento. Um exemplo de situação em que isto é útil: um objeto representando um canal de comunicação recebe uma mensagem, e envia uma notificação para outro objeto, que a recebe trata a mensagem recebida. O paradigma adotado para implementar esta mecânica foi a utilização de callbacks. Esse paradigma envolve o uso de classes abstratas ou interfaces que definem métodos denominados “callbacks”, os quais são invocados toda a vez que houver uma nova notificação. É possível que qualquer classe estenda as clases abstratas ou implemente as interfaces que definem os callbacks, de forma que objetos projetados de forma independente possam registrar-se para receber notificações.

3.5.4.1 Protocolo

As classes do protocolo de comunicação são utilizadas nos três artefatos de software e implementam todas as operações associadas ao envio e ao recebimento de mensagens através de um fluxo de dados. Essas operações são referentes apenas ao protocolo de baixo nível, isto é, ao mecanismo de requisição-resposta. O protocolo de alto nível é implementado diferentemente em cada um dos artefatos de software. O diagrama com as classes de protocolo pode ser visualizado na figura 65.

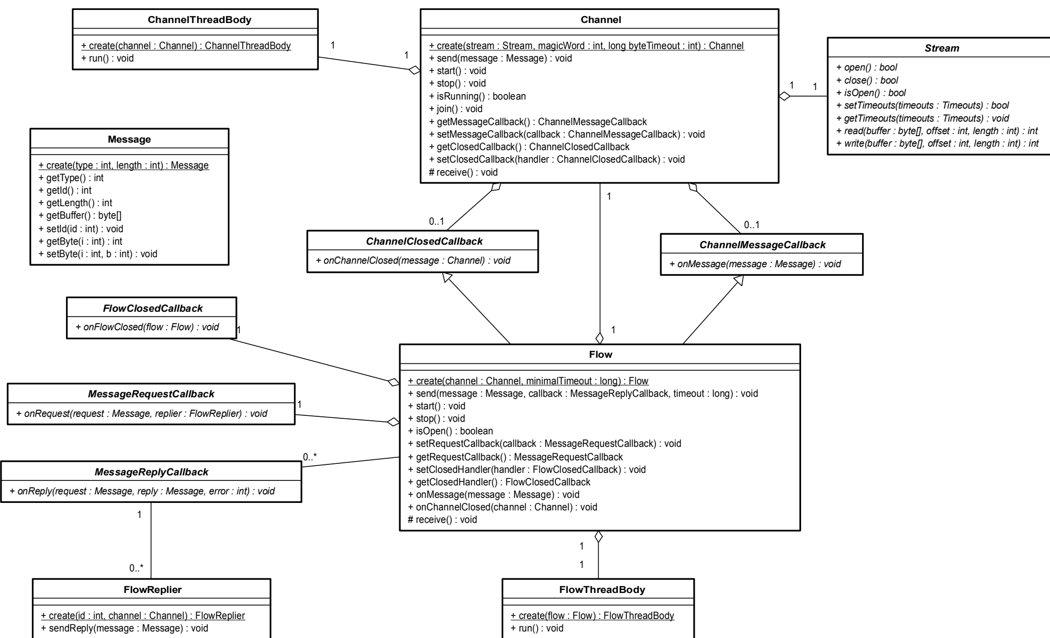


Figura 65: Diagrama de classes de protocolo.

Fonte: Autoria própria.

Todas as classes do protocolo de comunicação encontram-se no pacote (namespace em C++) “protocol”. Há três classes principais neste pacote: “Message”, “Channel” e “Flow”. A classe “Message” representa uma mensagem qualquer transmitida através do protocolo e tem três propriedades principais: o identificador, o tipo e o corpo. A classe “Channel” representa um canal de troca de mensagens construído sobre um fluxo de dados. Essa classe é responsavel por enviar e receber mensagens. A classe “Flow” representa um fluxo de troca de mensagens construído sobre um canal. Essa classe é responsável por fazer matching de requisições e respostas e verificar que requisições sofreram timeout. Em ambos os casos, as classes operam de forma assíncrona, então tanto a classe “Channel” quanto a classe “Flow” disparam notificações através de métodos callback.

3.5.4.2 Firmware

O *firmware* é executado sobre um processador NIOS II com sistema operacional MicroC/OS-II, e a linguagem de programação utilizada para desenvolvê-lo foi C++. O fato de as classes de alto nível serem quase inteiramente independentes da plataforma possibilitou que grande parte do *firmware* fosse desenvolvida em uma outra plataforma na qual o desenvolvimento fosse mais cômodo. As classes de baixo nível foram implementadas para ambiente Windows XP sobre processador x86, plataforma na qual foi

desenvolvido o *firmware*. Assim que o código atingiu um certo grau de maturidade, implementaram-se as classes de baixo nível para MicroC/OS-II sobre processador NIOS II. Na versão definitiva do *firmware*, há algumas operações de baixo nível que são realizadas diretamente nas classes de alto nível, como leitura de sensores e definir setpoints das patas. O diagrama de classes do *firmware* pode ser visualizado na figura 66.

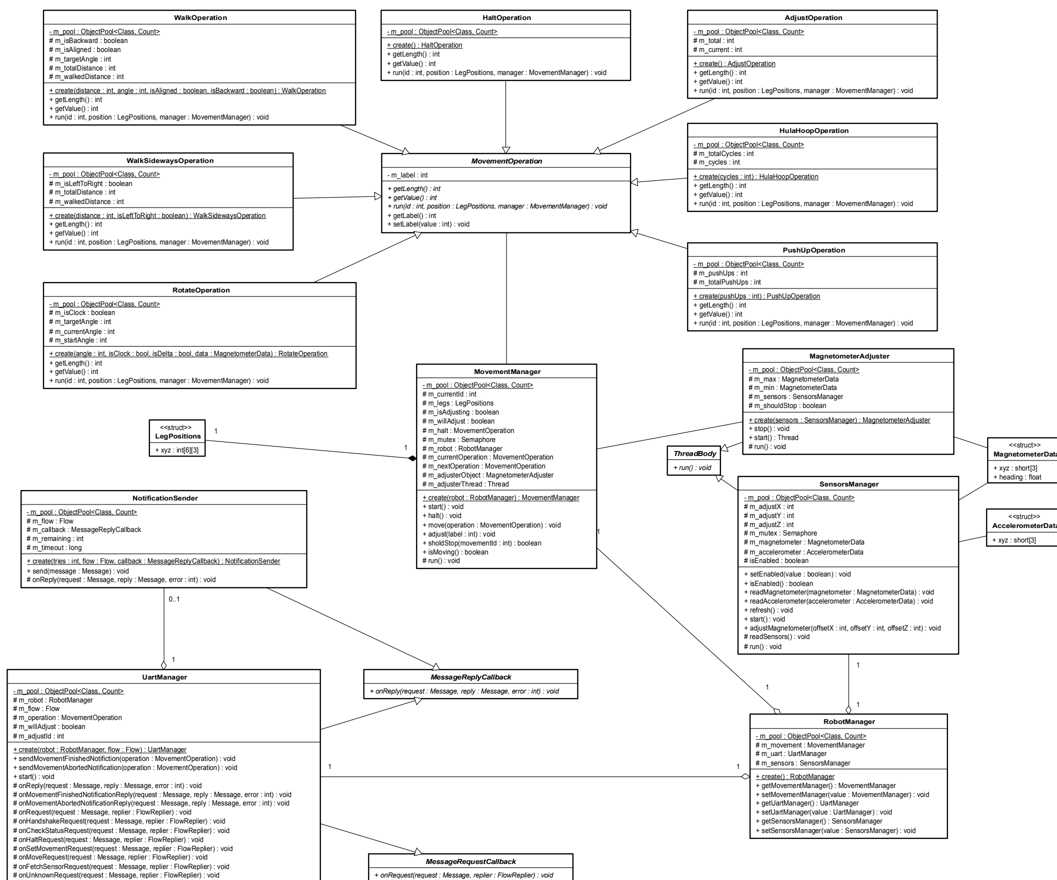


Figura 66: Diagrama de classes do firmware.

Fonte: Autoria própria.

O *firmware* foi feito de forma modular, com cada módulo responsável por um aspecto do funcionamento do robô. A cada um deles é associada uma classe, sendo a versão final composta de quatro módulos, um responsável pela integração entre os demais (classe “RobotManager”) e os outros três responsáveis por: movimentação (classe “MovementManager”), comunicação (classe “UartManager”) e leitura de sensores (classe “SensorsManager”). Uma visão geral do funcionamento do firmware, em forma de um statechart, é apresntada na figura 67.

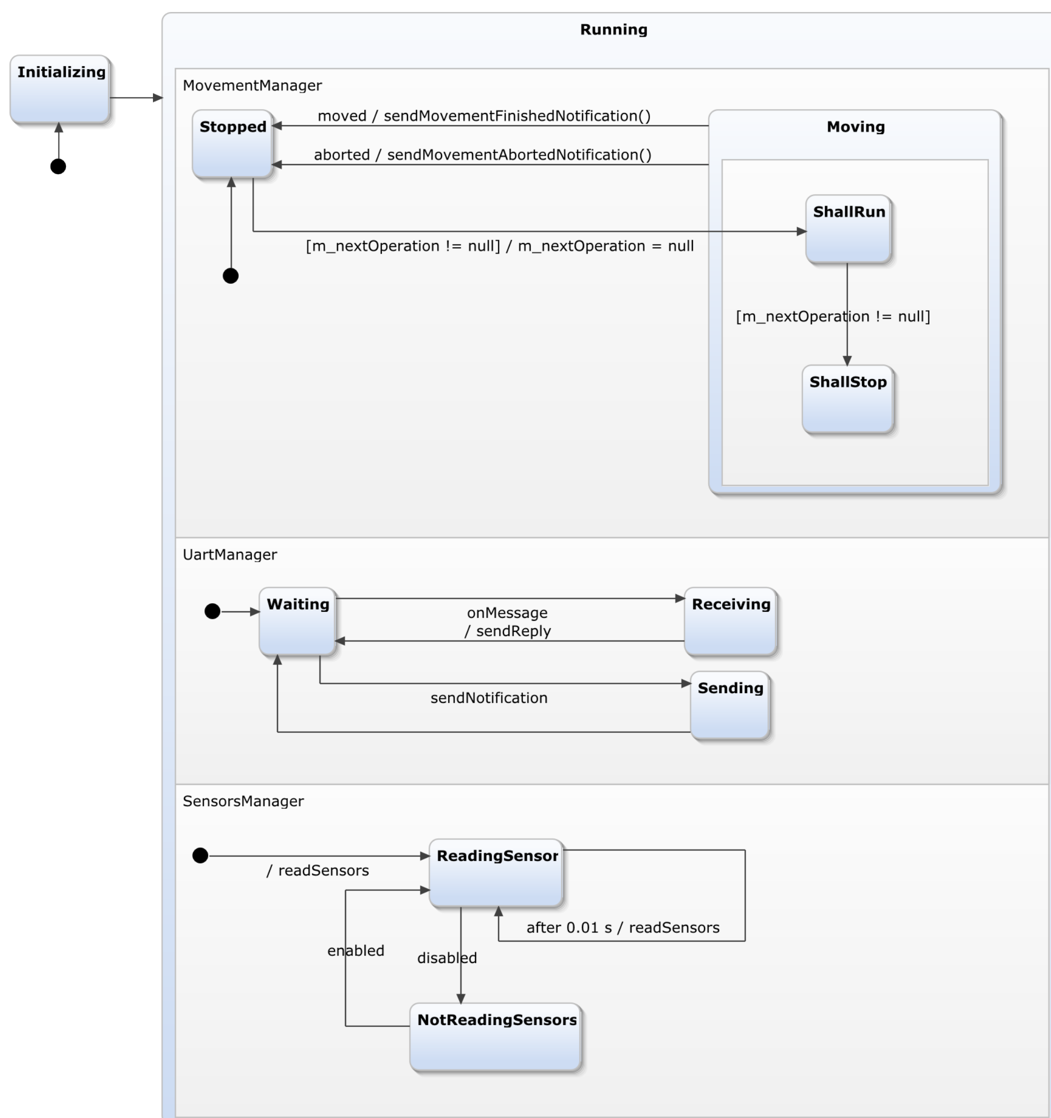


Figura 67: Statechart do firmware.

Fonte: Autoria própria.

O módulo de movimentação do robô, implementado pela classe MovementManager, gerencia o posicionamento das patas e as sequências de movimentos do robô. Ele inclui uma thread que é responsável por atualizar as posições das patas do robô durante uma operação de movimentação. Como o procedimento de cinemática inversa é feito por hardware, o módulo de movimentação apenas gerencia a posição desejada da ponta da pata e o hardware converte esta posição em PWM para os motores. A posição atual das seis patas do robô é representada por uma estrutura do tipo "LegPositions". Cada movimento que o robô pode realizar é implementado em uma classe distinta, que estende a classe abstrata MovementManager.

O módulo de leitura de sensores, implementado pela classe SensorsManager, gerencia a comunicação com os sensores. A classe mantém campos internos com leituras dos sensores, os quais são constantemente atualizados por uma *thread*. As operações de obter leituras dos sensores, expostas através dos métodos “readMagnetometer” e “readAccelerometer”, obtém o valor armazenado da leitura, não o valor instantâneo. As operações de comunicação com os sensores são bloqueantes, e consomem um certo tempo de processamento. Assim, a *thread* de leitura de sensores poderia interferir com as temporizações das demais operações, como no caso dos movimentos. A *thread* de leitura de sensores, por sua vez, é desativada durante o tempo de execução de um movimento e o próprio objeto responsável pela movimentação fica responsável por forçar o módulo dos sensores a atualizar suas leituras, o que é feito através do método “*refresh*”.

O módulo de comunicação, implementado pela classe UartManager, trata todas as mensagens recebidas do *driver* e também é responsável por enviar ao driver notificações de término de movimento. O envio de notificações é delegado a instâncias de uma outra classe, NotificationSender, que tenta reenviar as notificações caso detecte problemas na transmissão (até um número limite de tentativas). Esse número limite foi definido arbitrariamente em 8 tentativas, com 1 segundo de intervalo quando há um erro.

Para que os movimentos do robô fiquem mais suaves, a transição entre duas posições é decomposta em vários passos intermediários através de interpolação. Em diversos casos, a espera entre dois setpoints em um movimento interpolado deveria ser menor que o tick mínimo do MicroC/OS-II, que é de 10 ms. Para fazer esta espera havia duas possibilidades: utilizar espera ativa ou temporizadores externos e interrupções. Apesar de os temporizadores serem a melhor opção, utilizou-se espera ativa devido à simplicidade de implementá-la. Como o MicroC/OS-II executa sempre a tarefa não-bloqueada com maior prioridade, a tarefa de movimentação poderia impedir o restante do sistema de funcionar, pois em busy-wait, o processador nunca é explicitamente liberado pela tarefa. Para evitar este problema, a prioridade da tarefa de movimentação é mínima, entretanto isso raramente ocorre porque na maior parte do tempo as demais tarefas estão suspensas.

3.5.4.3 Driver

O driver é a peça-chave na integração do sistema. Ele é responsável por realizar a comunicação com o robô através de um fluxo de UART e por realizar a comunicação com

um cliente a partir de um canal TCP. Seu diagrama de classes pode ser visualizado na figura 68.

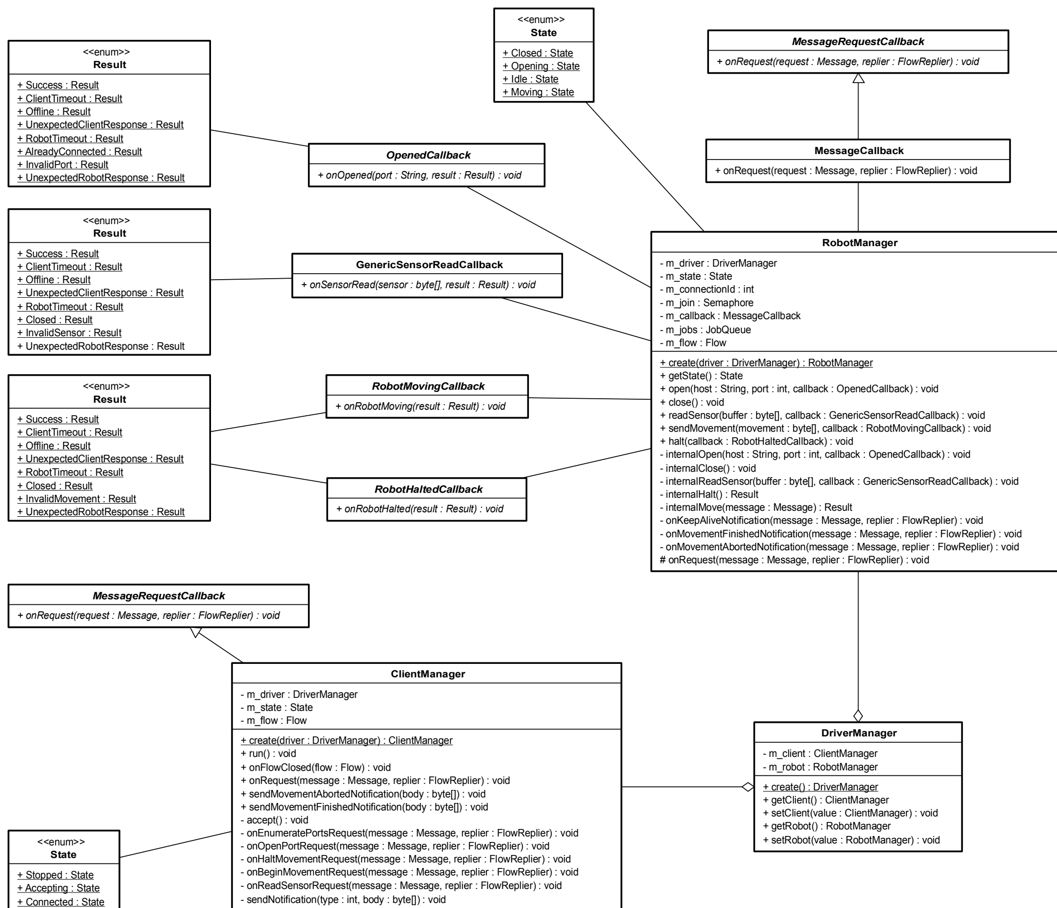


Figura 68: Diagrama de classes do driver.

Fonte: Autoria própria.

Da mesma forma que o firmware, o driver foi desenvolvido de forma modular, com cada módulo responsável por um aspecto distinto de seu funcionamento. O módulo central do driver é denominado “DriverManager” e faz a integração entre os dois outros módulos: um que gerencia a comunicação com o robô (classe “RobotManager”) e outro que gerencia a comunicação com o cliente (classe “ClientManager”). Uma visão geral do funcionamento do driver é apresentada, em forma de statechart, na figura 69.

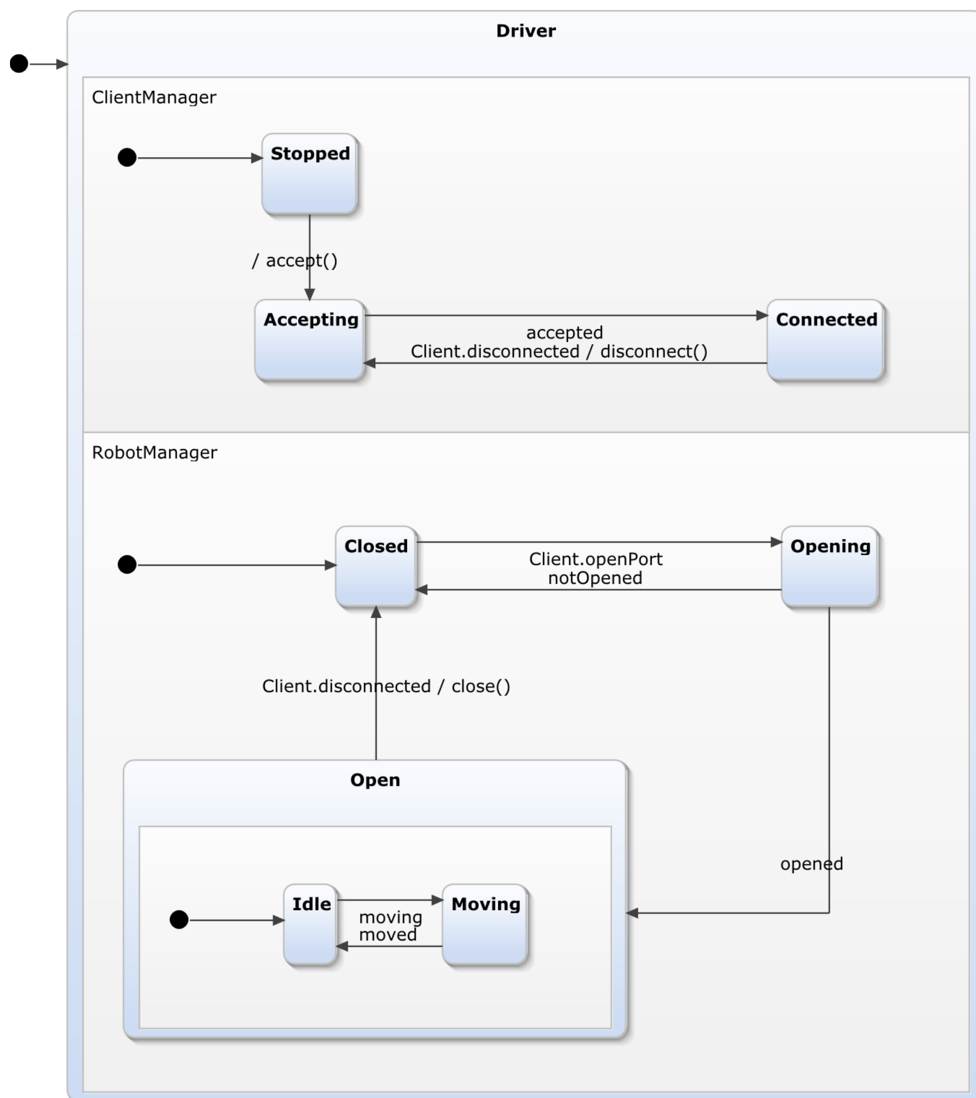


Figura 69: Statechart do driver.

Fonte: Autoria própria.

O módulo que gerencia a comunicação com o robô, implementado pela classe “RobotManager”, é responsável por estabelecer a conexão com o robô através de uma porta serial e coordenar todas as trocas de mensagens. Inicialmente, este módulo não está conectado a robô algum. Assim que o cliente envia uma requisição de abertura de porta é feita uma tentativa de conexão com o robô. Se tudo correr bem, o módulo passa para o estado “aberto”. Enquanto o cliente estiver conectado ao driver ele não pode fechar a conexão com o robô, então a única forma de fazer o RobotManager voltar para o estado “fechado” é desconectando o cliente do driver.

O módulo que gerencia a comunicação com o cliente, implementado pela classe ClientManager, aceita conexões em uma porta de rede. Assim que um cliente é aceito, ele

pode enviar comandos pelo canal TCP. Este módulo é responsável por tratar todas as mensagens de alto nível do cliente e invocar os métodos adequados do RobotManager. Quando o cliente é desconectado, este módulo volta a aceitar conexões, e o RobotManager é instruído a fechar a porta aberta para comunicação com o robô.

3.5.4.4 Cliente

O cliente é o elemento menos modular dentre os três artefatos principais de software. Ele foi feito desta forma pois se comporta como uma “interface” entre um outro software (por exemplo, uma interface gráfica) e o driver, então é mais conveniente que seja utilizado através de chamadas simples de métodos. O diagrama de classes do cliente pode ser visualizado na figura 70.

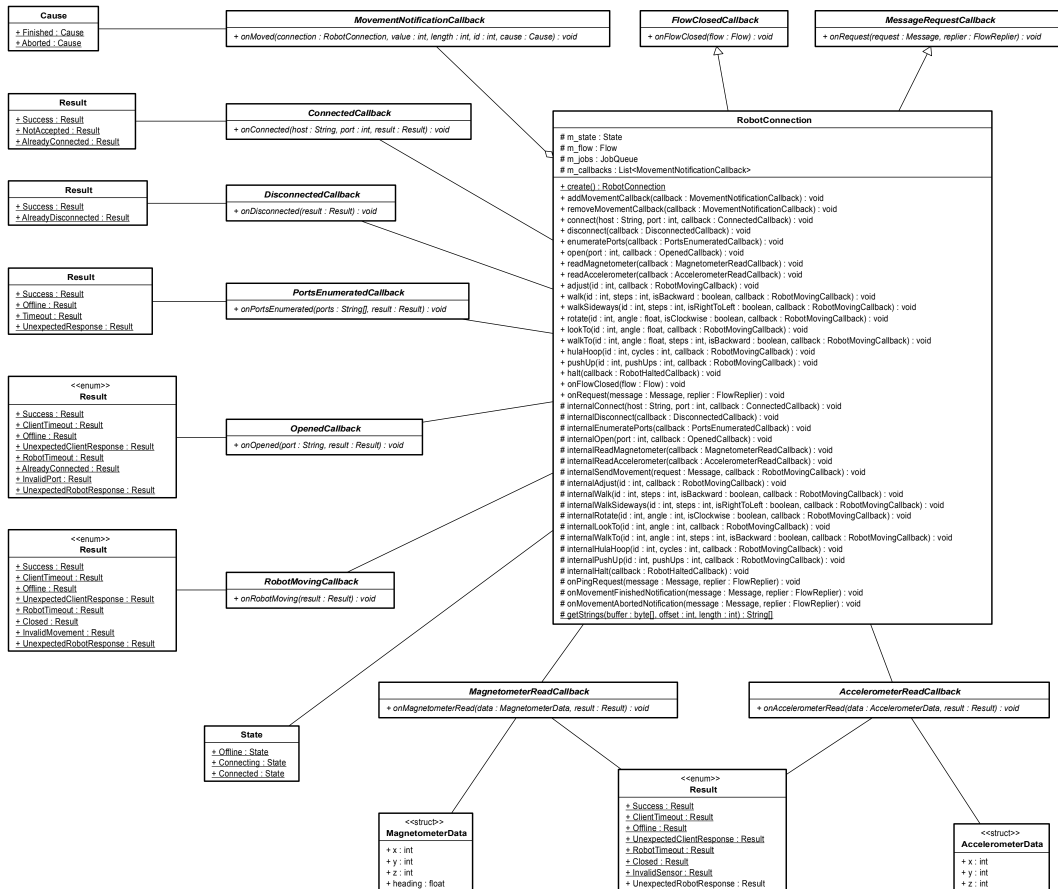


Figura 70: Diagrama de classes da biblioteca de cliente.

Fonte: Autoria própria.

O cliente é operacionalmente mais simples que as demais partes do software, pois não realiza operações em paralelo. Sua máquina de estados é apresentada na figura 71.

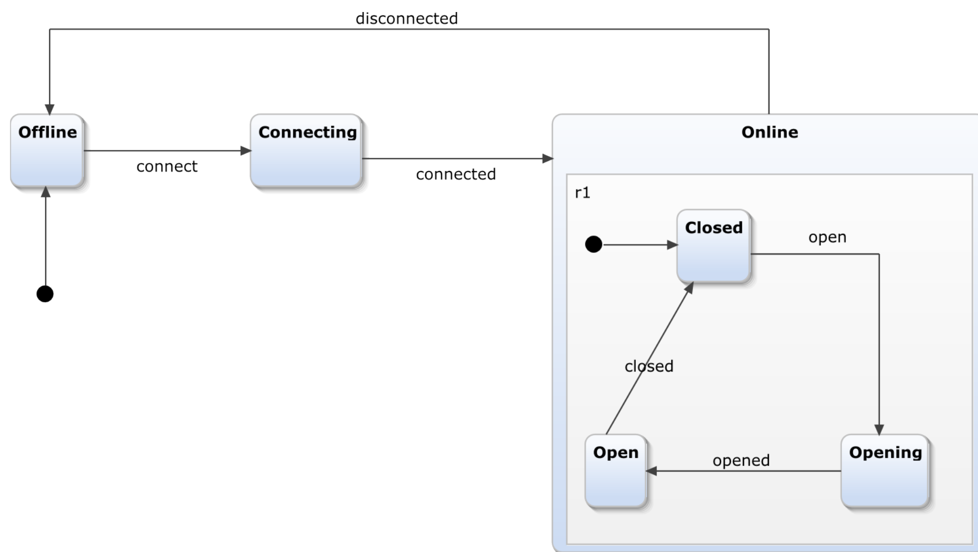


Figura 71: Statechart da biblioteca de cliente.

Fonte: Autoria própria.

Como acontece com as classes de protocolo, o acoplamento entre cada instância da classe de cliente e outros objetos é feita através de callbacks. Tanto o disparo de eventos quando as chamadas aos métodos são assíncronas, disparando callbacks em objetos específicos.

3.5.4.5 Interface gráfica

O software de interface gráfica desenvolvido foi implementado com o intuito de validar e testar o funcionamento do robô. Todo o código foi concentrado em um único arquivo-fonte, que contém o componente de controle do robô, "HexapodCommandsControl".

A interface foi desenvolvida em linguagem Java, com a biblioteca Swing. Ela foi concebida para permitir a utilização de todas as funcionalidades implementadas na biblioteca de cliente, ou quase todas. Uma captura de tela do software de interface gráfica logo após ser aberto é apresentada na figura 72.

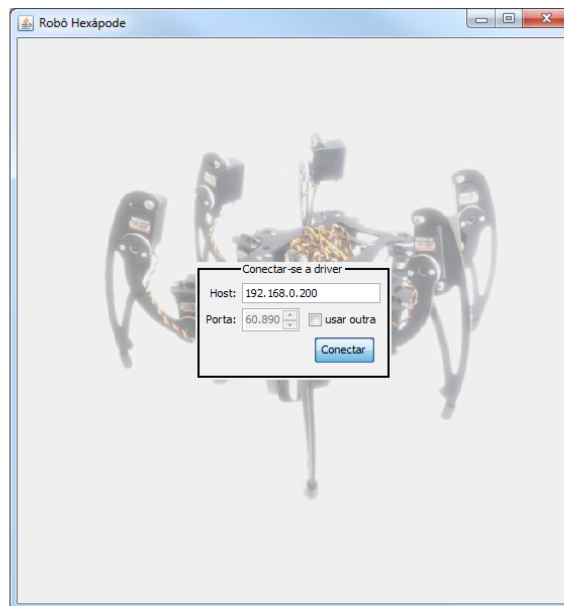


Figura 72: Tela inicial do software de interface gráfica.

Fonte: Autoria própria.

Assim que o software é aberto, ele não está conectado a driver algum. Para estabelecer a conexão com um driver é preciso indicar host do computador no qual o driver está e pressionar o botão “Conectar”. Se a conexão for estabelecida com sucesso, o usuário será direcionado para a tela apresentada na figura 73.

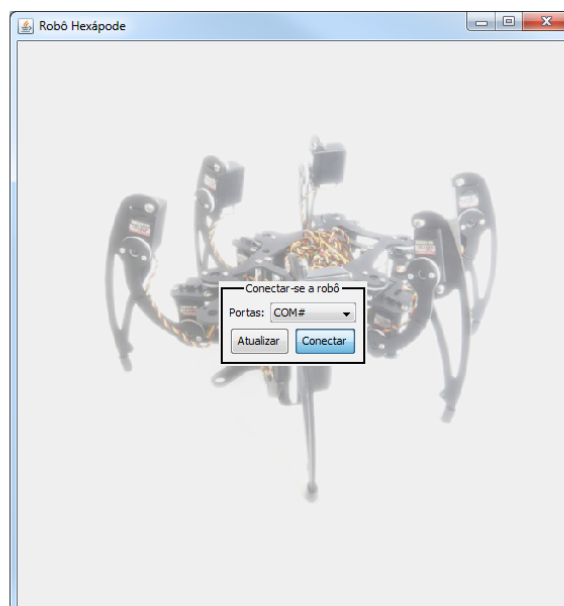


Figura 73: Tela de conexão com robô do software de interface gráfica.

Fonte: Autoria própria.

Nesta tela, o software está conectado ao driver, mas o driver ainda não está conectado ao robô. É preciso selecionar a porta na qual o robô se encontra e pressionar o botão “Conectar”. Se tudo correr bem, o usuário será direcionado para a tela apresentada na figura 74.

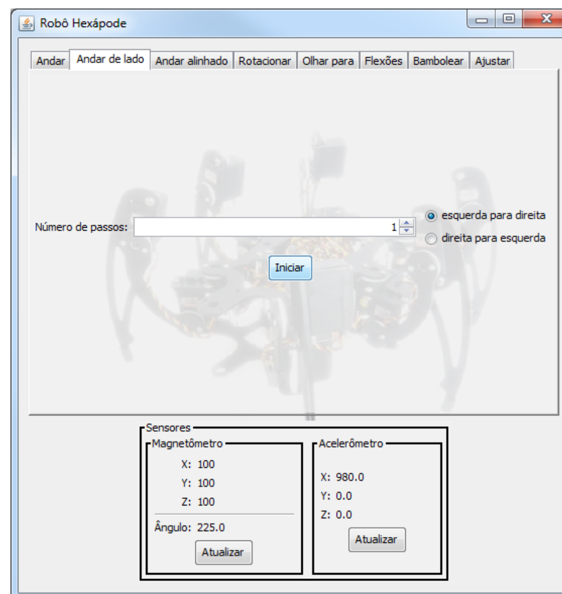


Figura 74: Tela de execução de movimentos e leitura de sensores do software de interface gráfica.

Fonte: Autoria própria.

Uma vez nesta tela, o usuário pode obter leituras de sensores (através de botões na parte inferior da tela) e iniciar a execução de movimentos (através das abas na parte superior da tela). Se os parâmetros de um movimento forem preenchidos e o botão “Iniciar” for pressionado, a tela apresentada na figura 75 será exibida. Nesta tela o usuário pode abortar o movimento que foi iniciado. Quando o movimento terminar, ou for abortado, a tela de leitura de sensores e execução de movimentos é exibida novamente.

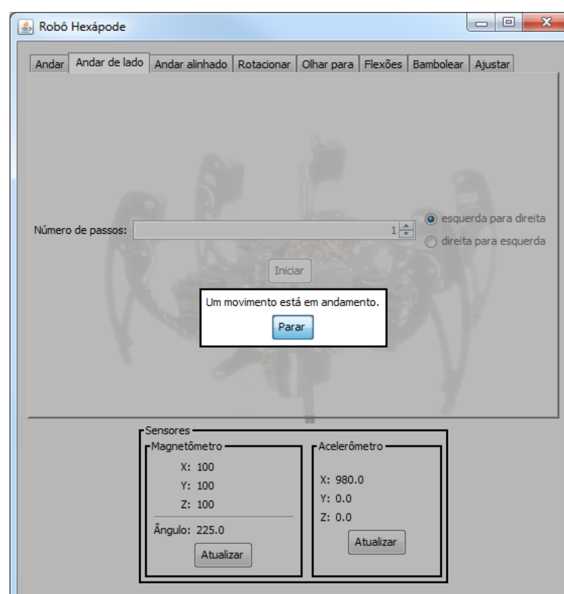


Figura 75: Tela de movimento em andamento.

Fonte: Autoria própria.

3.6 Movimentos

Durante a execução do projeto foram definidos, pela equipe, alguns movimentos para ilustrar e validar todo o sistema de controle do hexápode desenvolvido. Essa etapa, apesar de demandar um estudo prévio de como são realizados os movimentos de um hexápode, foi muito mais prática, no sentido de realizar testes e calibragens, do que as outras.

Através da cinemática inversa foi mais simples criar os movimentos pois não é preciso controlar a variação angular de cada um dos motores. Para isso basta variar a posição da ponta da pata utilizando coordenadas cartesianas que a cinemática trata de ajustar os ângulos, quando for possível, para que ocorra o deslocamento correto. Com isso foi possível criar um conjunto de posições pré definidas para realizar cada um dos movimentos executados pelo hexápode. Entretanto, para criar movimentos mais fluídos todos as posições pré definidas são interpoladas, entre 10 e 32 pontos, fazendo com que os servomotores realizem pequenos movimentos, em vez de um movimento brusco, para mover de uma posição para outra.

A Seção “Gait” apresentou algumas possíveis configurações para criar movimentos para um robo hexápode. Das opções de gait disponíveis, optou-se por utilizar o tripod gait. Além de ser a opção mais rápida é umas das que mais se assemelha ao movimento de

animais hexápodes. Tomando o movimento de andar como base de demonstração, são utilizados 4 passos para cada pata com a finalidade de completar um ciclo, desconsideradas as etapas de saída da posição de repouso e retorno a posição de repouso. Quando 3 patas estão no ar, as outras 3 estão no chão. Essa rotina se repete, alternando as patas no ar e no chão, até que o movimento seja completado.

Foram desenvolvidos 3 tipos de movimentos no projeto. Os movimentos básicos consistem em movimentos simples de locomoção do robô, como andar para frente ou andar de lado. Os movimentos com magnetômetro são aqueles responsáveis por controlar a direção apontada pelo robô através de uma malha de controle onde os movimentos são ajustados de acordo com as medidas do magnetômetro. Por último, os movimentos demonstrativos são aqueles que não deslocam o robô, eles apenas executam uma série de movimentos para ilustrar as capacidades do hexápode.

3.6.1 Movimentos básicos

Existem quatro movimentos básicos que podem ser executados pelo robo hexápode: andar para frente, andar para trás, andar de lado para esquerda e andar de lado para direita. Todos eles executam quatro etapas básicas de movimentos pré definidos para realizar um ciclo que pode ser encadeado inúmeras vezes e com isso gerar os movimentos de andar do robô. O ciclo é basicamente o mesmo para todos os movimentos básicos, porém, o que os diferencia é o sentido (no caso do andar para frente ou para trás) ou a direção (no caso de andar para frente ou para o lado) em que eles ocorrem. Outro fator importante é que antes e depois de cada movimento são realizados alguns deslocamentos das patas para sair da posição de descanso do hexápode e colocá-lo na primeira posição do ciclo de movimento (início do movimento), ou sair da última posição do ciclo e colocá-lo novamente na posição de descanso (final do movimento). As etapas pré-ciclo e pós-ciclo são importantes para evitar que haja deslocamento no robô, portanto todos os movimentos são executados com as patas no ar e sempre alternando os conjuntos de patas utilizadas no “gait triplo” para manter a estabilidade.

As quatro etapas do ciclo de um movimento básico do hexápode são: 1) deslocar as patas da frente e de trás do lado direto do robô e a pata central do lado esquerdo no chão, enquanto as demais são deslocadas no sentido oposto no ar. 2) As patas que estavam no chão se deslocam para cima (para o ar) e as patas que estavam no ar se

deslocam para baixo (para o chão). 3) deslocar as patas da frente e de trás do lado esquerdo do robô e a pata central do lado direito no chão, enquanto as outras são deslocadas no sentido oposto no ar. 4) Novamente as patas que estavam no ar se deslocam para o chão e as patas que estavam no chão se deslocam para o ar.

Ao final das quatro etapas o robô volta a posição inicial do ciclo, que pode ser repetido inúmeras vezes e com isso gerar os movimentos básico de locomoção do hexápode. As diferenças entre os ciclos de movimentos para gerar os quatro movimentos básicos de locomoção são explicadas a seguir:

- Andar para frente: consiste em realizar os deslocamentos no chão fazendo com que as patas comecem mais a frente do robô e sejam deslocadas para trás. Enquanto isso os movimentos no ar seguem o sentido oposto, ou seja, se deslocam de trás para frente. Um ponto importante nesse movimento é que para evitar que o hexápode ande em “zigue-zague” é necessário que a pata central de cada lado se desloque o dobro das patas frontais e traseiras para que a força resultante do movimento não faça com que o hexápode gire.
- Andar para trás: é basicamente o mesmo movimento do andar para frente, entretanto, as patas no chão são deslocadas de trás para frente as patas no ar são deslocadas da frente para trás.
- Andar de lado para direita: consiste em realizar os deslocamentos no chão fazendo com que as patas que estão no chão comecem mais à direita do robô e sejam deslocadas para a esquerda. Ao mesmo tempo as patas que estão no ar são deslocadas da esquerda para a direita.
- Andar de lado para esquerda: o movimentos é o mesmo do andar para a direita, mas as patas que estão no chão se deslocam da esquerda para a direta e as patas que estão no ar se deslocam da direita para a esquerda.

3.6.2 Movimentos com magnetômetro

Diferente dos movimentos básicos de locomoção, que são controlados através do número de ciclos de execução, os movimentos com o magnetômetro são responsáveis por fazer com que o hexápode gire e aponte para alguma direção (variação em ângulo com relação ao Norte) através de um controle utilizando o magnetômetro. Foram criados três movimentos básicos utilizando magnetômetro: olhar para uma direção, girar no sentido horário e girar no sentido anti-horário.

Para utilizar o magnetômetro corretamente é necessário calibrá-lo, por isso foi criado um quarto movimento de ajuste. Esse movimento faz com que o hexápode gire completamente em torno do próprio eixo e realize medições com o magnetômetro. Com isso são armazenados os valores extremos dos 2 eixos horizontais ao plano em que se encontra o hexápode e ajustados os *offsets* para que o ponto médio fique no ponto (0,0).

Apesar de existirem três movimentos para rotacionar o hexápode, todos eles são executados internamente no *firmware* como sendo o movimento de olhar para uma direção. Para simplificar a movimentação os movimentos de girar tantos graus no sentido horário ou anti-horário é realizada uma leitura antes de começar a movimentação e calculada a variação de acordo com os parâmetros repassados para o usuário e com isso encontrar o ângulo que o hexápode deve ser direcionado. Após essa etapa é iniciado efetivamente o movimento de rotação do hexápode.

O movimento de rotação é executado em três etapas, primeiramente é realiza a leitura do magnetômetro, depois é calculado o erro e por último é realizado um ciclo do giro. Essas etapas são repetidas até que o hexápode consiga ser direcionado para o ângulo de destino. Após alguns testes foi definida que uma boa margem de erro para que ele complete o movimento de rotação é de 3 graus.

Após ser realizada a leitura da posição atual do robô é calculado o erro, ou seja, quantos graus faltam para o hexápode ficar alinhado ao ângulo desejado. Através desse erro dois pontos são estabelecidos para definir como será o movimento para rotacionar o robô. O primeiro fator verifica para qual o sentido que deverá acontecer o giro, caso o erro encontrado seja maior do que zero o giro acontece no sentido anti-horário, caso contrário ele será no sentido horário. Como a margem de erro utilizada é de 3 graus, se o módulo do erro for menor do que 3 o movimento é encerrado.

O segundo ponto estabelecido durante o cálculo do erro é definir qual será o deslocamento das patas para realizar o giro. Depois de realizados alguns testes verificou-se que o hexápode precisa na média de 26 giros para deslocar 360 graus utilizando passadas de aproximadamente 50mm. Através disso é possível calcular a constante “c” que relaciona quantos graus cada milímetro de movimento rotaciona utilizando a equação 13 abaixo.

$$c=360/(26*50) \quad (13)$$

Através disso foi possível encontrar a constante proporcional para realizar o controle do giro do hexápode (c = 0,27). Para calcular qual é o deslocamento (Δy) da pata para realizar o movimento de rotacionar. A equação 14 apresenta a relação existente entre o erro atual e o deslocamento das patas para realizar o próximo movimento de rotação. Onde “Δy” representa o deslocamento da pata em milímetros e “e” representa o erro em graus.

$$\Delta y = 0{,}27 * |e| \quad (14)$$

Depois de calculado o erro, o movimento de rotacionar é efetivamente realizado pelo hexápode. Tanto a rotação no sentido horário quanto a rotação no sentido anti-horário são realizadas utilizando um ciclo com 7 pontos pré-definidos. O movimento foi inspirado no mecanismo de giro utilizado pelos tanques de guerra, ou seja, enquanto as patas de um lado do hexápode se deslocam para frente, as outras se deslocam para trás fazendo com que o robô gire. A única diferença entre o sentido horário ou anti-horário do movimento é qual o lado do hexápode que vai se deslocar para frente. Ao mover, no chão, as patas da direita para trás e as patas da esquerda para frente o robô gira no sentido anti-horário, caso ocorra o oposto ele gira no sentido horário.

Para exemplificar as sete etapas do ciclo do movimento de rotação será utilizado o movimento no sentido anti-horário: 1) A pata da frente do lado direito, a pata traseira do lado direito e a pata central do lado esquerdo são erguidas. 2) As patas do lado direito que estão suspensas se deslocam para a frente enquanto a pata do lado esquerda se desloca para trás. 3) As patas que estavam no ar são deslocadas para o chão. 4) A pata da frente do lado esquerdo, a pata traseira do lado esquerdo e a pata central do lado direito são erguidas. 5) A pata do lado direito suspensa se desloca para frente enquanto as patas do lado esquerdo se deslocam para trás. 6) As patas que estavam no ar são deslocadas para o chão. 7) As patas do lado direito que estavam mais a frente se deslocam para trás e as patas do lado esquerdo que estavam mais atrás se deslocam para frente. Com isso o hexápode realiza o giro e volta para a posição inicial.

3.6.3 Movimentos demonstrativos

A última categoria de movimentos recebeu esse nome pois não apresenta nenhuma utilidade prática de locomoção e serve apenas pra ilustrar o controle existente do hexápode. Para fins demonstrativos foram criados 2 movimentos. O primeiro, nomeado

"bambolê", faz com que o robô alterne a altura das suas seis patas em quatro diferentes níveis, sempre alternando a mais baixa e a mais alta em um ciclo de 6 posições (uma para cada pata). Fazendo com que uma das patas fique em uma posição mais elevada, a pata oposta aquela fique em uma posição mais baixa e as demais em níveis intermediários (elevados ou rebaixados dependendo da elevação da pata ao seu lado) é gerado um movimento semelhante ao de uma pessoa brincando com um bambolê.

O outro movimento, intitulado "flexão", faz com que o hexápode assuma uma posição semelhante ao movimento de uma pessoa fazendo flexão, onde as patas traseiras ficam posicionadas mais para trás e as patas dianteiras ficam ligeiramente abertas e variando entre uma posição mais elevada e uma posição no mesmo nível das patas traseiras, simulando o movimento realizado pelo exercício físico. Durante a realização do movimento as patas centrais ficam erguidas para não influenciarem na execução do mesmo.

3.7 Considerações

O hardware desenvolvido na FPGA teve de integrar diversos conceitos para concretizar os requisitos do projeto. A utilização de um processador softcore foi bastante proveitosa principalmente no que se refere a comunicação. Mesmo nos casos em que a biblioteca padrão não tinha todos os recursos necessários, o acoplamento de blocos de hardware de terceiros aconteceu sem grandes dificuldades. Alguns conceitos, como por exemplo a cinemática inversa, ofereceram maiores dificuldades por necessitarem uma maior maturidade teórica pré-implementação e planejamento para integrar ao resto do sistema. Deve-se mencionar também a mudança de paradigma que se deve ao fato da FPGA tratar-se de um dispositivo que opera de forma concorrente, em oposição aos microcontroladores e microprocessadores normalmente utilizados. O resultado final é um hardware que realiza o que foi proposto e uma solução bastante robusta e compacta.

Para distribuir a alimentação e os dados entre a FPGA e os outros periféricos (motores, sensores e XBee) foram desenvolvidas 2 placas de circuito impresso. Através disso foi possível separar a alimentação em dois canais, um de alta potência utilizado nos servomotores e outro de baixa potência utilizado para alimentar o restante do projeto. Apesar das dificuldades enfrentadas durante esta parte do projeto, foi possível construir um circuito modular, que além de possuir um rendimento satisfatório, apresentou uma manutenção fácil por permitir que seus componentes sejam desacoplados rapidamente.

A alimentação do hexápode foi dividida em 2 partes, uma mais simples (baixa potência) utilizada na FPGA, XBee e sensores era fornecida por uma fonte comum de 9V e 1.5A. A outra parte utilizava uma Fonte ATX de computador, utilizada no circuito de alta potência para alimentar os servomotores, que não apresentou nenhuma grande complicação e foi suficiente para fornecer os aproximados 14A que os motores consomem durante o funcionamento do hexápode.

A elaboração do protocolo em dois níveis (baixo e alto nível) tornou mais simples o desenvolvimento das classes de comunicação. Enquanto o protocolo de alto nível é diferente para cada par de entes comunicantes, e ainda é tratado diferentemente em cada lado da comunicação, o protocolo de baixo nível é consistente em todos os sistemas. Assim, o projeto das classes de comunicação também pôde ser desenvolvido em duas partes: uma comum a todos os artefatos de software (classes de protocolo) e outra específica para cada (classes de alto nível específicas). Outra vantagem da implementação elaborada é o fato de as classes de comunicação serem elaboradas sobre fluxos genéricos de dados, de maneira que podem ser utilizadas sobre outros meios em versões futuras do projeto.

As classes de gerenciamento automático de memória e *pools* de objetos elaboradas em C++ são totalmente independentes do código especializado do *firmware*, então podem ser utilizadas em outros projetos de software. As classes de baixo nível e parte das classes de alto nível (especialmente as referentes ao protocolo de baixo nível) também têm esta propriedade. Assim, um subproduto do robô hexápode são bibliotecas de código aberto em linguagens C++ e Java, que incluem classes de baixo nível e comunicação por troca de mensagens.

Adotou-se o procedimento de preservar grandemente a estrutura do código os diversos artefatos de software, mesmo entre linguagens distintas. Este procedimento tem um lado negativo, por não levar em conta características específicas de cada linguagem plataforma, o que às vezes pode levar à escrita de código desnecessário. Em contrapartida, o procedimento de desenvolvimento foi bastante simplificado, pois, se os códigos seguem a mesma sequência de operações, todas as alterações feitas em um (como por exemplo, correções de erros) são replicadas para os demais de forma trivial.

Dos três artefatos principais de software, o mais complexo é provavelmente o firmware, pois é responsável por gerenciar diversos dispositivos em interação: as seis patas, os sensores e a comunicação com o *driver*. O *driver* pode ser colocado em segundo lugar em ordem de complexidade, pois, além de se comunicar com o robô e com

o cliente, tem de coordenar, em termos de controle de concorrência, a comunicação entre ambos.A biblioteca de comunicação é a seguinte em ordem decrescente de complexidade: apesar de ter um controle rudimentar de concorrência, sua função acaba resumindo-se apenas a enviar comandos ao *driver*, e então esperar e tratar suas respostas. O software de interface gráfica também é simples em termos de complexidade de projeto, pois sua função é apenas enviar comandos utilizando a biblioteca de comunicação, recebendo e tratando suas respostas. Ele, entretanto, apresenta outra dimensão de complexidade, que é a elaboração da interface em si, em termos de usabilidade.

4 Plano de testes

Para verificar o funcionamento do sistema integrado, foi elaborado um plano de testes. O plano contempla os requisitos funcionais sistêmicos e os requisitos funcionais do software de interface gráfica. Cada caso de teste é composto de um roteiro que deve ser seguido. Se uma etapa do teste levar a um resultado diferente do esperado, considera-se que o caso de teste como um todo falhou. Assim que o projeto foi terminado, no período de integração, todos os casos de teste foram executados com sucesso.

4.1 Caso de teste I: Conexão com o driver

Este caso de testes é referente ao requisito RF9. Em resumo, consiste em verificar se o software de interface gráfica consegue conectar-se ao driver através da biblioteca de comunicação.

Pré-condições: o software de interface gráfica deve estar instalado em um computador A, e o *driver* deve estar em execução em um computador B.		
Etapa	**Instrução**	**Resultado esperado**
1	Abra o software de interface no computador A.	A tela inicial do software é exibida.
2	Digite um endereço de IP no qual o *driver* não esteja sendo executado. Pressione o botão de conectar.	Após algum tempo, é exibida a mensagem de que não foi possível conectar ao *driver*.
3	Digite o endereço de IP do computador B. Pressione o botão de conectar.	A tela de selecionar porta é exibida.

4.2 .Caso de teste II: Conexão com o robô

Este caso de testes é referente aos requisitos RF3, RF7 e SWRF1. Em resumo, consiste em verificar se o software de interface gráfica consegue conectar-se ao robô através da biblioteca de comunicação e do *driver*.

Pré-condições: o software de interface gráfica está em execução em um computador A, e o *driver* está em execução em um computador B. O software de interface gráfica do computador A está conectado ao *driver* do computador B. O robô está totalmente ligado.		
Etapa	**Instrução**	**Resultado esperado**
1	Selecione uma porta, no computador A, na qual o robô não esteja conectado. Se não houver tal porta, pule para o passo 2. Pressione o botão de conectar.	Após algum tempo, é exibida a mensagem de que não foi possível conectar ao robô.
2	Selecione uma porta, no computador A, na qual o robô esteja conectado. Pressione o botão de conectar.	A tela de enviar comandos e ler sensores é exibida.

4.3 Caso de teste III: Posição de repouso do robô

Este caso de testes é referente aos requisitos RF1 e RF4. Em resumo, consiste em verificar se o robô vai para a posição de repouso e permanece nela.

Pré-condições: a eletrônica de potência do robô está desligada, e as patas estão em posição diferente da de repouso.		
Etapa	**Instrução**	**Resultado esperado**
1	Ligue a eletrônica de potência do robô.	O robô fica na posição de repouso.
2	Aguarde alguns instantes.	O robô não sai na posição de repouso.

4.4 .Caso de teste IV: Movimentação do robô

Este caso de teste é referente aos requisitos RF2, RF4, RF5, RF8 e SWRF2. Em resumo, consiste em verificar se o robô locomove-se quando recebe um comando.

Pré-condições: o software de interface gráfica está em execução em um computador A, e o *driver* está em execução em um computador B. O software de interface gráfica do computador A está conectado ao *driver* do computador B. O *driver* do computador B está conectado a um robô. O robô está totalmente ligado.		
Etapa	**Instrução**	**Resultado esperado**
1	Selecione a aba de “andar” e preencha os parâmetros para andar um passo para a frente. Pressione o botão de iniciar.	O robô dá um único passo para a frente e para.
2	Selecione a aba de “andar” e preencha os parâmetros para andar dez passos para a frente. Pressione o botão de iniciar. Quando o movimento iniciar, pressione o botão de abortar.	O robô começa a andar para a frente, mas para quando o botão é pressionado.

4.5 Caso de teste V: Leitura de sensores do robô.

Este caso de teste é referente aos requisitos RF6, RF8 e SWRF3. Em resumo, consiste em verificar se é possível obter leituras dos sensores do robô.

Pré-condições: o software de interface gráfica está em execução em um computador A, e o driver está em execução em um computador B. O software de interface gráfica do computador A está conectado ao driver do computador B. O *driver* do computador B está conectado a um robô. O robô está totalmente ligado.		
Etapa	**Instrução**	**Resultado esperado**
1	Com o robô parado, pressione o botão de atualizar leitura do magnetômetro.	Após alguns instantes, a leitura do magnetômetro na tela é atualizada.
2	Mude o ângulo do robô em relação ao norte, e pressione o botão de atualizar leitura do magnetômetro.	Após alguns instantes, a leitura do magnetômetro na tela é atualizada para um valor distinto do apresentado anteriormente.
3	Pressione o botão de atualizar a leitura do acelerômetro.	Após alguns instantes, a leitura do acelerômetro na tela é atualizada.

4.6 Considerações

Os casos de testes foram elaborados como uma maneira formal de validar o funcionamento do robô. O plano de testes como um todo é bastante curto, mas cada um deles (à exceção do caso I) verifica mais de um requisito. É comum que cada teste englobe um único requisito, mas, dado à sofisticação do sistema integrado, não faria sentido avaliar cada requisito independentemente.

5 Gestão

Para desenvolver o projeto foi utilizado o processo em espiral. Esta abordagem foi escolhida por conta do grande número de diferentes tecnologias e conceitos envolvidos.

O projeto seguiu três ciclos principais de desenvolvimento: “estudos e validações”, “desenvolvimento da arquitetura” e “montagem e integração". O primeiro ciclo consistiu em estudar as tecnologias utilizadas e validar algumas ideias, como cinemática inversa, desenvolvimento dos servos próprios e blocos de controle do PWM, para definir a arquitetura e verificar a viabilidade de certos aspectos do projeto. Nessa etapa foi construída boa parte da fundamentação teórica e criado o “alicerce” do projeto.

No segundo ciclo, foi estabelecida e desenvolvida a arquitetura do sistema. Nele, foram desenvolvidas as placas de circuito impresso, criada boa parte da lógica de controle, aprimorada a cinemática inversa, desenvolvido o driver da estação base e criado o protocolo de comunicação. Nessa etapa adquiriram-se boa parte dos componentes utilizados no projeto (estrutura mecânica, servomotores e sensores) e foi criada toda a estrutura base do projeto para permitir integrar o hardware das PCIs, o hardware embarcado, o firmware embarcado e o software da estação base.

No último ciclo, aconteceu a integração de toda a estrutura desenvolvida anteriormente. Nele a estrutura mecânica foi montada junto com os servomotores e as placas de circuito impresso de alta e baixa potência. Além disso, ela foi integrada com a FPGA, o XBee e os sensores. Nessa etapa também foram desenvolvidos os movimentos, boa parte do software embarcado e o software de interface. Por fim, neste ciclo ocorreu a grande maioria dos testes de integração, ajustes e calibragens do hexápode.

5.1 Escopo

O principal objetivo do projeto era desenvolver um robô hexápode controlado por lógica reconfigurável, que fosse extensível de alguma forma e pudesse ser controlado através de algum dispositivo externo. Também foram levantados alguns outros objetivos específicos: informar a direção apontada pelo robô, comunicar-se com um dispositivo externo através de uma conexão sem fio e possibilitar o envio de comandos básicos de movimentação.

Todos os objetivos apontados no início do projeto foram contemplados de alguma maneira. Foi desenvolvido um robô hexápode utilizando uma FPGA para prover a lógica

reconfigurável que pode ser controlado a partir de uma estação base (computador de mesa comum) utilizando um módulo de comunicação serial sem fio XBee. É possível enviar uma série de comandos básicos ao hexápode e ainda realizar a leitura de alguns sensores (magnetômetro e acelerômetro).

Entretanto, alguns objetivos surgiram de acordo com o desenrolar do projeto, mas nem todos foram totalmente contemplados. Grande parte deles surgiram ainda no primeiro ciclo do projeto e tinham um impacto muito grande no resultado final dele, como, por exemplo, o desenvolvimento dos servos próprios e o desenvolvimento da cinemática inversa. Nessa etapa surgiram duas possíveis vertentes para atingir o objetivo principal, a primeira consistia em desenvolver a cinemática inversa do hexápode, realizar movimentos mais elaborados e utilizar servomotores já existentes no mercado. A outra tinha como principal foco desenvolver os servomotores através do controle na própria FPGA, desenvolver uma arquitetura de processador específica para a robótica e realizar movimentos mais simples utilizando somente a variação de ângulos pré estabelecidos, sem utilizar uma cinemática inversa.

Durante o primeiro ciclo a equipe trabalhou em paralelo com essas duas vertentes e acabou escolhendo implementar a da cinemática inversa por diversos motivos. Um dos pontos mais interessantes em tentar desenvolver os próprios servos era o fato de poder comprar servos mais baratos e melhorá-los, reduzindo o custo efetivo do projeto, entretanto, para fazer o controle desses servos seria necessário utilizar um CI de ponte H (L298) para inverter o sinal e um LM555 para realizar a leitura de posição dos motores. Com isto o custo final se tornaria semelhante a um servo importado e de boa qualidade para o uso estabelecido no projeto. Outro fator importante foi que ao desenvolver o servo a equipe percebeu que um dos grandes limitadores dos servos mais baratos é a baixa qualidade das engrenagens da caixa de redução (feitas geralmente de plástico em servos de baixo custo) e do potenciômetro, resultando em uma melhoria de desempenho abaixo do esperado. Por fim, a PCI ficaria consideravelmente maior, mais cara e mais complexa caso fossem utilizados os servos desenvolvidos pela equipe.

Foram analisados outros dois pontos durante o projeto que acabaram sendo substituídos por outras alternativas. Uma das possibilidades consistia em utilizar uma bateria para alimentar o hexápode, entretanto, essa alternativa foi descartada pois além de aumentar o custo, já que seriam necessárias pelo menos duas baterias durante a fase de desenvolvimento, ele precisaria de um controle da carga para regular os ajustes do projeto para suportarem a queda de tensão e sua autonomia seria muito baixa devida a

grande quantidade de corrente consumida pelos servomotores utilizados. Outro ponto que foi descartado foi a possibilidade de criar um projeto mecânico, ou utilizar um projeto já existente, e confeccionar as peças em algum local especializado em cortes de metal. Entretanto, para agilizar o processo de aquisição da estrutura mecânica e evitar problemas com fornecedores e possíveis problemas com peças fora de medida, optou-se por importar uma estrutura mecânica (MSR-H01) com qualidade já reconhecida por outros desenvolvedores de hexápodes.

Apesar de algumas ideias terem sido descartadas, foram acrescentadas alguns pontos positivos ao projeto que não tinham sido contemplados na ideia original. O projeto da PCI foi dividido em duas placas separadas, isolando a parte de alta potência que alimenta os motores da de baixa potência que envia o sinal de PWM para os servos e alimenta a FPGA e os sensores. Outro ponto positivo foi a utilização de um processador embarcado na FPGA (NIOS II) com o RTOS MicroC para facilitar na comunicação e gerenciar melhor a estrutura do hardware embarcado. Por último, a mudança de escopo que mais beneficiou o projeto foi uma alteração na arquitetura do software da estação base. Para melhorar consideravelmente a extensibilidade do projeto decidiu-se por separar o *driver* que comunica diretamente com o Hexápode, e precisa estar ao alcance do XBee, do software de interface criando uma nova camada de comunicação através de *sockets*. Com isso é possível controlar o hexápode em qualquer computador, ou dispositivo, que consiga acessar, através da rede ou da internet, a máquina que está executando o software com o *driver* do robô, desde que ele conheça a interface de comunicação implementada no *driver*.

5.2 Custos e Cronograma

O custo do projeto pode ser avaliado através do gasto, em dinheiro, com equipamentos e materiais e do total de horas trabalhadas pela equipe para desenvolvê-lo. Em termos monetários o projeto custou R$2.291,10, entretanto, esse valor reflete o gasto real da equipe e não leva em consideração algumas taxas de importação e equipamentos que não precisaram ser comprados. Em relação ao tempo gasto foram necessárias aproximadamente 1600 horas de trabalho, que foram divididas entre os 3 membros da equipe durante um período pouco maior do que 1 ano (05/03/2012 – 30/04/2013). Informações mais detalhadas a respeito das atividades realizadas e dos gastos do projeto são apresentadas no decorrer desta seção.

A primeira estimativa de gastos foi apresentada no início deste documento durante a seção de viabilidade financeira e pode ser vista na tabela 1. Entretanto, por conta de algumas mudanças no escopo detalhadas na seção anterior, vários itens utilizados não foram estimados e outros nem chegaram a fazer parte do projeto. Apesar disso o custo total estimado, que variava de R$2.000,00 a 2.400,00, foi mantido dentro do possível. A tabela 10 apresenta com detalhes todos os gastos do projeto. Eles estão separados em 9 grupos para facilitar a visualização dos gastos. Dentro dos grupos podem ser vistas informações referentes a gastos unitários e totais de um determinado material, gastos com transporte e taxas de importação. Para todos os produtos importados são apresentados os valores originais referentes a moeda corrente daquele país e a cotação em real utilizada no dia da compra. A coluna Tipo do custo é utilizada para diferenciar o custo total do projeto, ela indica se o produto é um custo exclusivo de produção (P), exclusivo de desenvolvimento (D) ou um custo efetivo (E) tanto para produção quanto para desenvolvimento. Gastos com componentes eletrônicos de uso geral (ex: resistores, capacitores, fios e material para solda) não foram discretizados para não poluir a tabela, entretanto, o valor gasto com eles está agrupado nos itens de componentes diversos.

Tabela 10: Tabela de gastos do projeto

Fonte: Autoria própria.

Tabela de Gastos do TCC						
Grupo	**Quantidade**	**Material/Taxas/Transporte**	**Custo unitário**	**Custo Total**	**Tipo do custo**	**Valor Original da Importação**
FPGA	1	FPGA DE0-Nano + Frete + Taxas	R$ 280,00	R$ 280,00	E	US$175,00 (US$1,00 = R$1,60)
Servos Motores	12	BMS-620MG High Torque Metal Gear Servo	R$ 28,59	R$ 343,08	E	US$157,55 (US$1,00 = R$2,18)
	2	BMS-620MG High Torque Metal Gear Servo (RESERVA)	R$ 28,59	R$ 57,18	D	US$26,26 (US$1,00 = R$2,18)
	6	Corona DS329MG Digital Metal Gear Servo	R$ 18,12	R$ 108,72	E	US$49,82 (US$1,00 = R$2,18)
	1	Corona DS329MG Digital Metal Gear Servo (RESERVA)	R$ 18,12	R$ 18,12	D	US$8,32 (US$1,00 = R$2,18)
	1	Frete dos servos motores importados	R$ 64,88	R$ 64,88	E	US$29,79 (US$1,00 = R$2,18)
	1	Taxa de importação dos servos motores	R$ 271,08	R$ 271,08	P	US$124,35 (US$1,00 = R$2,18)
Estrutura Mecânica	1	MSR-H01 Hexapod Robot Kit (Estrutura Mecânica)	R$ 403,46	R$ 403,46	E	£$116,66 (£1,00 = R$3.46)
	1	Frete da estrutura mecânica importada	R$ 69,58	R$ 69,58	E	£$20,12 (£1,00 = R$3.46)
	1	Taxa de importação da estrutura mecânica	R$ 235,86	R$ 235,86	E	£$68,17 (£1,00 = R$3.46)
Xbee	2	Módulo XBee® 802.15.4 S1	R$ 38,10	R$ 76,20	E	€29,88 (€1,00 = R$2,55)
	1	XBee® USB Adapter Board	R$ 44,11	R$ 44,11	E	€17,30 (€1,00 = R$2,55)
	1	XBee® Board Adapter (Shield)	R$ 6,40	R$ 6,40	E	€2,51 (€1,00 = R$2,55)
Sensores	1	GY-80 BMP085 Magnetic Acceleration Gyroscope Module (IMU)	R$ 79,00	R$ 79,00	E	-
	1	Frete da IMU	R$ 12,00	R$ 12,00	E	-
PCI	1	Produção das PCIs (Servos e FPGA)	R$ 230,00	R$ 230,00	E	-
	1	Componentes diversos da placa final	R$ 55,00	R$ 55,00	E	-
	18	CI FOD817B - Optoacopladores	R$ 1,54	R$ 27,72	E	-
	9	CI FOD817B - Optoacopladores (RESERVA)	R$ 1,54	R$ 13,86	D	-
Alimentação	1	Fonte ATX para alimentação dos servos	R$ 75,00	R$ 75,00	P	-
	1	Fonte para alimentação da FPGA/Sensores	R$ 15,00	R$ 15,00	P	-
Testes	3	Servo motor Motortech	R$ 19,46	R$ 58,38	D	-
	1	Frete dos servos Motortech	R$ 13,38	R$ 13,38	D	-
	1	Componentes diversos da placa de servo caseiro	R$ 44,55	R$ 44,55	D	-
Outros	1	Outros gastos diversos com materias	R$ 49,62	R$ 49,62	D	-
Totais	**Custo Real de Desenvolvimento do Projeto:**		**R$**	**2.291,10**	**TOTAL (E + D)**	
	Custo de Produção (com taxas de importação):		**R$**	**2.397,09**	**TOTAL (E + P)**	
	Custo Total de Desenvolvimento:		**R$**	**2.652,18**	**TOTAL (E + D + P)**	

Durante o processo de construção do plano de projeto, durante o primeiro semestre de 2011, foi criado um planejamento das atividades necessárias para realizar o desenvolvimento do projeto e com isso foi elaborado um diagrama de Gantt com o cronograma preliminar que pode ser visto na figura 76. Apesar do diagrama contemplar boa parte das tarefas desenvolvidas durante o projeto, ele foi criado em uma etapa onde a equipe não tinha uma visão totalmente clara do projeto e nem um escopo completamente definido. Por isso existe uma grande divergência entre o planejamento inicial e o realizado efetivamente. O período de execução do projeto também foi diferente do planejado por conta de dois eventos distintos, o primeiro envolve o intercâmbio planejado de seis meses de um dos membros da equipe e que durou um ano, entretanto, apesar de afetar o planejamento a equipe chegou a desenvolver uma parte do projeto em paralelo mesmo com um dos integrantes na Alemanha. O evento que efetivamente alterou todo o cronograma do projeto foi a greve de 4 meses da UTFPR que ocorreu no ano de 2012. Com isso o calendário acadêmico foi deslocado e o ano letivo que deveria acabar em Dezembro de 2012 deve acabar somente em Maio de 2013. Por conta da greve a equipe pode estender o cronograma e realizar a maior parte do trabalho após o retorno do integrante que estava realizando o itercâmbio.

Através da tabela 10 é possível extrair 3 tipos de custos finais do projeto. O primeiro representa o custo real de desenvolvimento do projeto, ou seja, tudo aquilo que foi gasto de fato pela equipe. Nesse gasto são consideradas todos os materiais reservas utilizados no projeto e todos os materiais utilizados em testes e que não foram agregados ao produto final, entretanto, não são contemplados gastos com materiais que não precisaram ser comprados, como por exemplo a fonte ATX, e nem a taxa alfandegária dos servomotores, pois estes não foram tributados. O custo de produção, por sua vez, descarta todos os materiais utilizados somente no desenvolvimento e as peças reservas, porém, ele contempla o gasto com equipamentos que não precisaram ser comprados e as taxas de importação de todos os produtos. Ele representa o “custo base” para que o projeto seja replicado caso necessite comprar todos os materiais e sejam tarifados todos os produtos no processo de importação. Por último, o custo total de desenvolvimento reflete a soma de todos os componentes listados na tabela, ou seja, ele representa o custo de desenvolvimento incluindo todas as taxas de importação e compra de todos os componentes utilizados.

	Nome	Início	Término
1	Intercambio Leandro	01/08/11 08:00	24/02/12 17:00
2	Estudos Diversos	01/08/11 08:00	24/02/12 17:00
3	Compra de Componentes	01/08/11 08:00	24/02/12 17:00
4	Desenvolvimento Protocolo de Comunicação	24/02/12 12:00	26/04/12 13:00
5	Desenvolvimento da Comunicação Da Estação Base	26/04/12 13:00	27/06/12 13:00
6	Desenvolvimento da Comunicação do Embarcado	26/04/12 13:00	27/06/12 13:00
7	Projeto da Lógica do Hardware Reconfiguravel	24/02/12 13:00	27/03/12 13:00
8	Desenvolvimento da Lógica Reconfigurável	27/03/12 13:00	23/10/12 17:00
9	Montagem da Estrutura Mecânica	26/03/12 08:00	23/04/12 17:00
10	Projeto da PCI do Circuito Eletrônico	21/02/12 09:00	22/03/12 09:00
11	Testes Com Protótipos	22/03/12 09:00	23/05/12 09:00
12	Execução da PCI definitiva	23/05/12 09:00	22/06/12 09:00
13	Projeto de Software	24/02/12 12:00	27/03/12 13:00
14	Desenvolvimento Software Estação Base	27/03/12 13:00	28/08/12 13:00
15	Testes de Integração Parcial	24/03/12 08:00	24/10/12 09:00
16	Ajustes Finais	25/10/12 14:00	23/11/12 17:00
17	Escrever Monografia	01/08/11 08:00	08/11/12 13:00
18	Fazer Apresentação	22/10/12 08:00	23/11/12 17:00

Figura 76: Diagrama de Gantt do cronograma preliminar.

Fonte: Autoria própria.

Além dos eventos que alteraram as datas do cronograma preliminar do projeto a equipe percebeu que o modelo de trabalho utilizado nele, apresentando datas de inicio e de fim completamente estabelecidas para alguns grupos de tarefas, não se mostrou muito eficaz pois ele não apresentava as horas estimadas e realizadas de trabalho. Com o passar do tempo o projeto adquiriu mais consistência e o escopo foi melhor definido, porém até chegar em uma versão mais sólida do que faltava ser feito várias tarefas sofreram alterações e muitas outras foram acrescentadas. Por esses fatores a equipe resolveu adotar uma nova estratégia para gerenciar as horas gastas no projeto e tentar simplificar o tempo gasto com planejamento de tarefas futuras e frequentes alterações que ocorreram no cronograma. A solução encontrada foi utilizar uma tabela criando inicialmente grupos de tarefas e estimando algumas horas para eles. Durante a execução do projeto foram criadas tarefas para aqueles grupos e alocadas horas estimadas para elas. Cada tarefa também apresenta uma coluna de status para acompanhar sua evolução e uma data de inicio e termino dela. Por fim uma coluna apresenta as horas trabalhadas em cada tarefa. A versão completa da tabela com todas as atividades listadas pode ser vista no Apêndice E. A versão simplificada da tabela 11, apresenta somente o agrupamento das tarefas.

A tabela de atividades representa quando foram realizadas as tarefas do TCC e quantas horas foram necessárias para executá-las. O período apresentado nela vai de Março de 2012 a Abril de 2013 pois foram esses os meses de trabalho efetivo do projeto. Algumas tarefas, como a compra de alguns componentes, ocorreram antes desse período, entretanto, todas foram repassadas para a nova data de início do projeto por não haver um registro exato de quando elas ocorreram. A maioria das tarefas de estudo e análise se concentraram durante os 7 primeiros meses do projeto, ciclo 1 comentado no inicio da seção sobre gestão, após esta etapa o escopo do projeto foi definido integralmente e começou o ciclo 2, entre o inicio do mês de Novembro de 2012 e a metade de Fevereiro de 2013, onde foi construída grande parte da arquitetura do sistema e realizada a compra de boa parte dos componentes finais do projeto como a estrutura mecânica, os servomotores e as placas de circuito integrado. O terceiro e último ciclo foi responsável por integrar toda a arquitetura do sistema, implementar tarefas de mais alto nível de abstração (software de interface e movimentos) e realizar testes, ajustes e calibragens.

Tabela 11: Tabela resumida de atividades do projeto

Fonte: Autoria própria.

Grupo de Tarefas	Data de Início	Data de Término	Horas Estimadas	Horas Trabalhadas
Realizar estudos diversos	05/03/2012	20/02/2013	45,0	45,0
Escrever o relatório	05/03/2012	30/04/2013	100,0	144,0
Gerenciar o projeto	05/03/2012	30/04/2013	80,0	80,0
Comprar componentes	05/03/2012	21/02/2013	28,0	40,0
Elaborar e testar os "servos caseiros"	13/08/2012	02/11/2012	80,0	80,0
Desenvolver a arquitetura de controle da FPGA	30/07/2012	20/01/2013	70,0	84,0
Implementar as funções básicas do NIOS	17/09/2012	23/12/2012	36,0	29,0
Elaborar o mecanismo da cinemática inversa	23/10/2012	06/12/2012	88,0	88,0
Elaborar os diagramas de Hardware	08/10/2012	18/04/2013	15,0	13,0
Desenvolver a comunicação da estação base com a FPGA	21/10/2012	10/03/2013	142,0	164,0
Projetar o protocolo de alto nível	20/10/2012	05/02/2013	16,0	35,0
Desenvolver a comunicação entre driver e software de interface	15/12/2012	18/01/2013	30,0	12,0
Projetar a PCI de distribuição, alimentação e sensores	15/12/2012	04/02/2013	80,0	108,0
Definir e testar uma fonte de alimentação	02/01/2013	23/02/2013	36,0	44,0
Montar o hexápode	01/02/2013	30/2/2013	45,0	54,0
Desevolver a interface com os sensores	15/01/2013	28/02/2013	24,0	38,0
Estudar e elaborar os movimentos	18/02/2013	13/04/2013	88,0	78,0
Desenvolver o Software da EB [driver]	01/03/2013	13/04/2013	32,0	20,0
Desenvolver o Firmware	18/02/2013	13/04/2013	32,0	32,0
Desenvolver a biblioteca do cliente	24/03/2013	20/04/2013	20,0	22,0
Elaborar os diagramas de Software	02/01/2013	20/04/2013	52,0	16,0
Documentar o Software	07/04/2013	20/04/2013	40,0	57,0
Desenvolver o software da interface	15/03/2013	15/04/2013	36,0	28,0
Realizar testes de integração final	08/03/2013	30/04/2013	157,0	266,0
Total	05/03/2012	30/04/2013	1372,0	1577,0

O tempo total gasto no projeto registrado na tabela de atividades foi de 1577 horas, mesmo esse tempo não sendo absoluto, pois muitas vezes é difícil mensurar o tempo real

trabalhado em uma determinada tarefa, ele é importante para manter um registro de horas e ajudar em estimativas de trabalhos futuros. Como foram estimadas 1372 horas para realizar o projeto, o calculo do erro entre o tempo estimado e o realizado foi de 13%, cerca de 200 horas. Entretanto dois fatores contribuíram para este valor não ser mais elevado, primeiramente como o cronograma antigo não contemplava o número de horas trabalhadas em cada tarefa o valor estimado foi igualado ao realizado em algumas tarefas da parte inicial do projeto. Em segundo lugar, como o processo foi dividido em alguns grupos de tarefas, grande parte das estimativas sugiram durante o próprio processo de desenvolvimento, fazendo com que fosse possível realizar algumas estimativas baseadas no tempo gasto em tarefas anteriores.

5.3 Análise de Riscos

Durante o planejamento do projeto foram levantados seis principais riscos que poderiam afetar o projeto de diferentes maneiras, podendo impactar no escopo, cronograma ou orçamento do projeto de alguma maneira. Esta seção tem como principal objetivo apresentar como os riscos se desenvolveram durante o projeto e quais ações foram tomadas para contorná-los. No Apêndice F são apresentados os riscos identificados de maneira mais detalhada com impacto, probabilidade e estratégias para tentar reduzir o seu efeito negativo.

De todos os riscos levantados o “Erro na Estimativa de Tempo, Custo ou Complexidade”, risco 1 do Apêndice F, foi o mais evidente e frequente durante a execução do projeto. Na seção anterior, Custos e Cronograma, foi apresentado o fato do projeto ter ultrapassado o número de horas estimadas, entretanto, como foi realizado um esforço adicional da equipe isso acabou por não impactar na data final de entrega do mesmo e não foi necessário realizar uma redução de escopo para entregá-lo a tempo. Apesar de grande parte das estimativas não coincidirem com o planejado, em termos de horas, foi possível realizar as tarefas dentro do intervalo de tempo estimado. Algumas tarefas como o projeto das PCIs e a implementação da cinemática inversa passaram por constante revisão e precisaram ser refeitos algumas vezes, fazendo com que as horas realizadas de trabalho fossem bem maiores do que as estimadas. Outra etapa do projeto que acabou demandando mais tempo do que o planejado foram as etapas de integração do sistema, pois além de muitas vezes dependerem da presença de todos os membros da equipe

para a sua execução, foram necessários vários testes e ajustes para garantir que tudo estava funcionando conforme planejado.

Apesar do risco citado anteriormente ter sido o mais frequente durante a execução do projeto, a “Demora ou Atraso na Aquisição de Componentes”, risco 2 do Apêndice F, foi provavelmente o risco que fez a equipe ficar mais receosa. Como muitos componentes precisaram ser importados, para reduzir efetivamente os gastos ou por não haver disponibilidade dos mesmos no Brasil, o projeto ficou muito suscetível a dependências externas em uma fase crítica que poderia ter afetado diretamente o prazo de entrega. O tempo de espera na alfândega, que pode variar de dias a meses, era o fator mais significativo nesse atraso e, por se tratar de um órgão de fiscalização federal, não havia como a equipe, ou o vendedor do produto, fazerem nada para agilizar o processo. Entretanto, nenhum dos componentes adquiridos demoraram tempo suficiente para impactar o cronograma geral do projeto. A demora na chegada dos motores, comprados no inicio de Dezembro e entregues somente no final de Janeiro, foi a única situação que quase levou a equipe a tomar uma ação imediata de aceitação do risco, fazendo com que fosse necessário realizar a compra de novos motores. Um dos motivos que agravou a preocupação, e quase fez com que a equipe tomasse uma medida mais radical, foi o fato do rastreamento dos correios não ter sido atualizado desde o envio do produto por parte do vendedor.

O risco “Queima de Componente ou Componente Defeituoso”, risco 4 do Apêndice F, ocorreu durante o projeto algumas vezes com componentes menores, entretanto de fácil reposição. A única ocasião que poderia ter acarretado em algum problema maior foi um dos motores importados ter apresentado defeito de fabricação, entretanto a equipe tinha tomado as devidas precauções comprando motores reservas e com isso o problema foi resolvido de imediato. A “Inexperiência da Tecnologia Utilizada”, risco 5 do Apêndice F, foi outro risco presente no projeto, entretanto, como um período de estudos tinha sido programado no inicio do cronograma a equipe já estava ciente e preparada para a existência deste problema. Não foram encontradas ocorrências evidentes dos outros dois riscos levantados: “Erro nas Escolhas Tecnológicas”, risco 3 do Apêndice F, e “Documentação Errônea das Tecnologias utilizadas”, risco 6 do Apêndice F, durante a execução do projeto.

5.4 Considerações

Apesar de não possuir um escopo completamente definido desde o inicio do projeto, foi possível realizá-lo sem grandes problemas que pudessem impactar o custo ou o cronograma. Em termos de gastos monetários o projeto com custo real de aproximadamente R$2300,00 ficou dentro da faixa estimada, entre R$2000,00 e R$2400,00, mesmo com as mudanças de abordagem realizadas durante a sua execução.

O total de horas trabalhadas (1577 horas) foi superior ao total de horas estimadas (1372 horas), entretanto, como foi possível cumprir todos os objetivos propostos e dentro do prazo, esse erro não apresentou impacto real no resultado final do projeto. Quanto aos riscos não houve nenhum problema realmente sério que tenha impactado no desenvolvimento do robô hexápode. Entretanto, a equipe reconhece ter corrido um alto risco ao importar peças fundamentais do projeto, em uma etapa critica, e que poderia ter acarretado sérios problemas impedindo a conclusão do projeto dentro do prazo.

6 Trabalhos futuros

O controle realizado para a movimentação atualmente é open-loop, ou seja, não envolve realimentação. Dessa forma, quando um movimento é realizado, não há noção precisa de quanto de deslocamento realmente ocorreu. Inicialmente havia a ideia de utilizar um acelerômetro para estimar a distância percorrida, mas percebeu-se que a precisão obtida usando este procedimento é muito ruim. Outra possibilidade era fazer uso de um GPS, o que também foi descartado, pois na maioria das situações o robô é utilizado em ambientes internos. Portanto, atualmente a estimativa de distância é feita somente com base nos movimentos realizados, confiando na relação teoria-prática da cinemática inversa. Diante das dificuldades encontradas, sugere-se para trabalhos futuros procurar alguma forma melhor de realizar a odometria.

Uma boa distribuição de peso é muito importante para que o robô consiga realizar seus movimentos corretamente. No projeto, os motores foram calibrados manualmente neste sentido, o que permitiu apenas eliminar as discrepâncias mais grosseiras entre as patas. Com sensores de pressão nas patas, seria possível avaliar de que forma a distribuição de peso está ocorrendo, e então ajustá-la em tempo de execução. Sugere-se a adição de sensores de pressão para trabalhos futuros.

Na época do início do projeto, não foram encontrados módulos de comunicação sem fio por Wi-Fi, que fossem comercialmente viáveis, então optou-se por utilizar tecnologia ZigBee. Se fosse utilizada conexão por Wi-Fi, seria possível aproveitar a própria infra-estrutura de rede para conectar o driver e o robô. Sugere-se como trabalho futuro utilizar diretamente um hardware Wi-Fi conectado ao robô, que empregaria sockets em lugar do canal de UART.

No projeto atual, tanto a alimentação dos motores como a alimentação da parte lógica são feitas por cabo. Uma evolução natural é a utilização de uma bateria junto do corpo do hexápode, para que ele não tenha de ficar sempre próximo à fonte de energia e limitado pelo alcance do cabo. Entretanto, antes de realizar essa mudança, é preciso primeiramente reavaliar a capacidade dos servos, para verificar se são capazes de suportar o peso adicional. Mesmo assim, sugere-se tal alteração para trabalhos futuros.

Atualmente são utilizados servos de hobby, que embora supram bem as necessidades do projeto em termos de força e precisão, são pouco eficientes: por utilizarem no máximo 6V de tensão de alimentação, o consumo de corrente acaba ficando

muito elevado. Seria uma boa opção utilizar servos com maior torque ou desenvolver servos próprios a partir de motores DC de 12V ou superior, o que sugere-se para trabalhos futuros.

Usou-se no projeto o MicroC/OS-II, que é um sistema operacional de tempo real comercial. Ele é gratuito para projetos com fins educacionais, como é o caso do robô hexápode, mas é preciso comprar licenças para utilizá-lo comercialmente. Uma alternativa seria empregar um sistema operacional gratuito, como é o caso do FreeRTOS.

7 Considerações finais

O objetivo geral do projeto é o desenvolvimento de um robô hexápode implementado em um dispositivo de lógica reconfigurável e controlado remotamente. Além de servir como forma de aplicação dos conhecimentos adquiridos durante o curso de engenharia de computação, o projeto tem alguns objetivos específicos. Ele foi concebido para ser expansível, permitindo a adição de novas funcionalidades sem necessidade de grandes alterações. Outro objetivo específico é a divisão do sistema em três partes: o robô (sistema embarcado), a estação-base, com um driver de comunicação com o robô, e a estação remota, com um software de alto nível que comunica-se com o robô através do driver da estação-base.

O robô desenvolvido é controlado por FPGA, na qual foi utilizado um processador embarcado de arquitetura RISC, o NIOS II, em conjunto com o sistema operacional MicroC/OS-II, desenvolvido pela empresa Micrium. Dentro da FPGA, existem diversos blocos de hardware personalizados ligados ao processador, responsáveis por realizar funções de baixo nível como geração de PWM, cálculo de cinemática inversa, entre outros. Os pinos de I/O da FPGA são conectados a diversos dispositivos periféricos, como o magnetômetro HMC5883, o módulo ZigBee e dezoito optoacopladores. O magnetômetro é utilizado como bússola. O dispositivo ZigBee emula o funcionamento de um canal de UART e é utilizado para comunicação com um driver na estação base, de onde o robô recebe instruções.

A estrutura mecânica do robô é a MSR-H01, desenvolvida pela Micromagic Systems. Esta estrutura é de alumínio e necessita de três servomotores por pata. Os motores utilizados são semelhantes aos indicados na especificação da estrutura mecânica: seis Corona DS329MG nas articulações dos ombros, capazes de gerar um torque de até 3,8 kgf.cm quando alimentados com 4,8 V, e doze BMS-620MG nas demais articulações, capazes de gerar torque de até 9,1 kgf.cm se alimentados com 4,8 V.

O circuito eletrônico do robô é dividido em uma parte de baixa potência (alimentada por uma fonte DC comum 7.5V e 1A), cujo elemento central é a FPGA, e outra de alta potência (alimentada por uma fonte ATX de 140W em regime na linha de 5V), cujos elementos centrais são os dezoito servomotores. Não há ligação eletrônica entre os dois circuitos, que são optoacoplados.

Cada movimento do robô é decomposto em uma série de “setpoints”, dados em coordenadas cartesianas (x, y e z) das exterminadas de cada uma das patas, que passam por algumas etapas antes de se transformarem em sinais de controle para servomotores. A primeira etapa envolve o processamento das coordenadas cartesianas por um bloco de cinemática inversa (por hardware), que as converte em ângulos dos motores. Esses ângulos são convertidos em larguras de pulso, e depois em sinais de PWM por outros blocos, para então serem enviados para os pinos de saída da FPGA. Estes pinos estão ligados a optoacopladores, que funcionam como chaves no circuito de alta potência, resultando em sinais de PWM de maior potência ligados aos servomotores, que só então mudam de posição.

Há quatro elementos de software no projeto como um todo: firmware (software embarcado), driver, biblioteca de comunicação e software de interface gráfica. O firmware controla a máquina de estados do hexápode, enviando para o hardware os movimentos que deverão ser executados pelo robô. O driver é responsável por comunicar-se com o firmware através de um dispositivo ZigBee, recebendo comandos a partir de uma biblioteca de comunicação. A biblioteca de comunicação é utilizada pelo software de interface gráfica para envio de comandos a partir do driver.

Enquanto o firmware foi desenvolvido em linguagem C++, os demais elementos foram desenvolvidos em linguagem Java. Todos os elementos de software foram construídos sobre uma mesma biblioteca de baixo nível, cuja implementação é diferente em cada linguagem. Em ambas as linguagens empregam-se meios de gerenciamento automático de memória: garbage collection em Java (o que é nativo da máquina virtual Java) e contagem de referências em C++ (o que faz parte do código desenvolvido).O código C++ ainda utiliza um mecanismo próprio de alocação de memória denominado object pooling, que preserva as vantagens da alocação dinâmica e da alocação estática, sem ter o não-determinismo inerente da alocação dinâmica.

O desenvolvimento do projeto robô seguiu o modelo em espiral e foi feito de forma modular, o que permitiu que diversas etapas fossem realizadas em paralelo. O tempo total de trabalho em horas (somando os tempos gastos pelos três membros) excedeu em aproximadamente 15% o estimado, totalizando 1577 horas (de 1372 estimadas). Ao término do desenvolvimento o plano de testes, que contemplou todos os requisitos funcionais do projeto, foi executado com sucesso sem que houvesse maiores problemas.

O projeto exigiu grande parte dos conhecimentos adquiridos durante a formação de engenharia de computação, especialmente referentes às disciplinas de Sistemas

Embarcados, Lógica Reconfigurável, Engenharia de Software, Eletrônica II, Redes de Computadores e Comunicação de Dados. Conhecimentos não relacionados às disciplinas da grade curricular também foram necessários, como a cinemática inversa e o projeto da PCI de alta potência.

Geralmente robôs hexápodes mais robustos têm custos elevados e documentação fechada. O robô desenvolvido tem documentação completamente aberta em termos de software e hardware, apesar de manter um custo elevado (cerca de R$ 2400,00). O projeto ainda destaca-se por permitir expansões futuras: o hardware personalizado na FPGA pode ser alterado e o software é modular e bem documentado.

Referências

[1] **RoboCup.** Site oficial. Acessível em: <http://www.robocup.org/>. Acessado em 20/04/2013.

[2] BÖTTCHER, S. (2006). **Principles of robot locomotion.** Em Proc. Human Robot Interaction Seminar SS2006, Univ. de Dortmund.

[3] SOYGUDER, S.,, ALLI, H. (2007). **Design and prototype of a six-legged walking insect robot.** Em Industrial Robot: An International Journal, v. 34, p. 412 - 422.

[4] **BigDog - The Most Advanced Rough-Terrain Robot on Earth.** Boston Dynamics. Site oficial. Acessado em 5 de abril de 2011. Acessível em: <http://www.bostondynamics.com/robot_bigdog.html>.

[5] WANG, Z.-Y, DINGA, X.-L., ROVETTA, A. (2010). **Analysis of typical locomotion of a symmetric hexapod robot.** Em Robotica, v. 28, n..6.

[6] CAMPOS, R., MATOS, V., SANTOS, C. (2010). **Hexapod Locomotion: a Nonlinear Dynamical Systems Approach.** Em IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society, p. 1546 - 1551. ISBN: 9781424452255.

[7] PAP, Z., KECSKÉS, I., BURKUS, E., BAZSÓ, F., ODTY, P. (2010). **Optimization of the Hexapod Robot Walking by Genetic Algorithm.** Em Intelligent Systems and Informatics (SISY), 2010 8th International Symposium on 2010. p. 121 - 126. ISBN: 781424473946.

[8] GO, Y. BOWLING, A. (2005). **A Design Study of a Cable-Driven Hexapod.** Em Intelligent Robots and Systems / IEEE/RSJ International Conference, p. 671 - 678. ISBN: 0780389123.

[9] HOLLINGUM J. (1997). **Hexapods to take over?** Em Industrial Robot: An International Journal, v. 24, p. 428 - 431.

[10] BLAISE, J., BONEV, I., MONSARRAT, B., BRIOT, S., LAMBERT, J.M., PERRON, C. (2010). **Kinematic characterisation of hexapods for industry.** Em Industrial Robot: An International Journal, v. 37, p. 79 - 88.

[11] KURPIEL, F.D.; DO NASCIMENTO, L.P.; HIGASKINO, M.M.K., DA COSTA, R.F. (2010). **Robô Omni2 - Sistema de Navegação do Robô Omnidirecional.** Monografia. Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba, 2010.

[12] **Eclipse - The Eclipse Foundation open source community website.** Site.

Acessado em 26 de abril de 2011. Acessível em: <http://www.eclipse.org>.

[13] REDA, S. (2007). **Reconfigurable Computing.** Notas de Aula. Divisão de Engenharia, Universidade de Brown.

[14] **FPGA DE0-Nano.** Acessado em 31 de janeiro de 2013. Acessível em: <http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html>.

[15] **Java Language and Virtual Machine Specifications.** Em Oracle. Acessível em: <http://docs.oracle.com/javase/specs/>. Acessado em 9 de abril de 2013.

[16] **Quartus II Web Edition Software.** Site oficial. Acessado em 31 de janeiro de 2013. Acessado em: <http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html>.

[17] **CadSoft Online: EAGLE Product information.** Site oficial. Acessado em 26 de abril de 2011. Acessível em: <http://www.cadsoft.de/info.htm>.

[18] **The Official Stiquito Homepage.** Site oficial. Acessível em: <http://www.stiquito.com>. Acessado em 22 de janeiro de 2013.

[19] SARANLI, U., BUEHLER, M, KODITSCHEK, D.E. (2001). **RHex: A Simple and Highly Mobile Hexapod Robot.** Em International Journal of Robotics Research, vol. 20, no. 7, pp. 616-631.

[20] **RHex - Devours Rough Terrain.** Boston Dynamics. Site oficial. Acessível em <http://www.bostondynamics.com/robot_rhex.html>. Acessado em 16 de fevereiro de 2013.

[21] **Lynxmotion Robot Kits.** Acessível em <http://www.lynxmotion.com>. Acessado em 22 de janeiro de 2013.

[22] **Hexy - Open-Source, Low-Cost Hexapod.** Em ArcBotics - Igniting Open Robotics. Acessível em <http://arcbotics.com/products/hexy>. Acessado em 22 de janeiro de 2013.

[23] LOPES, C., CALORY FILHO, P., TEREZIN, T.A. (2010). **Robô Hexápode Orientado para Aplicações Pedagógicas.** Trabalho de Conclusão de Curso (Graduação em Engenharia Eletrônica) - Universidade Tecnológica Federal do Paraná.

[24] **1 USD IN BRL.** Google. Site. Acessado em 26 de abril de 2011. Acessível em: <http://www.google.com.br/search?q=1+usd+in+brl&ie=utf-8&oe=utf-8>.

[25] MEIRA. W.H.T., GIOPPO, L.L., FONTOURA, F.M., LANES JUNIOR, E.A. ,MARÇAL, M.S. (2011). **FunIcon Prancha Eletrônica.** Monografia. Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba, 2010.

[26] **FPGAs. What is an FPGA?.** Acessado em 31 de janeiro de 2013. Acessível em: <http://www.altera.com/products/fpga.html>.

[27] **FPGA Fundamentals.** Em National Instruments. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.ni.com/white-paper/6983/en>.

[28] **Nios II Processor Reference Handbook.** Manual. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf>.

[29] **RTOS Concepts.** Em ChibiOS/RT Homepage. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.chibios.org/dokuwiki/doku.php?id=chibios:articles:rtos_concepts>.

[30] **μC/OS-II – The Real-Time Kernel.** Micrium. Site oficial. Acessado em 31 de janeiro de 2013. Acessível em: <http://micrium.com/rtos/ucosii/overview/>.

[31] **Software Development Tools for the Nios II Processor.** Altera. Site oficial. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.altera.com/devices/processor/nios2/tools/ide/ni2-ide.html>.

[32] **Understanding ZigBee.** ZigBee Alliance. Site oficial Acessado em 31 de janeiro de 2013. Acessível em: <http://www.zigbee.org/About/UnderstandingZigBee.aspx>.

[33] **Digi XBee Wireless RF Modules.** Digi International. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.digi.com/xbee/>.

[34] **XBee USB Adapter Board.** Parallax Inc. Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.parallax.com/Store/Accessories/CommunicationRF/tabid/161/ProductID/643/List/0/Default.aspx>.

[35] **Xbee Adapter Board.** Parallax Inc. Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.parallax.com/StoreSearchResults/tabid/768/txtSearch/xbee/List/0/SortField/4/ProductID/642/Default.aspx>.

[36] **X-CTU Software.** Digi International. Site oficial. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.digi.com/support/productdetail?pid=3352>.

[37] **A beginner's guide to accelerometers.** Dimension Engineering. Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.dimensionengineering.com/info/accelerometers>.

[38] **ADXL345.** Analog Devices. Datasheet. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf>.

[39] **How stuff works? - Magnetometer.** HowStuffWorks. Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://science.howstuffworks.com/magnetometer-info.htm>.

[40] **HMC5883L.** Honeywell. Datasheet. Acessado em 31 de janeiro de 2013. Acessível em: <http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf>.

[41] **MSR-H01 Hexapod Kit.** Micromagic Systems. Site oficial. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.hexapodrobot.com/products/robots/Hexapod_MSR-H01.html>.

[42] **AN-556 Introduction to Power Supplies.** Texas Instruments. Application Report. Acessado em 10 de fevereiro de 2013. Acessível em: <http://www.ti.com/lit/an/snva006b/snva006b.pdf>.

[43] **BMS-620MG – Particular Specification.** Blue Bird Technology. Em BP Hobbies. Acessado em 10 de fevereiro de 2013. Acessível em: <http://www.bphobbies.com/pdf/BMS-620MG.pdf>.

[44] **Corona DS329MG.** Corona. Site Oficial. Acessado em 10 de fevereiro de 2013. Acessível em: <http://www.corona-rc.com/coproductshowE.asp?ArticleID=179>.

[45] CRAIG, J.J. (2004). **Introduction to Robotics.** 3ª edição, páginas: 28; 62; 90. Pearson Prentice Hall.

[46] BUSS, S.R. (2009). **Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods.** Departamento de Matemática, Universidade da Califórnia, San Diego. Acessado em 1 de abril de 2013. Acessível em: <http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf>.

[47] FISCHER, M. (2007). **Inverse Kinematik.** Notas de Aula. FH München.

[48] **Types of Robot Gait.** Hexapod Robots - The Future of Robotics. site. Acessado em 05/05/2013. Acessível em: <http://hexapodrobots.weebly.com/types-of-robot-gait.html>.

[49] **Analysis of Multi-Legged Animal + Robot Gaits.** Oricom Technologies Site. Acessado em 05/05/2013. Acessível em: <http://www.oricomtech.com/projects/leg-time.htm>.

[50] SAWICZ, D. **Hobby Servo Fundamentals.** Artigo técnico. Acessado em 1 de fevereiro de 2013. Acessado em:

<http://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf>.

[51] **I2C-Bus: What's that?** Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.i2c-bus.org/>.

[52] **I2C Background.** Total Phase. Artigo técnico. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.totalphase.com/support/kb/10037/>.

[53] **I2C.** Mikrocontroller.net. Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.mikrocontroller.net/articles/I2C>.

[54] WILSON, P.R., JOHNSTONE, M.S., NEELY, M., BOLES, D. (1995). **Dynamic Storage Allocation: A Survey and Critical Review.** Em IWMM '95 Proceedings of the International Workshop on Memory Management, págs 1-116. Springer-Verlag.

[55] RUGGIERI, C., MURTAGH, T.P. (1988). **Lifetime analysis of dynamically allocated objects.** Em POPL '88 Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, páginas 285-293. ACM.

[56] SAKS, D. (2008). **The yin and yang of dynamic allocation.** Embedded. Artigo. Acessível em: <http://www.embedded.com/4007569>. Acessado em 9 de abril de 2013.

[57] WILTAMUTH, S., HEJLSBERG, A (2003). **C# Language Specification.** MSDN. Especificação. Acessível em: <http://msdn.microsoft.com/en-us/library/aa645596.aspx>. Acessado em 9 de abril de 2013.

[58] PITTS, R.I. **Intro to Dynamic Allocation.** Material didático. Em Boston University Computer Science Teaching Material Archive. Acessível em <http://www.cs.bu.edu/teaching/c/alloc/intro/>. Acessado em 9 de abril de 2013.

[59] DOUGLASS, B.P. (2003). **Real-Time Design Patterns: Robust Scalable Architecture for Real-time Systems.** Livro. Addison-Wesley Professional.

[60] NYSTROM, R (2011). **Optimizing Patterns: Object Pool.** Game Programming Patterns. Acessado em 28 de fevereiro de 2013. Acessível em <http://gameprogrammingpatterns.com/object-pool.html>. Acessado em 9 de abril de 2013.

[61] **Creational Patterns: Object Pool.** OODesign: Object Oriented Design. Site. Acessível em: <http://www.oodesign.com/object-pool-pattern.html>. Acessado em 9 de abril de 2013.

[62] XIE, Y., AIKEN, A. (2005). **Context- and Path-sensitive Memory Leak Detection.** Em Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software

engineering, páginas 115 - 125. ACM.

[63] KULKARNI, M. (2012). **Pointer Checker to detect buffer overflows and dangling pointers (part 2).** Em Intel Software. Acessível em: <http://software.intel.com/en-us/articles/pointer-checker-to-detect-buffer-problems-and-dangling-pointers-part-2>. Acessado em 9 de abril de 2013.

[64] MYTTON, D. (2004). **Why You Need Coding Standards.** Artigo. Em sitepoint. Acessível em <http://www.sitepoint.com/coding-standards/>. Acessado em 9 de abril de 2013.

[65] **Java Code Conventions** (1997). Sun Microsystems. Especificação. Acessível em <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>. Acessado em 9 de abril de 2013.

[66] JONES, D. (2008). **Operand names influence operator precedence decisions (part 1 of 2) - Experiment performed at the 2007 ACCU Conference.** Acessível em <http://www.knosof.co.uk/cbook/oprandname.pdf>. Acessado em 9 de abril de 2013.

[67] **GNU Coding Standards.** Free Software Foundation. Especificação. Última atualização em 8 de março de 2013. Acessível em <http://www.gnu.org/prep/standards/>. Acessado em 9 de abril de 2013.

[68] WEINBERGER, B., SIVERSTEIN, C., EITZMANN, G. MENTOVAI, M., LANDRAY, T. **Google C++ Style Guide.** Especificação. Revisão 3245. Acessível em <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>. Acessado em 9 de abril de 2013.

[69] **Documentation: SOPC Builder.** Altera. Documentação. Acessado em 13 de fevereiro de 2013. Acessível em: <http://www.altera.com/literature/lit-sop.jsp>.

[70] **Memory System Design.** Altera. Documentação. Acessado em 14 de fevereiro de 2013. Acessível em: <http://www.altera.com/literature/hb/nios2/edh_ed51008.pdf>.

[71] **DE0-Nano | NiosII with SDRAM.** eStuffz. Site. Acessado em 14 de fevereiro de 2013. Acessível em: <https://sites.google.com/site/fpgaandco/de0-nano-niosii-with-sdram>.

[72] **I2C (OpenCores).** Altera Wiki. Site. Acessado em 11 de fevereiro de 2013. Acessível em: <http://www.alterawiki.com/wiki/I2C_(OpenCores)>.

[73] **Robotics Toolbox.** PeterCorke. Site. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.petercorke.com/Robotics_Toolbox.html>.

[74] **Denavit-Hartenberg-Notation.** Acessado em 31 de janeiro de 2013. Acessível em:

<https://www.fh-muenster.de/fb11/labore/forschung/robotertechnik/downloads/Denavit_Hartenberg_Notation.pdf>.

[75] **VHDL function for finding square root.** VHDL coding tips and tricks. Site. Acessado em 31 de janeiro de 2013. Acessado em: <http://vhdlguru.blogspot.com.br/2010/03/vhdl-function-for-finding-square-root.html>.

[76] BARR, M. (2007). **Introduction to Pulse Width Modulation (PWM).** BarrGroup. Site. Acessado em 10 de fevereiro de 2013. Acessível em: <http://www.barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation>.

[77] **SOPC Builder User Guide.** Altera. Manual. Acessado em 31 de janeiro de 2013. Acessível em: <http://www.altera.com/literature/ug/ug_sopc_builder.pdf>.

[78] **OpenCores.** Site. Acessado em 11 de fevereiro de 2013. Acessível em: <http://opencores.org/>.

[79] **I2C controller core :: Overview.** OpenCores. Site. Acessado em 11 de fevereiro de 2013. Acessível em: <http://opencores.org/project,i2c>.

[80] **Wishbone B4 - WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores.** OpenCores. Documentação. Acessado em 11 de fevereiro de 2013. Acessível em: <http://cdn.opencores.org/downloads/wbspec_b4.pdf>.

[81] **Avalon Interface - Specifications.** Altera. Documentação. Acessado em 11 de fevereiro de 2013. Acessível em: <http://www.altera.com/literature/manual/mnl_avalon_spec.pdf>.

[82] HERVEILLE, R. (2003). **I2C-Master Core Specification.** OpenCores. Documentação. Acessado em 1 de fevereiro de 2013. Acessível em: <http://opencores.org/websvn,filedetails?repname=i2c&path=%2Fi2c%2Ftrunk%2Fdoc%2Fi2c_specs.pdf>.

[83] **Servo Motor Motortech.** Motor tech. Site oficial. Acessado em 16 de abril de 2013. Acessado em: <http://www.motortech.com.br/index.php/produtos/servo-motor>.

[84] **PCB Trace Width Calculator.** Reference Designer. Site. Acessado em 18 de abril de 2013. Acessível em: <http://www.referencedesigner.com/cal/cal_06.php>.

[85] **Convert an ATX PSU to a Bench Power Supply.** Basic Electronics Tutorials Blog. Site. Acessado em 17 de fevereiro de 2013. Acessível em: <http://www.electronics-tutorials.ws/blog/convert-atx-psu-to-bench-supply.html>.

[86] **MSR-H01 Assembly Guide.** Micromagic Systems. Guia. Acessado em 10 de abril

de 2013. Acessível em:<http://www.hexapodrobot.com/Files/Guides/MSR-H01_Assembly.pdf>.

[87] **Tabela de condutores de cobre.** Departamento de Engenharia de Materiais - USP. Material didático. Acessado em 19 de abril de 2013. Acessível em:<http://www.demar.eel.usp.br/eletronica/aulas/Tabela_condutores_cobre.pdf>.

[88] MAZIERO, C.A. (2011). Sistemas Operacionais (incompleto). Livro aberto.

[89] HEBEL, M. BRICKER, G. HARRIS, D. (2010). **Getting Started with XBee RF Modules - A Tutorial for BASIC Stamp and Propeller Microcontrollers**, v. 1.0. Parallax Inc.

[90] **RFC 793: Transmission Control Protocol - DARPA Internet Program Protocol Specification** (1981). Internet Engineering Task Force (IETF). Especificação técnica.

[91] COULOURIS, G.F. DOLLIMORE, J. KINDBERG, T. (2001). **Distributed Systems: Concepts and Design.** 3ª edição. Pearson Education.

[92] MAZIÈRES, D. **My Rant on C++'s operator new.** Artigo técnico. Acessível em <http://www.scs.stanford.edu/~dm/home/papers/c++-new.html>. Acessado em 9 de abril de 2013.

[93] **nrjavaserial - Neuron Robotics Java Serial Library.** Google Project Hosting. Site. Acessível em <http://code.google.com/p/nrjavaserial/>. Acessado em 20 de abril de 2013.

Apêndice A: Movimentos suportados pelo robô

Movimento	Código de tipo	Parâmetros	Descrição
Andar	*0xACE*	Passos à frente (4 bytes)	O robô anda em frente um certo número de passos. Se o número de passos for negativo, o robô anda para trás.
Andar de lado	*0xC0A*	Passos à direita (4 bytes)	O robô anda para a direita um certo número de passos. Se o número de passos for negativo, o robô anda para a esquerda.
Girar	*0xCAB*	Ângulo × 1024 (4 bytes)	O robô gira um certo número de ângulos no sentido horário. Se o número de ângulos for negativo, o robô gira no sentido anti-horário.
Olhar para	*0xB0A*	Ângulo × 1024 (4 bytes)	O robô gira para um certo ângulo no sentido horário em relação ao leste. Se o número de ângulos for negativo, o ângulo de destino é no sentido anti-horário em relação ao leste.
Andar para	*0xFACA*	Ângulo × 1024 (4 bytes) Passos à frente (4 bytes)	O robô caminha em frente um certo número de passos na direção a certo ângulo no sentido horário em relação ao leste. Se o número de ângulos for negativo, o ângulo de destino é no sentido anti-horário em relação ao leste. Se o número de passos for negativo, o robô anda para trás.
Bambolear	*0xC0CA*	Número de ciclos (4 bytes)	O robô faz movimento de bambolear um certo número de vezes.
Fazer flexões	*0xB0DE*	Número de flexões (4 bytes)	O robô faz um certo número de flexões.
Calibrar/ajustar	*0xBEC0*	Nenhum	O robô gira no próprio eixo para calibrar o magnetômetro.

Apêndice B: Sensores suportados pelo robô

Sensor	Código de tipo	Formato da leitura	Descrição
Acelerômetro	*0xBEBA*	Leitura em X (2 bytes) Leitura em Y (2 bytes) Leitura em Z (2 bytes)	Mede a aceleração nos três eixos, em cm/s².
Magnetômetro	*0xC0CA*	Leitura em X (2 bytes) Leitura em Y (2 bytes) Leitura em Z (2 bytes) Ângulo × 64 (2 bytes)	Mede o campo magnético nos três eixos, e o ângulo em relação ao leste.

Apêndice C: Mensagens do protocolo entre robô e driver

<table>
<tr><th colspan="4">Mensagens do driver para o robô</th></tr>
<tr><th>Mensagem</th><th>Código</th><th>Corpo</th><th>Descrição</th></tr>
<tr><td rowspan="2">Handshake</td><td>0x0000 (requisição)</td><td>Vazio</td><td rowspan="2">Verificar se há conexão com o robô.</td></tr>
<tr><td>0x0001 (resposta)</td><td>Vazio</td></tr>
<tr><td rowspan="2">CheckStatus</td><td>0x0002 (requisição)</td><td>Vazio</td><td rowspan="2">Verificar estado do robô.
Resultados:
0x00 = parado
0xFF = movimento em andamento</td></tr>
<tr><td>0x0003 (resposta)</td><td>Estado (1 byte)</td></tr>
<tr><td rowspan="2">Halt</td><td>0x0020 (requisição)</td><td>Vazio</td><td rowspan="2">Faz o movimento atual do robô parar.
O movimento em andamento náo para no instante que a mensagem é respondida, mas sim após o término do ciclo atual.</td></tr>
<tr><td>0x0021 (resposta)</td><td>Vazio</td></tr>
<tr><td rowspan="2">SetMovement</td><td>0x0022 (requisição)</td><td>Movimento (2 bytes)
Identificador (4 bytes)
Parâmetros (variável)</td><td rowspan="2">Definir o movimento seguinte do robô, associado a um identificador de 4 bytes definido pelo cliente.
Caso o movimento não seja compreendido pelo robô, é enviada uma resposta com tipo 0x0021 (halt).</td></tr>
<tr><td>0x0023 (resposta)</td><td>Vazio</td></tr>
<tr><td rowspan="2">Move</td><td>0x0024 (requisição)</td><td>Vazio</td><td rowspan="2">Inicia a execução do movimento seguinte do robô e limpa a variável que armazena o movimento seguinte.
Caso não haja movimento seguinte, essa operação não tem efeito.</td></tr>
<tr><td>0x0025 (resposta)</td><td>Vazio</td></tr>
<tr><td rowspan="2">FetchSensor</td><td>0x0040 (requisição)</td><td>Sensor (2 bytes)</td><td rowspan="2">Faz a leitura de um sensor. O valor lido não é o valor no instante do recebimento da mensagem, mas sim o valor da última leitura feita pelo robô.
Caso o robô não tenha o sensor, a resposta é enviada com corpo vazio.</td></tr>
<tr><td>0x0041 (resposta)</td><td>Leitura (variável)</td></tr>
<tr><td rowspan="2">ErrorNotification</td><td>0x00FE (requisição)</td><td>Vazio</td><td rowspan="2">Indica que ocorreu um erro com o driver.
Essa mensagem não é utilizada na versão atual do protocolo.</td></tr>
<tr><td>0x00FF (resposta)</td><td>Vazio</td></tr>
</table>

<table>
<tr><th colspan="4">Mensagens do robô para o driver</th></tr>
<tr><th>Mensagem</th><th>Código</th><th>Corpo</th><th>Descrição</th></tr>
<tr><td rowspan="2">KeepAlive</td><td>0x0100 (requisição)</td><td>Vazio</td><td rowspan="2">Verificar se há conexão com o driver.
Essa mensagem não é utilizada na versão atual do protocolo.</td></tr>
<tr><td>0x0101 (resposta)</td><td>Vazio</td></tr>
<tr><td rowspan="2">Movement Finished Notification</td><td>0x0102 (requisição)</td><td>Identificador (4 bytes)
Tamanho (4 bytes)</td><td rowspan="2">Indica que um movimento associado a um identificador de 4 bytes definido pelo cliente foi terminado com sucesso. A mensagem também acompanha um valor numérico que indica o tamanho do movimento, cuja interpretação varia de um movimento para outro.</td></tr>
<tr><td>0x0103 (resposta)</td><td>Vazio</td></tr>
<tr><td rowspan="2">Movement Aborted Notification</td><td>0x0104 (requisição)</td><td>Identificador (4 bytes)
Valor (4 bytes)
Tamanho (4 bytes)</td><td rowspan="2">Indica que um movimento associado a um identificador de 4 bytes definido pelo cliente foi abortado. A mensagem também acompanha valores numéricos que indicam o valor atual do movimento e seu tamanho, cuja interpretação varia de um movimento para outro.</td></tr>
<tr><td>0x0105 (resposta)</td><td>Vazio</td></tr>
</table>

Apêndice D: Mensagens do protocolo entre cliente e driver

<table>
<tr><th colspan="4">Mensagens do cliente para o driver</th></tr>
<tr><th>Mensagem</th><th>Código</th><th>Corpo</th><th>Descrição</th></tr>
<tr><td rowspan="2">EnumeratePorts</td><td>0x8000
(requisição)</td><td>Vazio</td><td rowspan="2">Enumerar as portas disponíveis no computador no qual o driver está sendo executado.
A versão atual do robô opera sobre canais seriais apenas, então essa mensagem enumera as portas seriais.
A lista de portas é codificada em ASCII e as portas são separadas por `\n`.</td></tr>
<tr><td>0x8001
(resposta)</td><td>Lista (variável)</td></tr>
<tr><td rowspan="2">OpenPort</td><td>0x8002
(requisição)</td><td>Porta (variável)</td><td rowspan="2">Tenta estabelecer a conexão com o robô conectado a certa porta. O nome da porta é codificado em ASCII.
Resultados:
0x00 = sucesso
0x01 = timeout
0x02 = já estava conectado
0x04 = porta inválida</td></tr>
<tr><td>0x8003
(resposta)</td><td>Resultado (1 byte)</td></tr>
<tr><td rowspan="2">BeginMovement</td><td>0x8080
(requisição)</td><td>Movimento (2 bytes)
Identificador (4 bytes)
Parâmetros (variável)</td><td rowspan="2">Tenta comandar o robô para iniciar um movimento. O movimento é codificado da mesma forma que no protocolo entre o driver e o robô.
Resultados:
0x00 = sucesso
0x01 = timeout
0x02 = não está conectado ao robô
0x04 = movimento desconhecido/inválido</td></tr>
<tr><td>0x8081
(resposta)</td><td>Resultado (1 byte)</td></tr>
<tr><td rowspan="2">HaltMovement</td><td>0x8082
(requisição)</td><td>Vazio</td><td rowspan="2">Tenta comandar o robô para parar o movimento atual.
Resultados:
0x00 = sucesso
0x01 = timeout
0x02 = não está conectado ao robô</td></tr>
<tr><td>0x8083
(resposta)</td><td>Resultado (1 byte)</td></tr>
<tr><td rowspan="2">ReadSensor</td><td>0x80C0
(requisição)</td><td>Sensor (2 bytes)</td><td rowspan="2">Tenta fazer a leitura de um sensor do robô.
Resultados:
0x00 = sucesso
0x01 = timeout
0x02 = não está conectado ao robô.
0x04 = sensor desconhecido/inválido</td></tr>
<tr><td>0x80C1
(resposta)</td><td>Resultado (1 byte)
Leitura (variável)</td></tr>
<tr><td rowspan="2">ErrorNotification</td><td>0x80FE
(requisição)</td><td>Vazio</td><td rowspan="2">Indica que ocorreu um erro com o cliente.
Essa mensagem não é utilizada na versão atual do protocolo.</td></tr>
<tr><td>0x80FF
(resposta)</td><td>Vazio</td></tr>
</table>

Mensagens do driver para o cliente			
Mensagem	**Código**	**Corpo**	**Descrição**
Ping	*0xC000 (requisição)*	Vazio	Verificar se há conexão com o cliente.
	0xC001 (resposta)	Vazio	
Movement Finished Notification	*0xC102 (requisição)*	Identificador (4 bytes) Tamanho (4 bytes)	Indica que um movimento associado a um identificador de 4 bytes definido pelo cliente foi terminado com sucesso. A mensagem também acompanha um valor numérico que indica o tamanho do movimento, cuja interpretação varia de um movimento para outro.
	0xC103 (resposta)	Vazio	
Movement Aborted Notification	*0xC104 (requisição)*	Identificador (4 bytes) Valor (4 bytes) Tamanho (4 bytes)	Indica que um movimento associado a um identificador de 4 bytes definido pelo cliente foi abortado. A mensagem também acompanha valores numéricos que indicam o valor atual do movimento e seu tamanho, cuja interpretação varia de um movimento para outro.
	0xC105 (requisição)	Vazio	

Apêndice E: Tabela completa de atividades realizadas

Grupo	Tarefa	Status	Data de Início	Data de Término	Horas Estimadas	Horas Trabalhadas
Realizar estudos diversos	Realizar estudos diversos	100%	05/03/2012	20/02/2013	45,0	45,0
Escrever o relatório	Escrever o relatório	100%	05/03/2012	30/04/2013	100,0	144,0
Gerenciar o projeto	Realizar reuniões e ajustes de cronograma	100%	05/03/2012	30/04/2013	80,0	80,0
Comprar componentes	Comprar FPGA	100%	05/03/2012	05/03/2012	3,0	3,0
	Comprar módulo XBee	100%	05/03/2012	05/03/2012	3,0	2,0
	Comprar os servos de teste	100%	20/08/2012	20/08/2012	3,0	4,0
	Comprar componentes da placa de teste	100%	20/09/2012	20/09/2012	3,0	6,0
	Comprar os sensores	100%	08/01/2013	08/01/2013	3,0	4,0
	Comprar os optoacopladores	100%	18/01/2013	18/01/2013	3,0	5,0
	Comprar a estrutura mecânica [importada]	100%	26/11/2012	26/11/2012	3,0	5,0
	Comprar os servos definitivos [importados]	100%	06/12/2012	06/12/2012	3,0	7,0
	Comprar os componentes da PCI final	100%	21/02/2013	21/02/2013	4,0	4,0
Elaborar e testar os "servos caseiros"	Definir o controle do servo	100%	04/09/2012	09/04/2012	4,0	4,0
	Criar o diagrama de controle do servo caseiro	100%	13/08/2012	18/08/2012	4,0	4,0
	Realizar testes simples com os servos	100%	06/09/2012	03/10/2012	6,0	6,0
	Projetar a malha caseira de controle com o 555	100%	24/09/2012	14/10/2012	12,0	12,0
	Testar a malha de controle com os servos abertos	100%	22/10/2012	02/11/2012	14,0	14,0
	Elaborar e testar a placa simples [soldada] com o controle do servo	100%	27/10/2012	02/11/2012	40,0	40,0
Desenvolver a arquitetura de controle da FPGA	Desenvolver bloco de controle do servo	100%	30/07/2012	03/08/2012	2,0	12,0
	Desenvolver bloco de controle de uma junta	100%	01/10/2012	05/10/2012	4,0	6,0
	Desenvolver bloco de controle de uma pata	100%	08/10/2012	12/11/2012	24,0	24,0
	Desenvolver o interfaceamento serial com o NIOS	100%	01/10/2012	05/10/2012	2,0	2,0
	Desenvolver o interfaceamento SPI com o NIOS	100%	01/10/2012	12/10/2012	8,0	8,0
	Ajustar o NIOS com os blocos de controle	100%	01/10/2012	15/01/2013	20,0	20,0
	Desenvolver o Interfaceamento do I2C com NIOS	100%	01/10/2013	20/01/2013	10,0	12,0
Implementar as funções básicas do NIOS	Realizar configurações básicas do NIOS	100%	17/09/2012	24/09/2012	12,0	12,0
	Implementar a leitura de dados do acelerômetro	100%	01/10/2012	12/10/2012	4,0	4,0
	Escrever as coordenadas de deslocamento das patas	100%	05/11/2012	23/12/2012	14,0	10,0
	Utilizar um RTOS embarcado como SO no NIOS	100%	14/12/2012	23/12/2012	6,0	3,0
Elaborar o mecanismo da cinemática inversa	Elaborar o modelo 1 [solução incremental]	100%	23/10/2012	23/10/2012	8,0	8,0
	Implementar em C o modelo 1	100%	24/10/2012	31/10/2012	24,0	24,0
	Testar o modelo 1 na estação base	100%	29/10/2012	31/10/2012	4,0	4,0
	Testar o modelo 1 no NIOS	100%	31/10/2012	12/11/2012	4,0	4,0
	Otimizar o modelo 1 para o NIOS	100%	31/10/2012	12/11/2012	12,0	12,0
	Elaborar o modelo 2 [solução geométrica]	100%	17/11/2012	19/11/2012	4,0	4,0
	Implementar em C o modelo 2	100%	19/11/2012	19/11/2012	2,0	2,0
	Testar o modelo 2 no NIOS	100%	20/11/2012	20/11/2012	2,0	2,0
	Implementar o modelo 2 em VHDL	100%	20/11/2012	27/11/2012	16,0	16,0
	Testar o modelo 2 em VHDL	100%	20/11/2012	06/12/2012	4,0	4,0
	Otimizar o modelo 2 em VHDL	100%	20/11/2012	06/12/2012	8,0	8,0

Elaborar os diagramas de Hardware	Organizar os diagramas do circuitos internos da FPGA	100%	16/02/2013	25/02/2013	1,0	1,0
	Elaborar o diagrama geral da arquitetura do sistema	100%	08/10/2012	10/12/2012	3,0	3,0
	Elaborar o diagrama de blocos do circuito eletrônico	100%	08/10/2012	10/12/2012	2,0	2,0
	Organizar os diagramas da PCI	100%	16/02/2013	18/04/2013	1,0	1,0
	Elaborar um fluxograma completo de um comando	100%	16/02/2013	18/04/2013	8,0	6,0
Desenvolver a comunicação da estação base com a FPGA	Implementar o código em ANSI-C	100%	21/10/2012	14/11/2012	16,0	16,0
	Portar o código existente para C++	100%	14/11/2012	03/12/2012	4,0	4,0
	Elaborar as classes base do sistema	100%	14/11/2012	22/12/2012	8,0	8,0
	Implementar e testar o código em Windows	100%	21/10/2012	22/12/2012	36,0	36,0
	Elaborar o protocolo de baixo nivel	100%	01/12/2012	03/12/2012	8,0	8,0
	Implementar o protocolo de baixo nivel	100%	03/12/2012	22/12/2012	8,0	10,0
	Implementar o protocolo de alto nível do driver em C++	100%	04/02/2013	12/02/2013	6,0	20,0
	Traduzir o protocolo de alto nível do driver para Java	100%	12/02/2013	20/02/2013	4,0	10,0
	Implementar o protocolo de alto nível do driver em Java	100%	20/02/2013	28/02/2013	12,0	12,0
	Implementar as classes base em RTOS embarcado	100%	15/12/2012	28/02/2013	20,0	12,0
	Ilmplementar o protocolo de alto nível no sistema embarcado	100%	20/02/2013	10/03/2013	20,0	28,0
Projetar o protocolo de alto nível	Elaborar o protocolo de alto nivel entre sistema embarcado e o driver	100%	20/10/2012	05/02/2013	8,0	21,0
	Elaborar o protocolo de alto nivel entre driver e cliente	100%	20/10/2012	05/02/2013	8,0	14,0
Desenvolver a comunicação entre driver e software de interface	Traduzir de C++ para Java as classes de comunicação	100%	15/12/2012	18/01/2013	30,0	12,0
Projetar a PCI de distribuição, alimentação e sensores	Elaborar circuito de interface da FPGA com sensores e atuadores	100%	15/12/2012	06/01/2013	8,0	4,0
	Elaborar circuito de alimentação do hexápode	100%	06/01/2013	13/01/2013	8,0	6,0
	Criar o projeto das PCIs no Eagle	100%	14/01/2013	25/01/2013	36,0	65,0
	Realizar testes, definir e realizar ajustes no diagrama	100%	22/01/2013	30/01/2013	24,0	30,0
	Encomendar a PCI	100%	04/02/2013	04/02/2013	4,0	3,0
Definir e testar uma fonte de alimentação	Estimar o gasto de energia total e de cada parte do hexápode	100%	02/01/2013	09/01/2013	4,0	4,0
	Definir uma fonte de alimentação para ser utilizada	100%	09/01/2013	13/01/2013	12,0	8,0
	Adquirir uma fonte de alimentação	100%	13/01/2013	31/01/2013	4,0	3,0
	Integrar e testar a fonte com o sistema	100%	15/02/2013	23/02/2013	8,0	11,0
	Projetar e confeccionar o cabo extensor	100%	15/02/2013	23/02/2013	8,0	18,0
Montar o hexápode	Montar das partes mecânicas	100%	02/02/2013	03/02/2013	9,0	18,0
	Encaixar e regular os servos na estrutura mecânica	100%	07/02/2013	07/02/2013	15,0	10,0
	Adicionar as PCIs a estrutura mecânica	100%	20/02/2013	23/02/2013	6,0	6,0
	Testar o controle simples com a estrutura	100%	01/02/2013	30/02/2013	15,0	20,0
Desevolver a interface com os sensores	Interfacear o acelerômetro com NIOS	100%	15/01/2013	30/01/2013	4,0	4,0
	Interfacear o giroscópio com NIOS	100%	15/01/2013	30/01/2013	4,0	2,0
	Interfacear o magnetómetro com NIOS	100%	15/01/2013	30/01/2013	4,0	2,0
	Integrar a coleta de dados	100%	16/02/2013	28/02/2013	12,0	30,0
Estudar e elaborar os movimentos	Estudar o movimento de hexápodes	100%	18/02/2013	28/02/2013	24,0	8,0
	Definir e implementar os movimentos	100%	01/03/2013	05/04/2013	48,0	50,0
	Testar os movimentos	100%	01/03/2013	13/04/2013	16,0	20,0
Desenvolver o Software da EB [driver]	Elaborar o sistema de controle	100%	01/03/2013	01/04/2013	8,0	8,0
	Implementar o sistema de controle	100%	24/03/2013	13/04/2013	24,0	12,0

Desenvolver o Firmware	Elaborar a maquina de estados	100%	18/02/2013	13/04/2013	8,0	8,0
	Implementar	100%	01/03/2013	13/04/2013	24,0	24,0
Desenvolver a biblioteca do cliente	Elaborar	100%	24/03/2013	13/04/2013	8,0	8,0
	Implementar	100%	01/04/2013	20/04/2013	12,0	14,0
Elaborar os diagramas de Software	Realizar a documentação do protocolo de baixo nível	100%	02/01/2013	15/04/2013	16,0	4,0
	Realizar a documentação do protocolo de alto nível	100%	02/01/2013	15/04/2013	16,0	4,0
	Criar o diagrama de blocos do driver	100%	02/01/2013	20/04/2013	12,0	4,0
	Diagrama geral de Software	100%	02/01/2013	20/04/2013	8,0	4,0
Documentar o Software	Documentar o Firmware	100%	07/04/2013	20/04/2013	10,0	15,0
	Documentar o Driver	100%	07/04/2013	20/04/2013	10,0	15,0
	Documentar a biblioteca de cliente	100%	07/04/2013	20/04/2013	10,0	15,0
	Documentar o sofware da interface	100%	14/04/2013	20/04/2013	10,0	12,0
Desenvolver o software da interface	Elaborar o sofware da interface	100%	15/03/2013	15/04/2013	12,0	8,0
	Implementar o sofware da interface	100%	15/03/2013	15/04/2013	24,0	20,0
Realizar testes de integração final	Integrar as PCIs com a FPGA	100%	08/03/2013	15/03/2013	42,0	40,0
	Integrar a cinematica inversa no hexápode	100%	15/03/2013	31/03/2013	30,0	80,0
	Integrar o driver com a FPGA	100%	15/03/2013	15/04/2013	35,0	50,0
	Integrar o driver com a interface	100%	01/04/2013	15/04/2013	20,0	24,0
	Testar a integração completa	100%	01/04/2013	30/04/2013	30,0	72,0
Total		100%	05/03/2012	30/04/2013	1372,0	1577,0

Apêndice F: Riscos identificados

Projeto: Robô hexápode
1ª etapa: **Identificação do Risco**
Denominação do risco: **Erro na Estimativa de Tempo, Custo ou Complexidade**
Número de identificação: **1**
Descrição do risco: Alguma das atividades pode ter seu prazo subestimado, devido à falta de experiência dos membros da equipe com o tipo de atividades envolvidas no projeto.
2ª etapa: **Avaliação do Risco**
Impacto: 5(alto) 4(médio/alto) 3(médio) 2(médio/baixo) 1(baixo) O projeto possui um deadline definida e a funcionalidade depende de praticamente todos os módulos estarem implementados. É esperado que esse risco torne o projeto inviável dentro do prazo caso ocorra este risco em grandes proporções.
Probabilidade: 5(alta) **4(média/alta)** 3(média) 2(média/baixa) 1 (baixa) Os elementos da equipe têm pequena experiência em trabalhar com robótica. Estimar tempo sob esta circunstância levará quase que certamente a algumas etapas excederem o tempo planejado.
3ª etapa: **Desenvolvimento da Resposta ao Risco**
Estratégias e Ações
Estratégias e Ações para eliminar ou reduzir este risco (minimizar impacto e/ou probabilidade) **Prevenir:** Controle próximo por parte da equipe da evolução das etapas; Planejamento dos tempos considerando as aptidões particulares dos elementos da equipe; **Mitigar:** Superestimar algumas tarefas para manter uma margem de segurança e planejar a data limite de algumas tarefas de modo a deixar uma sobre no cronograma. **Aceitar:** A somatória de atrasos não pode ultrapassar o tempo das folgas devido ao deadline. Se não houver folgas suficientes reduzir funcionalidades secundárias. Se isso também não for possível cortar funcionalidades principais em reunião com o orientador. **Transferir:** N/A

Projeto: *Robô hexápode*
1ª etapa: **Identificação do Risco**
Denominação do risco: **Demora ou Atraso na Aquisição de Componentes**
Número de identificação: **2**
Descrição do risco: A aquisição dos componentes é um item de cronograma com dependência externa. Atrasos, especialmente no caso dos componentes importados e nos críticos do projeto, representam um grande risco para o seu sucesso.
2ª etapa: **Avaliação do Risco**
Impacto: 5(alto) 4(médio/alto) 3(médio) 2(médio/baixo) 1(baixo) Os prazos de entrega dos componentes eletrônicos podem influenciar o bom andamento do projeto, forçando o início do trabalho em algumas etapas a atrasarem. O problema é ainda maior no caso de necessidade de reposição de componentes danificados. O impacto seria sentido provavelmente na qualidade ou na funcionalidade do produto. Nos cenários mais graves componentes críticos de reposição poderiam não chegar a tempo no deadline, o que produziria um produto não usável na data da entrega.
Probabilidade: 5(alta) 4(média/alta) **3(média)** 2(média/baixa) 1 (baixa) No caso de componentes importados não é raro haver pequenos atrasos. Podem ocorrer atrasos maiores devido aos órgãos fiscalizadores brasileiros como a receita e a alfândega. Além disso ainda existe a possibilidade de extravio da encomenda no pior cenário.
3ª etapa: **Desenvolvimento da Resposta ao Risco**
Estratégias e Ações
Estratégias e Ações para eliminar ou reduzir este risco (minimizar impacto e/ou probabilidade) **Prevenir:** Durante a elaboração da lista de componentes a serem usados no projeto deve-se levar em consideração os prazos de entrega. Componentes com menor prazo de entrega devem ser favorecidos. Deve ser prevista uma folga de cronograma na entrega dos componentes. Os componentes devem ser adquiridos o mais cedo possível. Para os componentes com maior risco determinar pelo menos 2 locais que o vendem, preferencialmente em Curitiba ou em cidades que tenham frete rápido para Curitiba, como é o caso de São Paulo. Deve-se também manter alternativas para componentes de aquisição mais rápida, mesmo que com características diferentes. **Mitigar:** No caso um componente estar atrasado a ponto de colocar em risco o projeto utilizar um componente alternativo, de aquisição mais rápida. Comprar componentes sobressalentes, no caso de componentes que não possam ser repostos rapidamente, caso ocorra um dano. **Aceitar:** Se não houver componente equivalente substituí-lo por outro com menor funcionalidade, desempenho ou qualidade; Se o produto puder ser concluído sem o componente, mesmo com funcionalidade reduzida, executá-lo, mediante reunião com o orientador do projeto. **Transferir:** N/A

Projeto: *Robô hexápode*
1ª etapa: **Identificação do Risco**
Denominação do risco: **Erro nas Escolhas Tecnológicas**
Número de identificação: **3**
Descrição do risco: A correta escolha de tecnologias é crítica para o sucesso deste projeto. Os itens escolhidos precisam atender em qualidade, desempenho e funcionalidades o projeto como um todo. Se as tecnologias utilizadas não suportarem o projeto não tem como ser um sucesso.
2ª etapa: **Avaliação do Risco**
Impacto: 5(alto) **4(médio/alto)** 3(médio) 2(médio/baixo) 1(baixo) Caso algum dos itens não possua as características necessárias o produto será afetado reduzindo qualidade ou funcionalidade. Se a escolha for muito errada, pode comprometer o projeto como um todo.
Probabilidade: 5(alta) 4(média/alta) 3(média) **2(média/baixa)** 1 (baixa) Dado que as funcionalidades do robô, pelo menos nos itens principais, serem mapeadas para funcionalidades providas pelos componentes e as características de desempenho poderem ser estimadas. A possibilidade disso ocorrer surge para os itens secundários do projeto ou para itens que, por inexperiência da equipe, sejam necessários e a equipe não tenha ciência disso.
3ª etapa: **Desenvolvimento da Resposta ao Risco**
Estratégias e Ações
Estratégias e Ações para eliminar ou reduzir este risco (minimizar impacto e/ou probabilidade) **Prevenir:** Levantar formalmente os itens de funcionalidade a serem atendidos e aprová-los com o cliente. Mapeá-los formalmente com recursos dos componentes a serem adquiridos. Solicitar a um professor experiente uma conferência. Escolher componentes que possam ser substituídos por outros mais poderosos, baratos e que possam ser adquiridos prontamente. Instruir a equipe a prestar atenção às características dos equipamentos de forma a notificar rapidamente aos demais no caso da possibilidade de alguma não-conformidade. **Mitigar:** Se houver um componente disponível prontamente de baixo custo e com as características necessárias deve-se adquiri-lo. Se for possível uma solução com baixa redução de qualidade, deve-se aplicá-la. **Aceitar:** Se a característica desejada reduzir alguma funcionalidade secundária do projeto, adequá-lo para operar adequadamente com esta redução. **Transferir:** N/A

Projeto: *Robô hexápode*
1ª etapa: **Identificação do Risco**
Denominação do risco: **Queima de Componente ou Componente Defeituoso**
Número de identificação: **4**
Descrição do risco: Durante o desenvolvimento do projeto existe o risco de dano a algum componente eletrônico, de forma que ele pode se tornar inutilizável no projeto.
2ª etapa: **Avaliação do Risco**
Impacto: 5(alto) **4(médio/alto)** 3(médio) 2(médio/baixo) 1(baixo) Alguns dos componentes do projeto, como a FPGA, desempenham papéis críticos no projeto. O não funcionamento destes componentes inviabiliza o projeto Outros componentes são usados em quantidade superior a uma unidade, como resistores ou opto acopladores, sendo que dano que afete apenas um deles possa apenas reduz a funcionalidade do produto ou seja facilmente substituído.
Probabilidade: 5(alta) 4(média/alta) 3(média) **2(média/baixa)** 1 (baixa) Dano em componente só ocorreria se houver falha grave no projeto da eletrônica ou se houver manuseio incorreto dos componentes. Sem estes cuidados especiais acreditamos que a probabilidade de ocorrência deste risco é média a baixa..
3ª etapa: **Desenvolvimento da Resposta ao Risco**
Estratégias e Ações
Estratégias e Ações para eliminar ou reduzir este risco (minimizar impacto e/ou probabilidade) **Prevenir:** Os elementos da equipe que tiverem mais proficiência com eletrônicos devem atuar diretamente na eletrônica do projeto. As especificações contidas nos *datasheets* sobre limites de operação dos componentes devem ser obedecidas. O manuseio, teste, montagem e operação da eletrônica devem ser feitos preferencialmente também pela equipe com mais proficiência nestes aspectos. Quando o orçamento permitir, adquirir componentes sobressalentes e especialmente os componentes críticos. Escolher, na medida do possível, componentes da fácil reposição. Na medida do possível, estabelecer formas de testar a operação individual dos componentes o mais cedo possível, antes de se iniciar a integração da eletrônica. **Mitigar:** Substituir por um sobressalente e adquirir novo componente, mas somente após se determinar por que o componente foi danificado e após garantir-se, dentro do possível, que o problema não vai recorrer; Adaptar projeto para operar com componentes similares com grande disponibilidade; Caso não haja, continuar trabalhando nas outras tarefas do projeto e convocar reunião de emergência. **Aceitar:** Se não houver componente equivalente substituí-lo por outro com menor funcionalidade, desempenho ou qualidade; Se o produto puder ser produzido sem o componente, mesmo com funcionalidade reduzida, executá-lo, mediante a provação com o cliente. **Transferir:** N/A

Projeto: *Robô hexápode*
1ª etapa: **Identificação do Risco**
Denominação do risco: **Inexperiência da Tecnologia Utilizada**
Número de identificação: **5**
Descrição do risco: Dificuldade de utilização da tecnologia por não entender o funcionamento e a utilização desta.
2ª etapa: **Avaliação do Risco**
Impacto: 5(alto) 4(médio/alto) **3(médio)** 2(médio/baixo) 1(baixo) Este risco pode provocar um atraso considerável, pois para algumas tecnologias é complexo encontrar referências esclarecedoras sobre pontos que só são entendidas com a prática com a utilização destas.
Probabilidade: 5(alta) 4(média/alta) 3(média) **2(média/baixa)** 1 (baixa) Dado que este é um trabalho de conclusão de curso, espera-se que haja familiaridade, nem que superficial, com a maior parte das tecnologias utilizadas.
3ª etapa: **Desenvolvimento da Resposta ao Risco**
Estratégias e Ações
Estratégias e Ações para eliminar ou reduzir este risco (minimizar impacto e/ou probabilidade) **Prevenir:** Inserir no plano de projeto datas para estudo e ensaio das tecnologias utilizadas no projeto para se adquirir a experiência necessária. **Mitigar:** Escolher preferencialmente tecnologias em que já se tem experiência ou que se saiba que há documentação clara e simples a respeito. Procurar alguém que tenha conhecimento sobre a tecnologia, como por exemplo os professores. **Aceitar:** N/A **Transferir:** N/A

Projeto: *Robô hexápode*
1ª etapa: **Identificação do Risco**
Denominação do risco: **Documentação Errônea das Tecnologias Utilizadas.**
Número de identificação: **6**
Descrição do risco: A documentação de qualquer das tecnologias utilizadas seja incompleta ou imprecisa.
2ª etapa: **Avaliação do Risco**
Impacto: 5(alto) **4(médio/alto)** 3(médio) 2(médio/baixo) 1(baixo) Os componentes escolhidos podem não fazer o que deveriam, ou fazer de forma errada.
Probabilidade: 5(alta) 4(média/alta) 3(média) 2(média/baixa) **1 (baixa)** Grande parte das tecnologias que se pensou em utilizar é muito utilizada, então é pouco provável que sejam mal documentadas ou não façam o que prometem.
3ª etapa: **Desenvolvimento da Resposta ao Risco**
Estratégias e Ações
Estratégias e Ações para eliminar ou reduzir este risco (minimizar impacto e/ou probabilidade) **Prevenir:** Busca-se o uso de tecnologias que já sejam bem estabelecidas, de forma que haja bastante informação sobre elas. **Mitigar:** Procurar saber se existe uma tecnologia alternativa que possa ser utilizado no lugar da tecnologia falha e estudá-la para caso o problema ocorra. **Aceitar:** Diminuir a funcionalidade do projeto devido a incapacidades técnicas ou trocar de tecnologia se possível. **Transferir:** N/A