

**UNIVERSIDADE DE SÃO PAULO  
ESCOLA DE ENGENHARIA DE SÃO CARLOS  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE  
COMPUTAÇÃO**

**RAFAEL SANTOS MOURA**

**DESENVOLVIMENTO DE UM SISTEMA  
DE ORIENTAÇÃO ESPACIAL INERCIAL**

São Carlos  
2013



**RAFAEL SANTOS MOURA**

# **DESENVOLVIMENTO DE UM SISTEMA DE ORIENTAÇÃO ESPACIAL INERCIAL**

Trabalho de Conclusão de Curso  
apresentado à Escola de Engenharia  
de São Carlos, da Universidade de  
São Paulo

Curso de Engenharia Elétrica com  
ênfase em Eletrônica

ORIENTADOR: Professora Mestre  
Luiza Maria Romeiro Codá

São Carlos  
2013

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTA TRABALHO,  
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS  
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

M929d Moura, Rafael Santos  
Desenvolvimento de um sistema de orientação  
espacial inercial / Rafael Santos Moura; orientadora  
Luisa Maria Romeiro Codá. São Carlos, 2013.

Monografia (Graduação em Engenharia Elétrica com  
ênfase em Eletrônica) -- Escola de Engenharia de São  
Carlos da Universidade de São Paulo, 2013.

1. Orientação espacial. 2. IMU. 3. MSP430. 4.  
Acelerômetro. 5. Giroscópio. I. Título.

# FOLHA DE APROVAÇÃO

Nome: Rafael Santos Moura

Título: "Desenvolvimento de um sistema de orientação espacial inercial"

Trabalho de Conclusão de Curso defendido e aprovado  
em 26 / 11 / 2013,

com NOTA 9,8 (Nove, oito), pela Comissão Julgadora:

*Profa. Assistente Luiza Maria Romeiro Codá - (Orientadora - SEL/EESC/USP)*

*Prof. Assistente Jerson Barbosa de Vargas - (SEL/EESC/USP)*

*Dr. Victor Antonio Cacciaccaro Lincoln - (Victor Vision)*

Coordenador da CoC-Engenharia Elétrica - EESC/USP:  
Prof. Associado Homero Schiabel



## **Agradecimentos**

Agradeço aos meus pais, por sempre me apoiarem e tornarem tudo isto possível.

À professora Luiza Maria Romeiro Codá, por todas as oportunidades oferecidas.

Aos meus amigos Telos Galante Mancera e Igor Guerrero por toda a ajuda neste projeto, e pela amizade de todos os dias.

A todos os professores, técnicos e funcionários da Escola de Engenharia de São Carlos que fizeram parte da minha formação, pelas lições e pelo auxílio.

À Universidade de São Paulo pela formação de qualidade.



## Resumo

MOURA, R. S. **Desenvolvimento de um sistema de orientação espacial inercial**. Trabalho de conclusão de curso de engenharia elétrica com ênfase em eletrônica da Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

Com o avanço das técnicas de construção de componentes eletromecânicos miniaturizados, vem se observando nos últimos anos a proliferação no mercado de dispositivos móveis com sensores de movimento. Este trabalho consiste no desenvolvimento de um sistema de orientação capaz de estimar continuamente sua inclinação e orientação espacial em relação à Terra. O sistema se baseia na leitura de sensores de aceleração e rotação através de um barramento SPI (*Serial Peripheral Interface*). As informações lidas dos sensores são processadas por um microcontrolador, responsável por calcular estimativas sobre a orientação atual do sistema. A orientação estimada é, então, transmitida a um computador, via protocolo UART (*Universal Asynchronous Receiver/Transmitter*), onde é representada visualmente, por meio de vetores, ou utilizada para movimentar motores de estabilização fixados ao sistema. Observou-se que o sistema, quando usando apenas as leituras do acelerômetro, se comporta muito ruidosamente, e é sensível a translações no espaço. Ao se adicionar as leituras do giroscópio, o sistema apresenta resposta muito menos ruidosa, e se torna quase insensível a translações. Este trabalho obteve estimativas corretas das inclinações do sistema em relação ao eixo vertical de referência da Terra, sendo implementado um sistema de estabilização vertical de câmera.

**Palavras-chave:** Orientação espacial; IMU; MSP430; Acelerômetro; Giroscópio.



## Abstract

MOURA, R. S. **Development of an inertial spatial orientation system**. Senior research project in electrical engineering with emphasis on electronics of the Escola de Engenharia de São Carlos (School of Engineering of São Carlos), Universidade de São Paulo (University of São Paulo), 2013.

Because of the improvement of miniature electromechanical devices fabrication techniques, mobile devices with motion sensing capabilities are getting increasingly present in the market. This work comprehends the development of an orientation system capable of continuously estimating its own tilt and spatial orientation with respect to Earth. This system is based on the reading of acceleration and rotation sensors through an SPI (Serial Peripheral Interface) bus. The retrieved sensor information is processed by a microcontroller, responsible for calculating an estimative regarding the current system orientation. The estimated orientation is sent to a computer, through UART (Universal Asynchronous Receiver/Transmitter) protocol, to be graphically represented, or used to cause movement in stabilizing motors fastened to the system. When considering solely the accelerometer's readings, the system behaves too noisily, and is sensitive to spacial translations. When also considering the gyroscope's readings, the noise is greatly reduced, and the system becomes nearly translation-insensitive. This work obtained correct estimations about the system's tilt with respect to the vertical reference axis of the Earth, successfully implementing a vertical camera stabilization system.

**Keywords:** Spatial orientation; IMU; MSP430; Accelerometer; Gyroscope.



## Lista de Figuras

|                                                                                                            |    |
|------------------------------------------------------------------------------------------------------------|----|
| Figura 1.1: Duas possíveis orientações da tela de um <i>smartphone</i> [1].                                | 19 |
| Figura 2.1: Representação gráfica de um sistema de coordenadas ortonormal tridimensional.                  | 21 |
| Figura 2.2: Sistema de coordenadas do corpo (em relação ao sistema de orientação).                         | 22 |
| Figura 2.3: Representação gráfica dos dois sistemas de referência.                                         | 22 |
| Figura 2.4: Sistemas de referência com mesma origem.                                                       | 23 |
| Figura 2.5: Eixos de rotação.                                                                              | 23 |
| Figura 2.6: Representação vetorial de uma rotação.                                                         | 24 |
| Figura 2.7: Exemplos de produto vetorial.                                                                  | 25 |
| Figura 2.8: Produto vetorial para encontrar rotações.                                                      | 25 |
| Figura 2.9: Veículo de transporte pessoal Segway [4].                                                      | 26 |
| Figura 2.10: Foto de um MEMS utilizando microscópio eletrônico, mostrando engrenagens [7].                 | 28 |
| Figura 2.11: Comparação entre uma engrenagem de um MEMS, um grão de pólen e células vermelhas humanas [8]. | 28 |
| Figura 2.12: Comparação entre um MEMS e um ácaro [9].                                                      | 29 |
| Figura 2.13: Princípio de funcionamento de um acelerômetro capacitivo [10].                                | 30 |
| Figura 2.14: Detalhe de construção de um acelerômetro capacitivo MEMS [11].                                | 30 |
| Figura 2.15: Giroscópio mecânico clássico [12].                                                            | 31 |
| Figura 2.16: Eixos de rotação [3].                                                                         | 32 |
| Figura 2.17: Detalhe de construção de um giroscópio MEMS com estrutura vibrante [13].                      | 32 |
| Figura 2.18: Ligação cruzada dos sinais RX e TX.                                                           | 33 |
| Figura 2.19: Formato geral de um pacote serial utilizando UART [16].                                       | 34 |
| Figura 2.20: Topologia típica de barramento SPI [17].                                                      | 36 |
| Figura 2.21: Diagrama de tempo para um barramento SPI [17].                                                | 37 |
| Figura 2.22: Mini servomotor HXT900 [19].                                                                  | 38 |
| Figura 2.23: Diagrama de tempo do sinal de controle do servomotor, para 3 posições [21].                   | 38 |
| Figura 3.1: Diagrama de blocos geral do sistema de orientação.                                             | 39 |
| Figura 3.2: Eixos dos sensores no sistema.                                                                 | 40 |
| Figura 3.3: <i>Kit</i> de desenvolvimento LaunchPad [28].                                                  | 41 |
| Figura 3.4: Pinagem do microcontrolador MSP430G2553.                                                       | 42 |
| Figura 3.5: Sistema de orientação sendo programado via cabo USB.                                           | 43 |
| Figura 3.6: Eixos de sensibilidade do acelerômetro [30].                                                   | 44 |
| Figura 3.7: Eixos de sensibilidade do giroscópio [31].                                                     | 45 |

|                                                                                                         |    |
|---------------------------------------------------------------------------------------------------------|----|
| Figura 3.8: Eixos de sensibilidade do magnetômetro [32].                                                | 45 |
| Figura 3.9: Encapsulamento LGA14 [33].                                                                  | 46 |
| Figura 3.10: Diagrama esquemático da placa <i>breakout</i> para o acelerômetro.                         | 47 |
| Figura 3.11: <i>Layout</i> da placa <i>breakout</i> para o acelerômetro.                                | 47 |
| Figura 3.12: Foto da <i>breakout</i> do acelerômetro.                                                   | 47 |
| Figura 3.13: Diagrama esquemático da placa <i>breakout</i> para o giroscópio.                           | 48 |
| Figura 3.14: <i>Layout</i> da placa <i>breakout</i> para o giroscópio.                                  | 48 |
| Figura 3.15: Foto da <i>breakout</i> do giroscópio.                                                     | 49 |
| Figura 3.16: Diagrama esquemático da placa <i>breakout</i> para o magnetômetro.                         | 49 |
| Figura 3.17: <i>Layout</i> da placa <i>breakout</i> para o magnetômetro.                                | 50 |
| Figura 3.18: Foto da <i>breakout</i> do magnetômetro.                                                   | 50 |
| Figura 3.19: Diagrama esquemático do <i>shield</i> V1.2.                                                | 51 |
| Figura 3.20: <i>Layout</i> do <i>shield</i> V1.2.                                                       | 51 |
| Figura 3.21: Fotos do <i>shield</i> V1.2.                                                               | 52 |
| Figura 3.22: Sistema de orientação com as <i>breakouts</i> inseridas.                                   | 52 |
| Figura 3.23: Diagrama esquemático do <i>shield</i> V1.3.                                                | 53 |
| Figura 3.24: <i>Layout</i> do <i>shield</i> V1.3.                                                       | 53 |
| Figura 3.25: Fotos do <i>shield</i> V1.3.                                                               | 54 |
| Figura 3.26: <i>Breakout</i> da tela de cristal líquido.                                                | 55 |
| Figura 3.27: Módulo da tela de cristal líquido acoplado ao sistema de orientação.                       | 55 |
| Figura 3.28: Cabo de comunicação.                                                                       | 56 |
| Figura 3.29: Conectores MODU.                                                                           | 56 |
| Figura 3.30: Sistema de orientação conectado ao computador via cabo de comunicação e conversor TTL/USB. | 57 |
| Figura 3.31: Módulos componentes do <i>software</i> .                                                   | 58 |
| Figura 3.32: Fluxograma do <i>software</i> .                                                            | 60 |
| Figura 3.33: Um dos eixos do acelerômetro sendo lido com o protocolo SPI.                               | 61 |
| Figura 3.34: Sinais antes e depois da aplicação do filtro de média móvel.                               | 62 |
| Figura 3.35: Sinal antes e depois da supressão de ruído próximo ao nível zero.                          | 63 |
| Figura 3.36: Diagrama de blocos simplificado do algoritmo de estimação.                                 | 64 |
| Figura 3.37: Diagrama de blocos simplificado, magnetômetro incluso.                                     | 65 |
| Figura 3.38: Matriz de rotação Mxy.                                                                     | 66 |
| Figura 3.39: Propriedades trigonométricas do vetor Vert.                                                | 67 |
| Figura 3.40: <i>Software</i> RealTerm, durante captura de dados enviados pelo microcontrolador.         | 68 |

|                                                                                                                                    |    |
|------------------------------------------------------------------------------------------------------------------------------------|----|
| Figura 3.41: Módulo conversor USB. ....                                                                                            | 70 |
| Figura 4.1: Estimação da orientação, durante movimento translacional circular, sem (superior) e com (inferior) o giroscópio.. .... | 71 |
| Figura 4.2: Movimento realizado para teste de supressão da aceleração linear. ....                                                 | 72 |
| Figura 4.3: Posição vertical positiva.....                                                                                         | 73 |
| Figura 4.4: Inclinação com <i>pitch</i> igual a 45 graus. ....                                                                     | 73 |
| Figura 4.5: Inclinação com <i>pitch</i> e <i>roll</i> iguais a -45 graus.....                                                      | 74 |
| Figura 4.6: Inclinação com <i>roll</i> igual a 45 graus. ....                                                                      | 74 |
| Figura 4.7: Sentidos de giro do vetor estimador. ....                                                                              | 75 |
| Figura 4.8: Sistema com servomotores em repouso. ....                                                                              | 76 |
| Figura 4.9: Resposta dos servomotores para uma inclinação arbitrária. ....                                                         | 76 |
| Figura 4.10: Canal 2: Linha de transmissão serial, cursores marcando o período de amostragem. ....                                 | 79 |
| Figura 4.11: Canal 2: Linha de transmissão serial, cursores marcando o tempo gasto pela comunicação. ....                          | 79 |



## **Lista de Siglas**

|       |                                                    |
|-------|----------------------------------------------------|
| ASCII | American Standard Code for Information Interchange |
| CI    | Circuito integrado                                 |
| GPS   | Global Positioning System                          |
| I2C   | Inter-Integrated Circuit                           |
| IMU   | Inertial Measurement Unit                          |
| INS   | Inertial Navigation System                         |
| MEMS  | Microelectromechanical System                      |
| PCI   | Placa de circuito impresso                         |
| PWM   | Pulse Width Modulation                             |
| RAM   | Random Access Memory                               |
| RS    | Recommended Standard                               |
| SMD   | Surface Mount Device                               |
| SPI   | Serial Peripheral Interface                        |
| TTL   | Transistor-Transistor Logic                        |
| UART  | Universal Asynchronous Receiver/Transmitter        |
| USB   | Universal Serial Bus                               |
| USP   | Universidade de São Paulo                          |
| UV    | Ultravioleta                                       |



# Sumário

|       |                                                                 |    |
|-------|-----------------------------------------------------------------|----|
| 1     | Introdução.....                                                 | 19 |
| 1.1   | Motivação .....                                                 | 19 |
| 1.2   | Objetivos do Trabalho.....                                      | 20 |
| 1.3   | Organização da Monografia.....                                  | 20 |
| 2     | Revisão Teórica .....                                           | 21 |
| 2.1   | Sistemas de coordenadas .....                                   | 21 |
| 2.2   | Produto vetorial .....                                          | 24 |
| 2.3   | Unidades de Medição Inercial.....                               | 26 |
| 2.4   | Sistemas Microeletromecânicos .....                             | 27 |
| 2.4.1 | Acelerômetro .....                                              | 29 |
| 2.4.2 | Giroscópio .....                                                | 30 |
| 2.4.3 | Magnetômetro .....                                              | 33 |
| 2.5   | <i>Universal Asynchronous Receiver/Transmitter (UART)</i> ..... | 33 |
| 2.6   | <i>Serial Peripheral Interface (SPI)</i> .....                  | 34 |
| 2.6.1 | Introdução.....                                                 | 34 |
| 2.6.2 | Operação.....                                                   | 36 |
| 2.7   | Servomotor .....                                                | 37 |
| 3     | Desenvolvimento do Projeto .....                                | 39 |
| 3.1   | Visão geral.....                                                | 39 |
| 3.2   | Escolha dos componentes.....                                    | 41 |
| 3.2.1 | Microcontrolador .....                                          | 41 |
| 3.2.2 | Sensores .....                                                  | 43 |
| 3.3   | Placas <i>breakout</i> para os sensores .....                   | 46 |
| 3.4   | Placa acopladora ( <i>Shield</i> ) .....                        | 50 |
| 3.5   | Mesa de testes rotativa .....                                   | 57 |
| 3.6   | <i>Software</i> .....                                           | 57 |
| 3.6.1 | Visão geral.....                                                | 58 |

|       |                                                                         |     |
|-------|-------------------------------------------------------------------------|-----|
| 3.6.2 | Aquisição de dados e processamento preliminar .....                     | 61  |
| 3.6.3 | Estimação da orientação .....                                           | 64  |
| 3.6.4 | Comunicação externa .....                                               | 67  |
| 4     | Resultados e Discussões.....                                            | 71  |
| 4.1   | Resultados do funcionamento .....                                       | 71  |
| 4.2   | Problemas .....                                                         | 77  |
| 5     | Conclusões .....                                                        | 81  |
| 5.1   | Sugestões para trabalhos futuros .....                                  | 81  |
| 6     | Bibliografia .....                                                      | 83  |
|       | APÊNDICE A - Método de fabricação das placas de circuito impresso ..... | 86  |
|       | APÊNDICE B – Códigos fonte em C.....                                    | 96  |
|       | APÊNDICE C – Código fonte em MATLAB.....                                | 132 |
|       | APÊNDICE D – Desenhos da mesa de testes .....                           | 134 |

# 1 Introdução

## 1.1 Motivação

Com a crescente presença no mercado de dispositivos sensíveis a movimento, como controles de *video game*, *smartphones* e dispositivos apontadores, devida ao barateamento e miniaturização dos sensores, o desenvolvimento de aplicações que envolvem movimento está cada vez mais popular. Para cada aplicação, diversos tipos de movimento podem ser considerados. Por exemplo, um *smartphone* pode utilizar apenas a direção e sentido da gravidade para orientar a tela apropriadamente no modo paisagem ou retrato, como pode ser visto na Figura 1.1, enquanto que uma aeronave utiliza sistemas sofisticados que unem informações provenientes de giroscópios, magnetômetros, GPS, acelerômetros, barômetros, entre outros sensores, para estimar com grande acurácia suas coordenadas geográficas, velocidade, inclinação, e curso.



Figura 1.1: Duas possíveis orientações da tela de um *smartphone* [1].

O estudo feito neste trabalho pretende servir de pilar para posteriores aperfeiçoamentos, tendo como meta final criar um sistema portátil, alimentado por bateria, e com custo reduzido, que consiga estimar com exatidão, durante longos períodos de tempo (na faixa de dias), a posição do usuário no espaço, dada uma posição inicial. Por exemplo, tal sistema poderia ser utilizado para mostrar constantemente ao usuário uma seta indicando onde seu veículo está estacionado. No entanto, tal sistema deve se basear exclusivamente em **navegação inercial**, ou seja, nenhuma comunicação entre o carro e o dispositivo é permitida, nem qualquer correção de posição baseada

em GPS, WiFi ou telefonia celular, eliminando, assim, a necessidade de céu aberto ou *internet* no ambiente.

## 1.2 Objetivos do Trabalho

O objetivo deste trabalho foi projetar e construir um sistema de orientação espacial capaz de estimar, continuamente, sua **atitude**, isto é, sua orientação (rotação tridimensional) em relação a um referencial inercial (no caso, a Terra). Para tanto, são utilizados três sensores tri-axiais (acelerômetro, giroscópio e magnetômetro), isto é, sensores com três eixos de leitura, e um microcontrolador MSP430.

O projeto e desenvolvimento deste trabalho proporcionaram, além da aplicação dos conhecimentos teóricos e técnicos adquiridos durante a graduação, a expansão destes conhecimentos, principalmente nas áreas de instrumentação, protocolos de comunicação, engenharia de *software* e design de placas de circuito impresso.

O trabalho pode ser dividido em objetivos parciais: integração dos sensores (acelerômetro, giroscópio e magnetômetro) ao sistema, estimação da inclinação vertical, uso desta estimação para implementar um sistema de estabilização de plataformas, exibição gráfica de vetores em um computador para representar as estimativas de orientação, fusão do magnetômetro e do acelerômetro para implementar uma bússola digital, e estimação da rotação do sistema ao redor de todos os eixos utilizando a compensação proporcionada pela bússola.

## 1.3 Organização da Monografia

Este trabalho está dividido em cinco capítulos, estruturados como a seguir:

1. Introdução: Apresenta a motivação, objetivos e organização do projeto.
2. Revisão Teórica: Breve revisão teórica com a finalidade de contextualizar o leitor sobre os principais assuntos tratados no projeto.
3. Desenvolvimento do Projeto: Descreve mais profundamente o projeto, detalhando aspectos técnicos e teóricos de seu desenvolvimento. Inclui o método de escolha dos materiais utilizados, e descrições sobre o *software* e *hardware* desenvolvidos.

4. Resultados e Discussões: Apresenta os resultados obtidos. Inclui análise dos problemas encontrados e discussão sobre possíveis soluções.
5. Conclusões: Apresenta as conclusões obtidas neste trabalho, bem como sugestões para aprimoramentos futuros.

## 2 Revisão Teórica

### 2.1 Sistemas de coordenadas

Na análise de orientação e posição de objetos, é indispensável definir sistemas de coordenadas de referência. Um sistema de coordenadas tridimensional é um conjunto de três valores que determinam unicamente qualquer posição no espaço [2] [3].

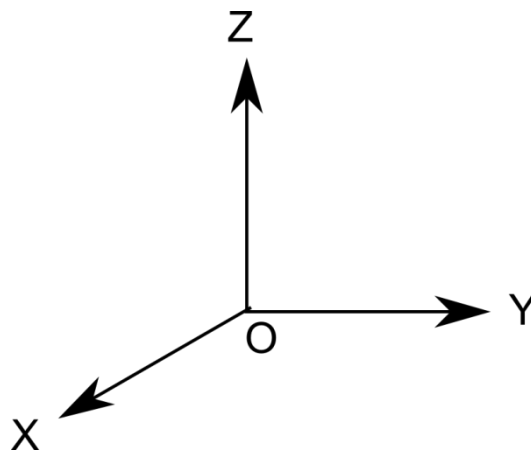


Figura 2.1: Representação gráfica de um sistema de coordenadas ortonormal tridimensional.

Um sistema de coordenadas pode ser representado visualmente como na Figura 2.1. O ponto **O**, de coordenadas  $(x,y,z) = (0,0,0)$ , é o centro do sistema, definido arbitrariamente, e as setas indicam os eixos do sistema. Num sistema de coordenadas ortonormal, os eixos X, Y e Z são perpendiculares uns aos outros, e têm tamanho unitário. Os sistemas de coordenadas considerados neste trabalho são todos ortonormais.

Apesar da Terra ter a forma de um geoide, e sua superfície não ser lisa, para efeitos de simplificação, será considerado como sistema de referência um sistema de coordenadas com centro em uma posição arbitrária próxima ao sistema de orientação projetado e discutido neste trabalho, e orientado de tal forma que seu eixo Z é ortogonal ao solo. Seja este sistema de referência chamado **S<sub>T</sub> (Sistema Terra)**, e representado como na Figura 2.1. Seja, então, **S<sub>C</sub> (Sistema Corpo)** um sistema de coordenadas com centro **O<sub>C</sub>** no centro de massa do sistema de orientação, e orientado de tal forma que seu eixo Z seja ortogonal ao plano formado pela superfície do sistema de orientação, como na Figura 2.2.

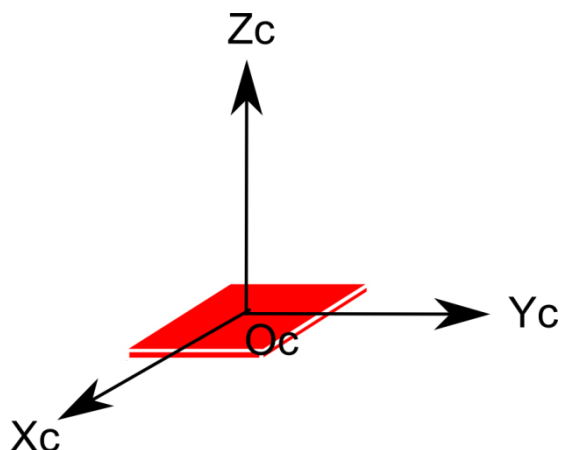


Figura 2.2: Sistema de coordenadas do corpo (em relação ao sistema de orientação).

O sistema  $S_T$  é considerado estático, enquanto que o sistema  $S_C$  é livre para se movimentar em relação a  $S_T$ , tanto por meio de translações como por meio de rotações. O ponto  $O_C$  tem coordenadas  $(X_C, Y_C, Z_C)$  em relação a  $S_T$ . Uma representação gráfica dos dois sistemas de referência pode ser vista na Figura 2.3.

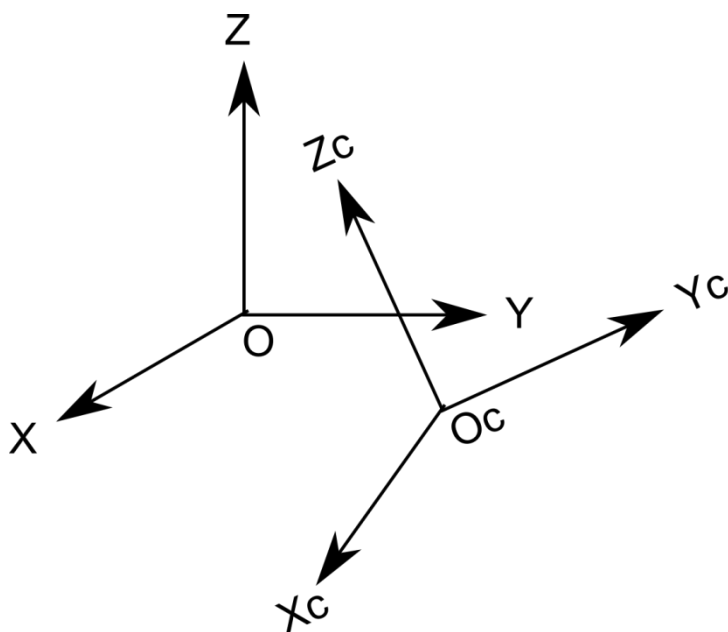


Figura 2.3: Representação gráfica dos dois sistemas de referência.

Como o sistema de orientação desenvolvido neste trabalho tem como objetivo estimar apenas a **orientação** do sistema de coordenadas do corpo em relação ao sistema de coordenadas da Terra, desconsiderando sua **posição** relativa, as translações de  $S_C$  podem ser ignoradas, fazendo com que ambos os sistemas tenham a mesma origem ( $O = O_C$ ). Esta simplificação é representada visualmente na Figura 2.4.

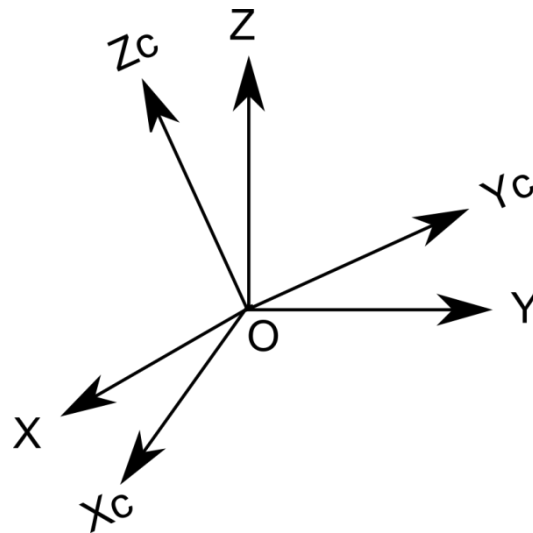


Figura 2.4: Sistemas de referência com mesma origem.

Além da definição dos eixos espaciais do sistema de coordenadas, é necessária a definição dos eixos de rotação do sistema. A rotação em torno do eixo Y será chamada *pitch*, com sentido positivo de Z para X. A rotação em torno do eixo X será chamada *roll*, com sentido positivo de Y para Z. A rotação em torno do eixo Z será chamada *yaw*, com sentido positivo de X para Y. A Figura 2.5 exemplifica os eixos de rotação, com seus respectivos sentidos positivos representados pelas setas [4].

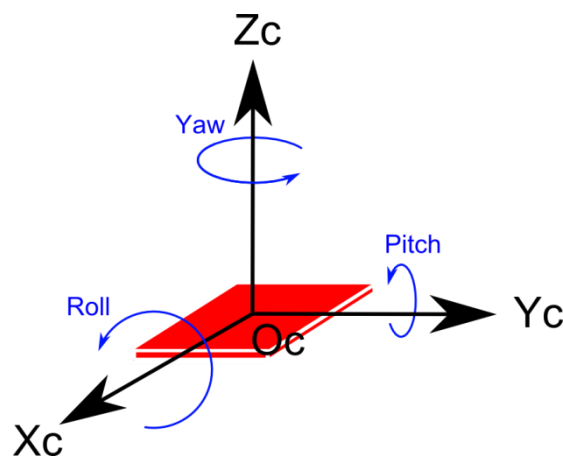


Figura 2.5: Eixos de rotação.

As rotações podem ser aplicadas aos vetores utilizando-se matrizes de rotação em cada um dos eixos, denotadas  $M_x$ ,  $M_y$  e  $M_z$ . Multiplicando-se as matrizes, obtém-se uma matriz completa de rotação, denotada, por exemplo,  $M = M_x.M_y.M_z$ . É importante notar que a orientação final do sistema de coordenadas depende da sequência de rotações aplicadas (visto que o produto de matrizes não é uma operação comutativa). No entanto, se os ângulos de rotação forem

infinitesimais, a ordem das matrizes, isto é, das rotações, não importa. Considerando-se que o período entre uma amostra e a seguinte é suficientemente pequeno, admite-se que as variações de ângulos são pequenas, permitindo que o modelo das rotações seja simplificado, supondo que a ordem das matrizes é irrelevante.

## 2.2 Produto vetorial

A magnitude, direção e sentido da rotação ao redor de um eixo pode ser representada graficamente através de um vetor paralelo ao dado eixo, com magnitude proporcional à magnitude da rotação, e com sentido adequado. Na Figura 2.6, por exemplo, uma rotação ao redor do eixo X, no sentido positivo, é representada pelo vetor azul, paralelo ao eixo X, com magnitude proporcional, e sentido positivo (valor positivo) [5].

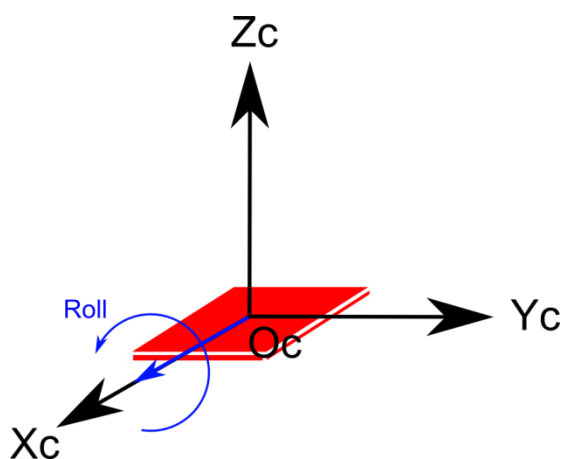


Figura 2.6: Representação vetorial de uma rotação.

O produto vetorial é uma operação entre dois vetores que resulta em um terceiro vetor, simultaneamente perpendicular aos outros dois, e cuja magnitude é igual à área do paralelogramo formado pelos dois primeiros vetores. O símbolo do operador “produto vetorial” é “ $\times$ ”. A ordem dos operandos é importante no resultado do produto vetorial. Por exemplo, considerando-se dois vetores, sendo  $V_1$  um vetor positivo paralelo ao eixo X, e  $V_2$  um vetor positivo paralelo ao eixo Y, o vetor  $V_3 = V_1 \times V_2$  será positivo e paralelo ao eixo Z; o vetor  $V_4 = V_2 \times V_1$ , por outro lado, será paralelo ao eixo Z, mas negativo (apontando para o lado oposto a  $V_3$ ), como pode ser visto na Figura 2.7.

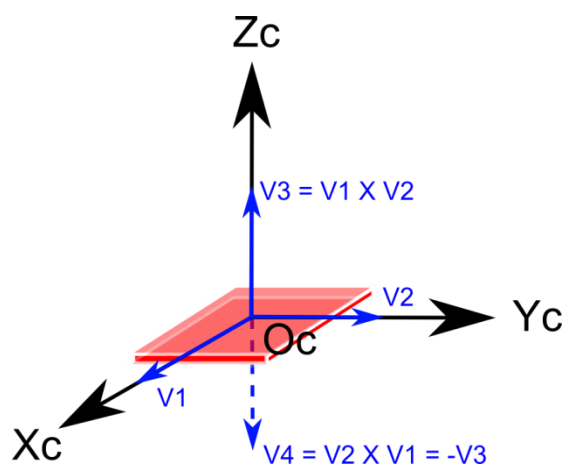


Figura 2.7: Exemplos de produto vetorial.

Esta operação pode ser utilizada para se obter um vetor que indica a direção e sentido que uma rotação deve ter para aproximar dois vetores. A Figura 2.8 ilustra esta afirmação: O produto vetorial  $V1 \times V2$  gera um vetor  $V3$  que é perpendicular a  $V1$  e  $V2$ , e que representa um eixo de rotação. Rotacionando-se  $V1$  positivamente ao redor de  $V3$ ,  $V1$  é aproximado de  $V2$ . Considerando-se o vetor oposto a  $V1$  (denotado  $-V1$ ), o resultado do produto vetorial é, também, oposto ao resultado anterior, indicando que a rotação deve ser no sentido contrário (negativo).

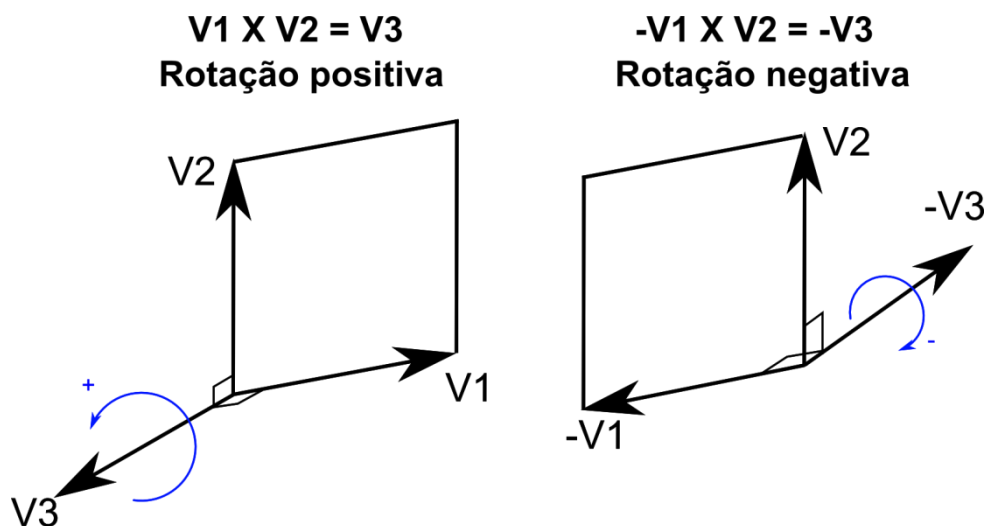


Figura 2.8: Produto vetorial para encontrar rotações.

Se o produto vetorial resultar em um vetor nulo, então os vetores são paralelos, com mesmo sentido ou não. Apesar do sentido do vetor resultante do produto vetorial indicar o sentido da rotação, a magnitude do vetor não indica a magnitude da rotação necessária para unir os vetores.

## 2.3 Unidades de Medição Inercial

Unidades de Medição Inercial (do inglês, *Inertial Measurement Units* – **IMUs**) são dispositivos eletrônicos que empregam acelerômetros, giroscópios e, eventualmente, magnetômetros para medir velocidade, orientação e posição de um veículo (ou qualquer objeto ao qual o sistema esteja fixado) [3]. Uma IMU pode ser empregada em uma ampla gama de aplicações, desde um simples dispositivo para equilíbrio automático, até complexos sistemas de navegação estimada (*dead reckoning*) [7].

O veículo de transporte pessoal Segway (Figura 2.9), que é basicamente um pêndulo invertido, usa uma IMU para se manter sempre na posição vertical. Seu sistema de controle interpreta a inclinação do passageiro como um comando de velocidade: se a inclinação é para frente, o veículo adquire velocidade para frente, e vice-versa.



Figura 2.9: Veículo de transporte pessoal Segway [4].

Por sua vez, **sistemas de navegação inercial** (*Inertial Navigation System* – INS: sistemas de estimativa de posição baseada na posição anterior, velocidade e tempo decorrido deste a posição anterior) também utilizam IMUs, porém requerem outros dispositivos para auxiliar as estimações e corrigir os desvios inerentes à IMU, como GPS, barômetro e sensores de velocidade externos. O nome IMU deriva do fato de que estas unidades baseiam suas medições em características inerciais (apesar do magnetômetro usualmente fazer parte de uma IMU, esse não faz medições inerciais), pois o acelerômetro mede a aceleração do objeto, e o giroscópio a taxa de rotação, ambas grandezas derivadas da massa (inércia), como será visto na seção 2.4. A necessidade de correção por equipamentos externos à IMU se origina da acumulação de erros inerente a este sistema. Os erros, por sua vez, são oriundos da impossibilidade de se adquirir os dados dos sensores continuamente, isto é, de se fazer a leitura a intervalos infinitamente pequenos de tempo.

Os valores perdidos entre uma amostragem dos sensores e o próximo são a fonte dos erros acumulativos de uma IMU. Se fosse possível criar um sistema com taxa de amostragem infinita (e velocidade de processamento também infinita), IMUs poderiam ser utilizadas sem o auxílio de sensores adicionais, sem desvios.

Numa IMU, o papel do acelerômetro é prover a característica do movimento a longo prazo. Acelerômetros apresentam alta taxa de ruído, e, além disso, não são capazes de medir exclusivamente as componentes lineares da aceleração sendo aplicada ao sistema, medindo também as componentes angulares da aceleração. E.g., ao se rotacionar um acelerômetro ao redor de seu centro de massa, ele registrará aceleração, apesar de não estar sendo submetido a aceleração linear. No entanto, os valores de aceleração não sofrem de erros acumulativos.

O papel do giroscópio é prover a característica do movimento a curto prazo. Giroscópios são muito menos suscetíveis a ruídos que acelerômetros, e apresentam resposta mais rápida. No entanto, como as leituras de um giroscópio fornecem a velocidade angular, é necessário se fazer a integração destas leituras para se obter o ângulo de rotação do sistema, o que introduz erros acumulativos de integração. Assim, a longo prazo, a estimativa baseada puramente em giroscópios deriva do valor real.

Juntando-se estes dois sensores em uma IMU, os problemas de cada um são compensados pelo outro. O giroscópio reduz a sensibilidade do acelerômetro a acelerações não lineares, enquanto que o acelerômetro corrige a deriva do giroscópio, usando o vetor da força gravitacional para determinar a direção vertical correta, diminuindo o erro da posição angular do sistema.

A inserção de magnetômetros corrige a deriva da posição angular ao redor do eixo da força gravitacional. O acelerômetro não é capaz de corrigir a deriva em rotações ao redor deste eixo, pois o vetor aceleração observado por um objeto é invariante a rotações sobre o eixo de gravidade. Por exemplo, um veículo trafegando em uma esfera perfeita sempre observará o vetor da aceleração gravitacional apontando para o centro da esfera, paralelamente ao seu próprio eixo vertical. O magnetômetro inclui a direção do norte magnético da Terra como elemento de correção no sistema.

## 2.4 Sistemas Microeletromecânicos

Sistemas Microeletromecânicos (*Microelectromechanical Systems* – MEMS) são dispositivos muito pequenos, constituídos de componentes com tamanhos entre 1 e 100 micrometros, medindo no total normalmente de 20 micrometros a 1 milímetro. São compostos usualmente por uma

unidade de processamento e alguns componentes que interagem com o meio, agindo como microsensores. Por causa de seu tamanho reduzido, os métodos de construção de dispositivos da física clássica não são adequados. Por exemplo, por causa da elevada razão área/volume, efeitos de superfície como eletrostática e molhabilidade [5] (habilidade de um líquido em manter contato com uma superfície sólida) se tornam preponderantes quando comparados com os efeitos de volume, como a inércia [6].

A habilidade de construir dispositivos tão pequenos possibilitou a miniaturização dos sensores, que por sua vez possibilitou a integração destes em dispositivos móveis. As figuras 2.10, 2.11 e 2.12, mostram fotos feitas por microscópio eletrônico, ilustrando o quão diminutos são os MEMS.

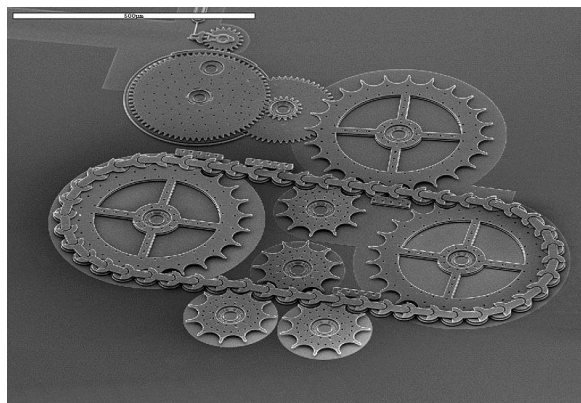


**Figura 2.10:** Foto de um MEMS utilizando microscópio eletrônico, mostrando engrenagens [7].

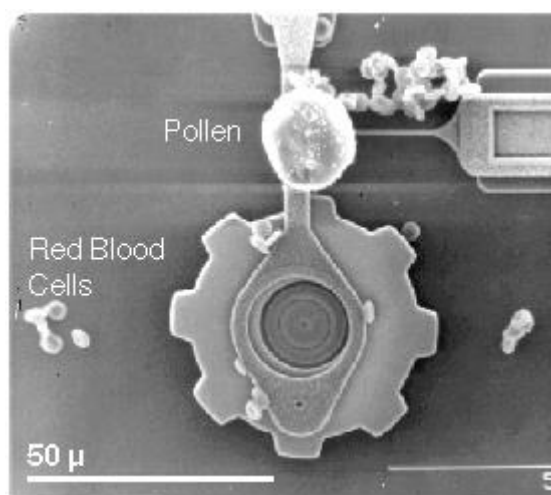


**Figura 2.11:** Comparação entre uma engrenagem de um MEMS, um grão de pólen e células vermelhas humanas [8].

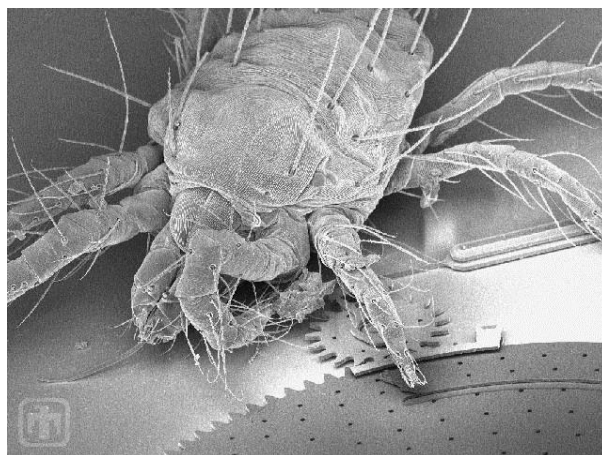


Figura 2.12: Comparação entre um MEMS e um ácaro [9].

Nas subseções 2.4.1, 2.4.2 e 2.4.3 são brevemente descritos os sensores MEMS utilizados neste trabalho. Um sensor é um dispositivo transdutor que converte uma grandeza física (sinal de entrada) qualquer em um sinal elétrico (sinal de saída) proporcional a esta grandeza física. Os sensores utilizados neste trabalho dispõem de conversores analógico-digital integrados, de forma que seus sinais de saída são digitais, sendo assim prontamente lidos pelo microcontrolador. O protocolo SPI (*Serial Peripheral Interface* – Interface Serial Periférica) foi escolhido para fazer a comunicação entre o microcontrolador e os sensores.

#### 2.4.1 Acelerômetro

Um acelerômetro é um sensor cujo sinal de saída indica a aceleração à qual está sujeito, ao longo de apenas uma direção. Há diversas formas de se construir um acelerômetro, sendo uma das mais comuns a que utiliza a variação de capacitância. O princípio de funcionamento deste tipo de acelerômetro se baseia no movimento que uma massa de prova, suspensa por molas, faz entre duas placas fixas ao substrato do sensor, de tal forma que as capacitâncias entre a massa de prova e cada uma das placas variam de acordo com as respectivas distâncias. Observa-se na Figura 2.13 que, quando há aceleração para a esquerda, as placas fixas são movimentadas para a esquerda. No entanto, como a massa de prova está suspensa por molas, devido à sua inércia, ela não é imediatamente movimentada. Assim, o efeito, do ponto de vista das placas móveis, é que a massa se aproxima da placa da direita, aumentando a capacitância entre elas ( $C_2$ ), e diminuindo a

capacitância entre a massa e a placa da esquerda ( $C_1$ ). A diferença entre estas duas capacitâncias proporciona um  $\Delta C$  que é proporcional à aceleração aplicada [7].

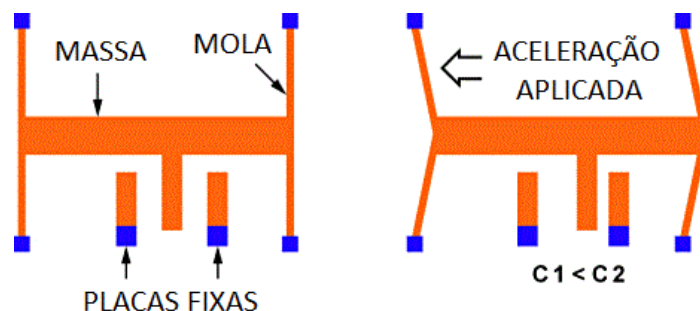


Figura 2.13: Princípio de funcionamento de um acelerômetro capacitivo [10].

A Figura 2.14 mostra em detalhe as placas fixas (também chamadas aletas ou hastes, são presas em pontos chamados âncoras, *anchors* na imagem) e a massa de prova móvel (*movable proof mass*) com hastes protuberantes (*sense fingers*), presa pela mola (*tether*). Uma vez que a variação de capacitância é diminuta, são usados vários conjuntos de placas fixas e móveis, a fim de se obter um sinal mensurável.

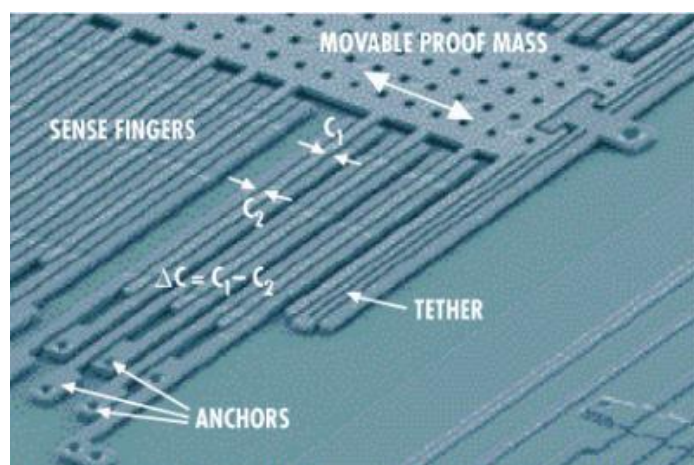


Figura 2.14: Detalhe de construção de um acelerômetro capacitivo MEMS [11].

## 2.4.2 Giroscópio

Giroscópios são mecanismos que se baseiam no princípio da conservação do momento angular, e são capazes de medir a orientação de um objeto. A versão clássica de um giroscópio (Figura 2.15) consiste em um disco (rotor) montado em uma estrutura que provê 3 graus de

liberdade rotacionais, isto é, o disco é livre para girar ao redor de qualquer um de seus eixos de rotação [7].

A Figura 2.16 ilustra estes três eixos, que no âmbito aeroespacial são comumente referidos por seus nomes em inglês, *pitch*, *roll* e *yaw* (em português, respectivamente, empinamento ( $\vartheta$ ), balanceio ( $\varphi$ ) e cabeceio ( $\psi$ )).

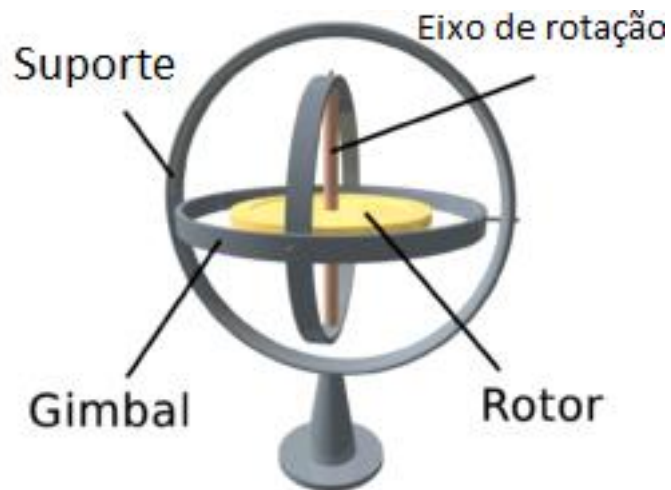


Figura 2.15: Giroscópio mecânico clássico [12].

Ao ser girado, o disco tende a continuar em rotação na mesma orientação, devido à conservação de momento angular. Assim, como o mecanismo é feito de modo a minimizar o atrito nas juntas dos arcos, quando a estrutura é girada, o disco continua na mesma orientação, girando quase independentemente do resto da estrutura. Deste modo, é possível medir o ângulo entre o eixo de rotação do disco e os arcos mais externos e determinar quanto a estrutura girou em relação à sua orientação original. Os arcos de suspensão são chamados gimbais. Os eixos de rotação de uma aeronave são ilustrados na Figura 2.16.



Figura 2.16: Eixos de rotação [3].

Os giroscópios MEMS, por sua vez, podem ser construídos de diversas maneiras. Uma das mais simples e baratas é a que utiliza microestruturas vibrantes. Ao invés de um disco girante, este tipo de giroscópio usa uma massa de prova que oscila em apenas uma direção. De forma semelhante ao acelerômetro discutido anteriormente, a saída do sensor é gerada pela aproximação e distanciamento entre as aletas que se projetam do corpo de prova e as aletas fixas ao substrato do circuito integrado. Quando o sensor é rotacionado, a massa de prova tende a continuar vibrando na mesma direção em que estava antes, de modo que surge uma deflexão entre as hastes proporcional à velocidade angular. A Figura 2.17 ilustra uma configuração típica de um giroscópio MEMS com estrutura vibrante. Na massa de prova, o eixo de oscilação é o *Driven Mode* e o eixo de sensibilidade é o *Sense Mode*.

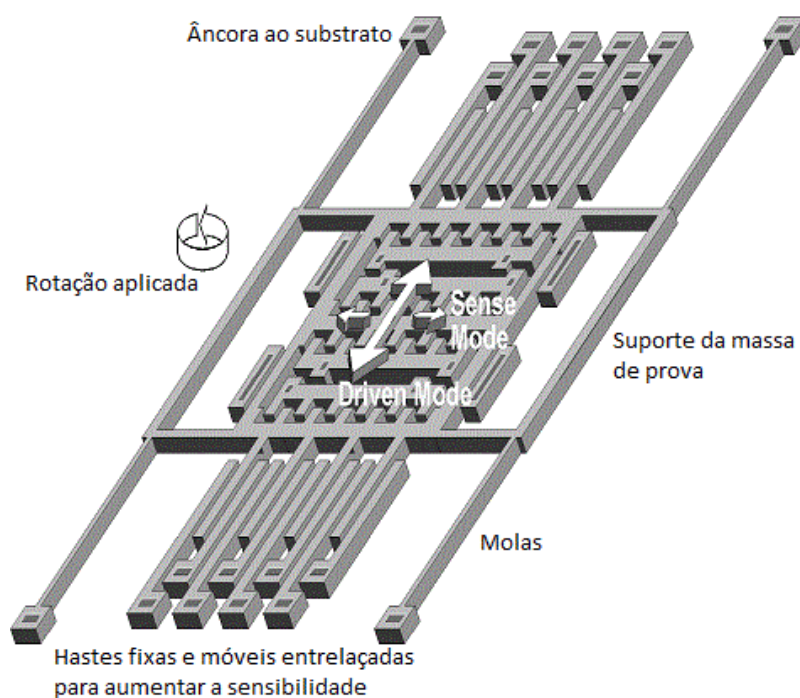


Figura 2.17: Detalhe de construção de um giroscópio MEMS com estrutura vibrante [13].

### 2.4.3 Magnetômetro

Magnetômetros são dispositivos empregados na medição da intensidade e, possivelmente, da direção e sentido de campos magnéticos. Magnetômetros são amplamente utilizados para medir o campo magnético da Terra. Há dois tipos de magnetômetros: escalares e vetoriais; enquanto que os magnetômetros escalares medem apenas a magnitude do campo magnético, os vetoriais são capazes de medir a magnitude e sentido do campo magnético na direção em que estão alinhados. Utilizando dois magnetômetros vetoriais dispostos paralelamente à Terra mas não paralelos entre si é possível calcular a direção do norte magnético da Terra, fazendo a soma vetorial das leituras [7].

Magnetômetros podem ter diversos princípios de funcionamento, como, por exemplo, os magnetômetros de Efeito Hall, de bobina rotativa, e de precessão protônica.

## 2.5 Universal Asynchronous Receiver/Transmitter (UART)

Um UART (Receptor/Transmissor Assíncrono Universal) é um dispositivo, normalmente incluso como um periférico embutido em microcontroladores, que transforma dados paralelos em dados seriais, e envia estes dados serializados por uma linha de comunicação para outro UART que transformará os dados seriais de volta em dados paralelos [14] [15] [20].

A comunicação por UART pode ser realizada em três modos:

- *Simplex* (comunicação em apenas um sentido);
- *Half-duplex* (comunicação de mão dupla alternada);
- *Full-duplex* (comunicação de mão dupla simultânea).

Estes modos de comunicação são possíveis pois o barramento de comunicação é composto por duas linhas de mão única, uma indo de um dispositivo para o outro, e a outra vindo deste último para o primeiro. O barramento deve ser ligado em modo cruzado (*cross-over*), como pode ser visto na Figura 2.18, onde o pino RX (recepção) de um módulo UART é conectado ao pino TX (transmissão) do outro, e vice-versa.

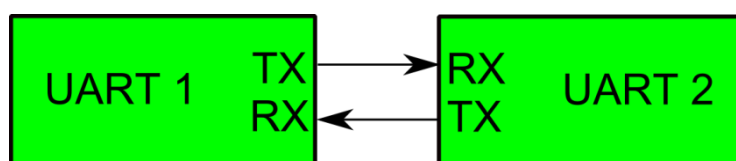


Figura 2.18: Ligação cruzada dos sinais RX e TX.

A designação “Universal” se deve ao fato de que a taxa de transmissão e o formato do pacote podem ser configurados [15]. É possível incluir no pacote um *bit* de paridade como medida de verificação de erros.

A designação “Assíncrono” indica que não há sinal de sincronismo entre as partes envolvidas na comunicação, o que faz com que sejam necessários *bits* de controle pra indicar o início e fim da comunicação.

O formato geral de um pacote enviado é mostrado na Figura 2.19. A linha em estado ocioso (*idle*) permanece em nível alto, até que haja uma condição de início. O pacote é iniciado com um *start bit*, colocando a linha em nível baixo. A seguir a palavra de dados é enviada, sendo que seu tamanho pode variar de 5 a 8 *bits*. A seguir é enviado o *bit* de verificação de paridade, e 1, 1,5 ou 2 *bits* de parada (*stop bit*). Após a parada, a linha retorna para estado ocioso, pronta para receber nova transação.

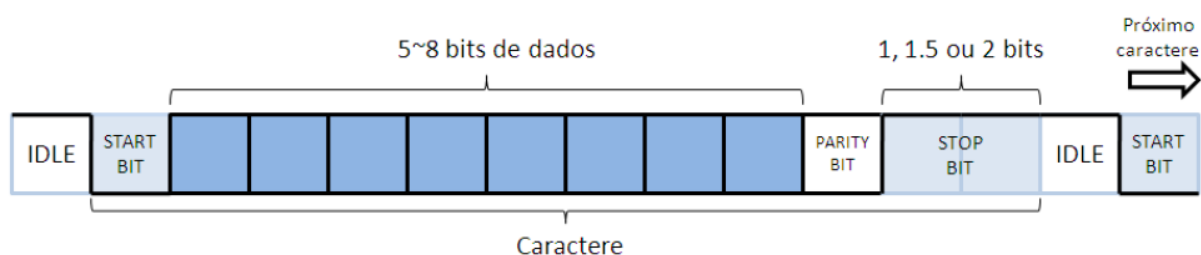


Figura 2.19: Formato geral de um pacote serial utilizando UART [16].

Nativamente, módulos UART geram sinais em nível TTL (0 a 5 V), porém entre os dispositivos pode haver circuitos conversores de nível elétrico, como conversores TTL/RS-232 ou TTL/RS-485. As placas-mãe de computadores mais antigos, por exemplo, eram fornecidas com portas seriais que operavam em nível RS-232.

## 2.6 Serial Peripheral Interface (SPI)

### 2.6.1 Introdução

O protocolo SPI é um protocolo serial síncrono de comunicação, que opera em modo *full duplex*. Os dispositivos se comunicam em uma estrutura *Master/Slave* (Mestre/Escravo), onde pode

haver apenas um mestre, que inicia as transações, e um ou mais escravos, que respondem às instruções do mestre [22].

Este protocolo é adequado para comunicação entre um controlador e seus dispositivos periféricos, pois seu funcionamento é simples. Outro protocolo largamente utilizado é o protocolo I2C (*Inter-Integrated Circuit*). O microcontrolador utilizado possui um módulo I2C, mas o protocolo SPI foi eleito por conta da sua elevada simplicidade (não utiliza endereçamento), quando comparado com o I2C. A seguir são expostas algumas vantagens e desvantagens do protocolo SPI.

### **Vantagens**

- Comunicação *full duplex*;
- *Throughput* (taxa de transferência) maior que do protocolo I2C;
- Escolha arbitrária do tamanho, conteúdo e função dos pacotes;
- Interface de *hardware* extremamente simples;
- Não requer endereçamento;
- O mestre proporciona o sinal de sincronismo aos escravos, fazendo desnecessária a inclusão de osciladores precisos para estes;
- Cada linha do barramento é unidirecional, facilitando a isolamento galvânica.

### **Desvantagens**

- Requer mais pinos no encapsulamento do que o protocolo I2C;
- Não possui protocolo de detecção de erros definido;
- Suscetível a picos de ruído;
- Suporta distâncias curtas quando comparado aos protocolos RS232 e RS485;
- Requer uma linha de seleção para cada escravo no barramento;
- Não suporta *hot plugging* (inserção de novo dispositivo no barramento em utilização).

Cada escravo em um barramento SPI possui um pino de seleção, comumente denominado *Chip Select* (CS) ou *Slave Select* (SS). Para iniciar a comunicação com um ou mais determinados escravos, o dispositivo mestre deve ativar este pino, informando o dispositivo escravo que uma nova transação endereçada a ele é iminente.

## 2.6.2 Operação

O barramento SPI especifica quatro sinais lógicos, expostos na Tabela 2.1, assim como nomenclaturas alternativas usuais para cada sinal. A Figura 2.20 ilustra uma topologia típica de um barramento SPI com um mestre e três escravos.

**Tabela 2.1: Nomenclatura do barramento SPI.**

| Nome | Nomes alternativos  | Descrição                                      |
|------|---------------------|------------------------------------------------|
| SCLK | SCK, CLK            | Clock (sinal de sincronismo)                   |
| MOSI | SIMO, SDO, DO, DOUT | Master Output, Slave Input (saída do mestre)   |
| MISO | SOMI, SDI, DI, DIN  | Master Input, Slave Output (entrada do mestre) |
| SS   | CS, SYNC            | Slave Select (seleção do escravo)              |

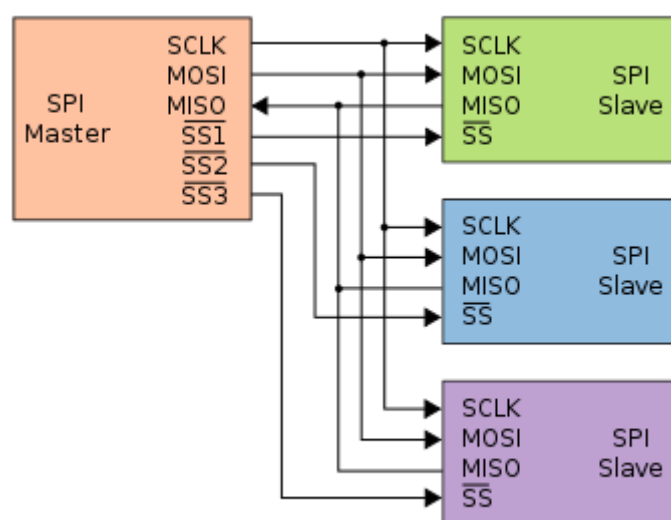


**Figura 2.20: Topologia típica de barramento SPI [17].**

Ao início de uma transação, o dispositivo mestre ativa (normalmente colocando em nível lógico baixo) a linha CS do escravo em questão. Uma vez que CS está ativa, o mestre inicia a transmissão do sinal de sincronismo, SCLK. Este sinal pode ser configurado em quatro modos, variando-se dois parâmetros: polaridade (CPOL) e fase (CPHA).

A polaridade define qual é o estado estável de SCLK, isto é, qual seu nível lógico de repouso. Se CPOL = 0, a primeira transição de SCL é uma borda de subida; se CPOL = 1, a primeira transição de SCLK é uma borda de descida.

A fase define se a amostragem dos *bits* começa na primeira ou na segunda borda de transição de SCLK. Até o fim da transmissão, a cada transição do sinal SCLK um *bit* é lido (amostrado) pelo

escravo e pelo mestre, sendo a direção da transição dependente da polaridade de SCLK. Terminada a transmissão dos *bits*, SCLK retorna a seu estado de repouso, e a linha CS é desativada. Na maioria dos dispositivos, quando o pino CS está desativado o pino MISO é mantido em alta impedância, de modo que outro dispositivo escravo conectado ao barramento possa controlar a linha MISO sem danificar outros dispositivos.

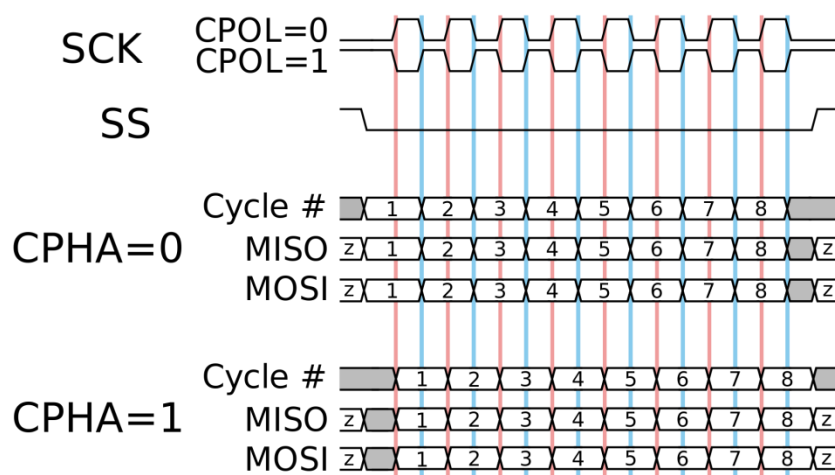


Figura 2.21: Diagrama de tempo para um barramento SPI [17].

O diagrama temporal mostrado na Figura 2.21 apresenta o funcionamento do barramento SPI configurado para pacotes de 8 *bits*, com CPOL e CPHA nas quatro configurações possíveis. As linhas vermelhas indicam o instante da amostragem das linhas MISO e MOSI para CPHA = 0, e as linhas azuis para CPHA = 1.

## 2.7 Servomotor

Um servomotor é um atuador rotativo que permite controle preciso de posição, velocidade e aceleração angulares. Ele consiste em um motor acoplado a um sensor que proporciona realimentação da posição [18]. Mini servomotores como o HXT900 [19], do fabricante Hextronik [20], visto na Figura 2.22, são amplamente utilizados, especialmente em aplicações de aeromodelismo. Este servo é controlado com um sinal de controle do tipo PWM (*Pulse Width Modulation*) [27].



Figura 2.22: Mini servomotor HXT900 [19].

O período do sinal de controle deve ser de 20 ms. Variando-se o tempo em que o sinal fica em nível alto entre 1 e 2 milissegundos, o ângulo do eixo do motor rotaciona entre 0 e 180 graus, respectivamente, como pode ser visto na Figura 2.23.

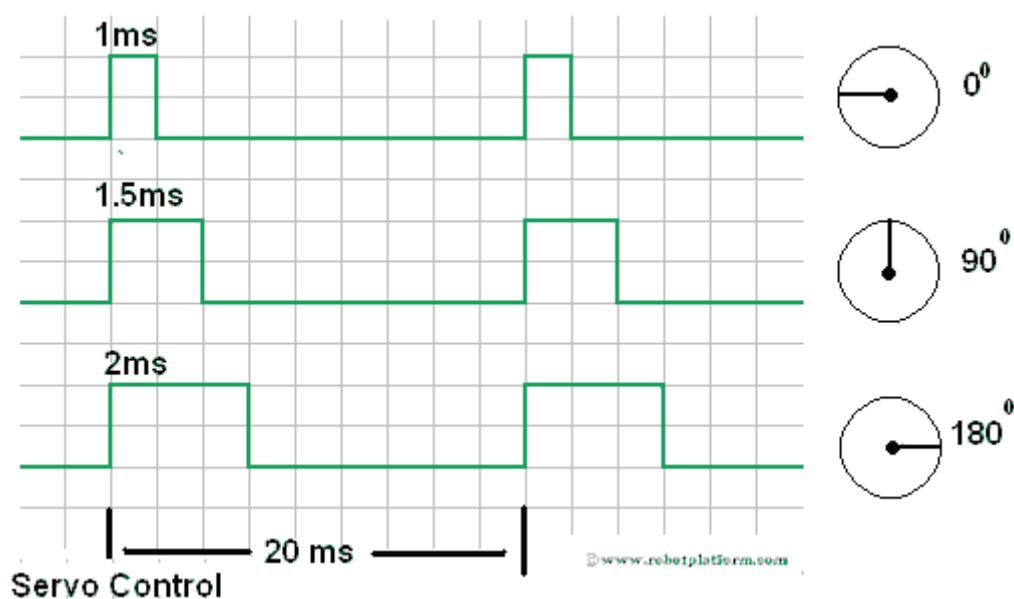


Figura 2.23: Diagrama de tempo do sinal de controle do servomotor, para 3 posições [21].

Não se deve tentar rotacionar o motor fora da faixa de 0 a 180 graus, pois isto pode danificar o potenciômetro interno que serve de sensor de posição.

### 3 Desenvolvimento do Projeto

#### 3.1 Visão geral

O sistema de orientação consiste em um microcontrolador, responsável por realizar a leitura de três sensores, processar as informações coletadas e gerar uma estimativa da orientação espacial atual do sistema, estimativa esta que pode ser transmitida ao mundo externo por meio de um ou mais métodos de interface. O diagrama de blocos visto na Figura 3.1 exibe os três blocos principais do projeto. Os três métodos de interface previstos estão reunidos no bloco à direita da imagem.

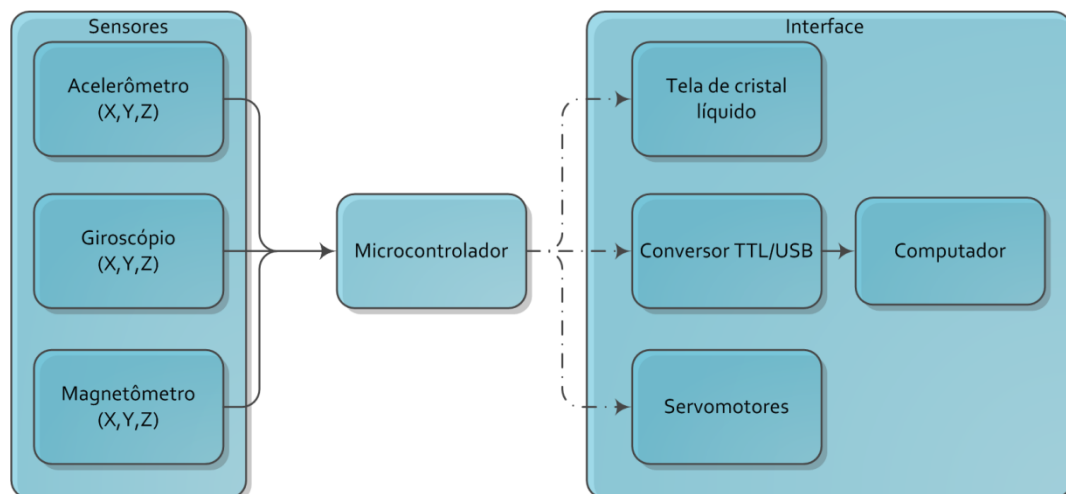


Figura 3.1: Diagrama de blocos geral do sistema de orientação.

A Figura 3.2 mostra a posição dos eixos do acelerômetro, giroscópio e magnetômetro. As setas indicam o sentido positivo das leituras.

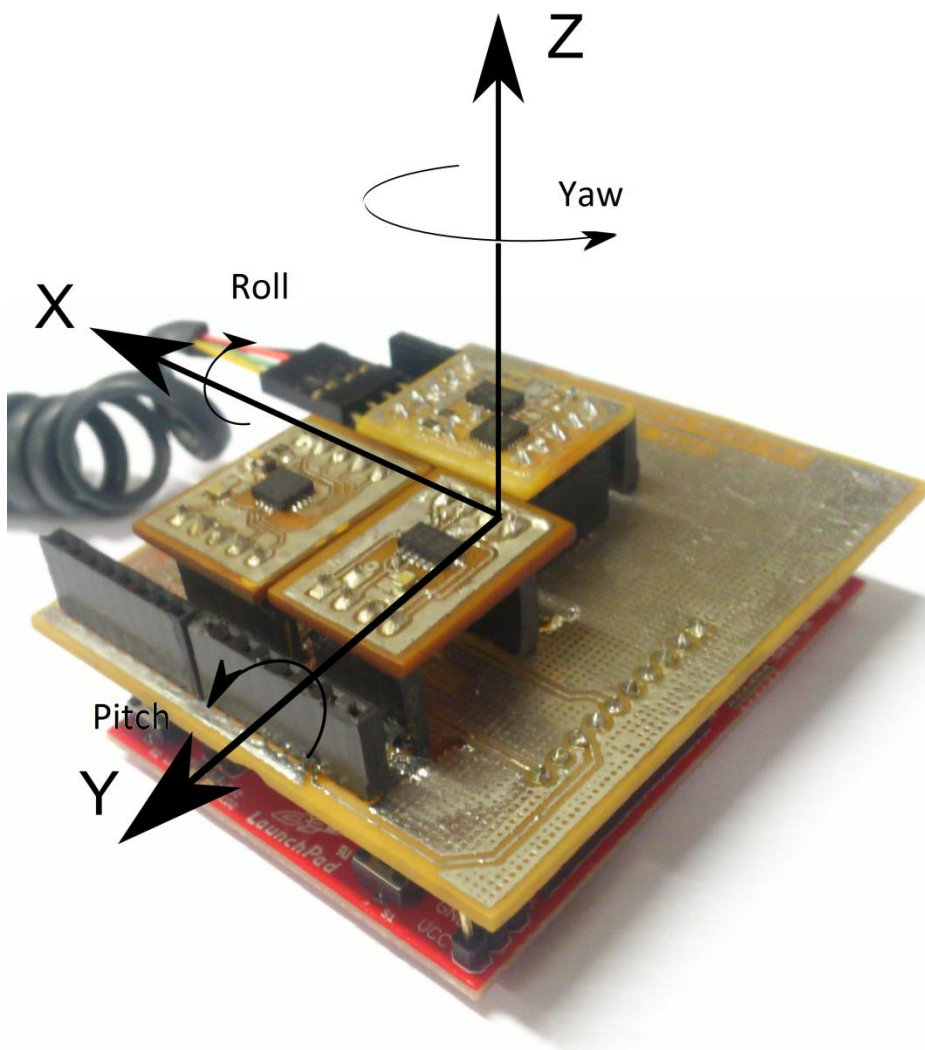


Figura 3.2: Eixos dos sensores no sistema.

As ferramentas de desenvolvimento computacionais (*softwares*) utilizadas neste trabalho estão listadas na Tabela 3.1.

Tabela 3.1: Lista de *softwares* utilizados.

| Software                     | Função                                                      |
|------------------------------|-------------------------------------------------------------|
| Code Composer Studio V4 [22] | Compilador do software para o microcontrolador              |
| Altium Designer V10 [23]     | Projeto das PCIs (placas de circuito impresso)              |
| Realterm [24]                | Terminal serial, captura dos dados no computador            |
| MATLAB V12 [25]              | Geração de vetores 3D a partir dos dados capturados         |
| KST V2.0.6 [26]              | Geração de gráficos temporais a partir dos dados capturados |
| SketchUp [27]                | Projeto mecânico da mesa de teste giratória                 |

## 3.2 Escolha dos componentes

### 3.2.1 Microcontrolador

O primeiro passo do projeto foi escolher uma plataforma de desenvolvimento, que é o conjunto microcontrolador/programador. Uma vez que se almeja um sistema embarcado, o microcontrolador deve consumir a menor potência possível. Duas famílias muito populares de microcontroladores de baixa potência são a família MSP430, do fabricante Texas Instruments, e a arquitetura ARM, desenvolvida por ARM Holdings. A família ARM é amplamente utilizada em dispositivos portáteis, como *smartphones* e *tablets*. A família MSP430 foi escolhida, pois a Texas Instruments comercializa um *kit* de desenvolvimento de baixo custo, que permite rápida programação e depuração (via cabo USB) nos microcontroladores, a um custo acessível. O *kit*, chamado **LaunchPad** [28] (MSP-EXP430G2, visto na Figura 3.3), é fornecido de fábrica acompanhado de um microcontrolador de baixa potência **MSP430G2553**, da linha Value Line, cuja pinagem pode ser observada na Figura 3.4. O *kit* possui a capacidade de programar microcontroladores MSP430 de outras linhas, utilizando os pinos de programação disponibilizados na lateral direita.

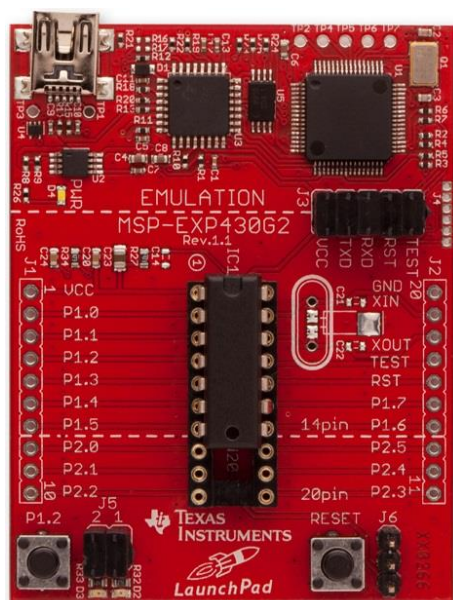


Figura 3.3: Kit de desenvolvimento LaunchPad [28].

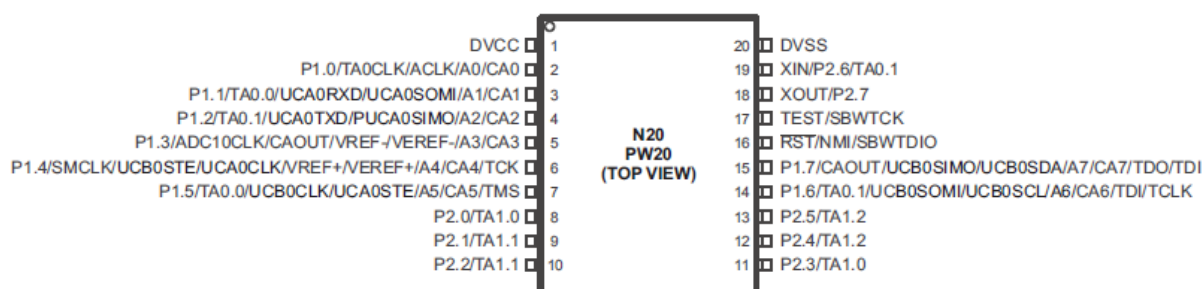


**Figura 3.4: Pinagem do microcontrolador MSP430G2553.**

A Tabela 3.2 reúne algumas especificações do microcontrolador utilizado. Apesar desta linha não ser a de mais baixo consumo deste fabricante, e também do encapsulamento DIP20 do CI não ser adequado, no quesito tamanho, à alta densidade requerida por sistema embarcado portátil, este microcontrolador ainda é adequado para o projeto, pois este ainda está na fase de prototipação, na qual não é importante que o projeto tenha alta eficiência energética nem apresente alta densidade de componentes na placa. Quando da realização do projeto final, pronto para uso pessoal, o microcontrolador deverá ser substituído por outro MSP430 com maior desempenho, menor consumo e menor tamanho físico. Como o código foi bem modularizado, ele é facilmente portátil para outros microcontroladores da mesma família.

**Tabela 3.2: Características do microcontrolador MSP430G2553.**

|                                      |                              |
|--------------------------------------|------------------------------|
| Faixa de alimentação                 | <b>1,8 V a 3,6 V</b>         |
| Consumo (modo ativo, @ 1 MHz, 2,2 V) | <b>230 <math>\mu</math>A</b> |
| UART                                 | <b>1</b>                     |
| SPI                                  | <b>Até 2</b>                 |
| I2C                                  | <b>1</b>                     |
| Frequência máxima                    | <b>16 MHz</b>                |
| Encapsulamento                       | <b>DIP20</b>                 |
| Gerador de PWM                       | <b>Até 4</b>                 |
| Memória RAM                          | <b>512 bytes</b>             |

Na Figura 3.5 é mostrado o sistema conectado ao computador pelo cabo USB, durante programação.

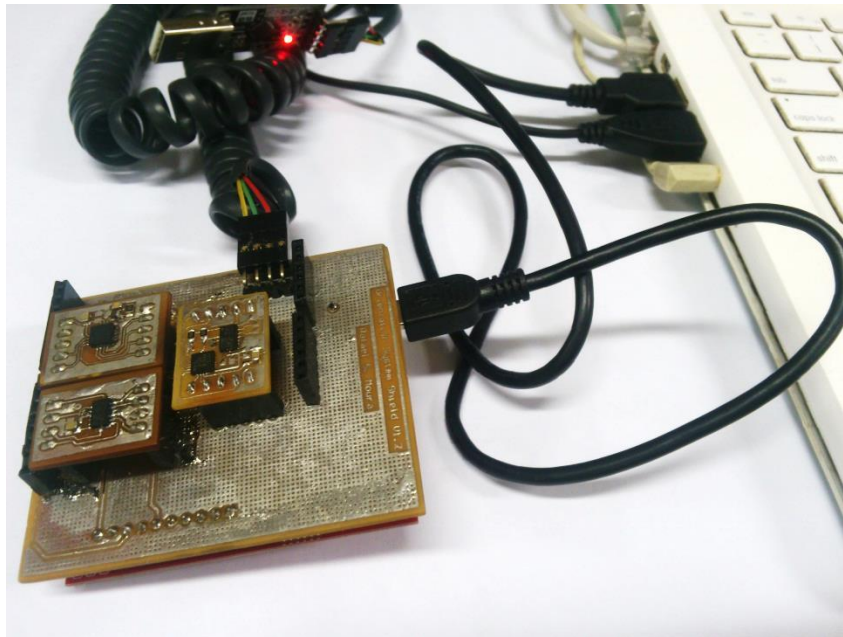


**Figura 3.5: Sistema de orientação sendo programado via cabo USB.**

Durante a execução do projeto, alguns obstáculos oriundos das limitações técnicas do controlador escolhido surgiram, o que causou problemas que serão discutidos posteriormente na seção 4.2. Tais problemas podem ser facilmente sanados utilizando-se microcontroladores mais adequados, o que não foi feito durante o projeto por falta de recursos financeiros e de tempo.

### **3.2.2 Sensores**

A escolha dos três sensores utilizados se baseou em três parâmetros principais:

- Presença de saída digital;
- Sensores tri-axiais;
- Custo.

Outros fatores, como faixa de alimentação, consumo de potência e fundo de escala de leitura, não foram considerados fatores limitantes, pois são atendidos por todos os sensores avaliados. A lista completa de sensores avaliados pode ser acessada em [30], [31] e [32].

Sensores com saídas digitais são desejáveis para eliminar a necessidade de conversão de sinais analógicos para sinais digitais compreensíveis pelo microcontrolador. Caso os sensores utilizados possuíssem apenas saídas analógicas, seria necessário utilizar o conversor analógico-digital embutido no microcontrolador, o que aumentaria a complexidade e tempo de execução do *software*, ou um conversor externo, o que aumentaria o tamanho e custo do sistema.

Os sensores possuem versões com um ou mais eixos de sensibilidade. Buscaram-se sensores com três eixos de sensibilidade ortogonais entre si (tri-axiais) visando à diminuição de custo, complexidade e tamanho do sistema, e o aumento do desempenho. A utilização de nove sensores uni-axiais (três acelerômetros, três giroscópios e três magnetômetros independentes) quase triplicaria o custo de aquisição dos sensores, mais que triplicaria o tamanho das placas, e tornaria o barramento de comunicação SPI muito maior. O aumento do desempenho é consequência da qualidade do alinhamento dos sensores feito durante a fabricação do circuito integrado. A precisão do alinhamento seria fortemente prejudicada se este fosse feito manualmente.

Finalmente, buscaram-se dispositivos que fossem economicamente tangíveis. Há diversas faixas de qualidade dos sensores, compreendendo desde a faixa de baixo-custo, destinada a produtos comerciais de baixa qualidade, até a faixa militar, cujos sensores suportam condições ambientais severas e apresentam altíssima precisão.

Considerados estes fatores, o fabricante escolhido foi a STMicroelectronics [29], e os componentes escolhidos foram:

#### Acelerômetro tri-axial: **LIS302DL** [30]

- Faixa de alimentação: 2,16 V a 3,6 V;
- Consumo de corrente: 0,3mA;
- Fundo de escala:  $\pm 2 / \pm 8$  g (selecionável).
- Precisão da leitura: 8 *bits*.

Os eixos de sensibilidade do acelerômetro são mostrados na Figura 3.6:

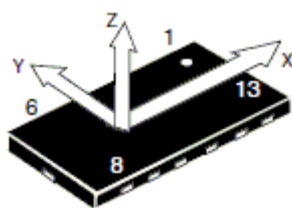


Figura 3.6: Eixos de sensibilidade do acelerômetro [30].

#### Giroscópio tria-axial: **L3G4200D** [31]

- Faixa de alimentação: 2,4 V a 3,6 V;
- Consumo de corrente: 6.1 mA;
- Fundo de escala:  $\pm 250 / \pm 500 / \pm 2000$  °/s (selecionável).
- Precisão da leitura: 16 *bits*.

Os eixos de sensibilidade do giroscópio são mostrados na Figura 3.7:

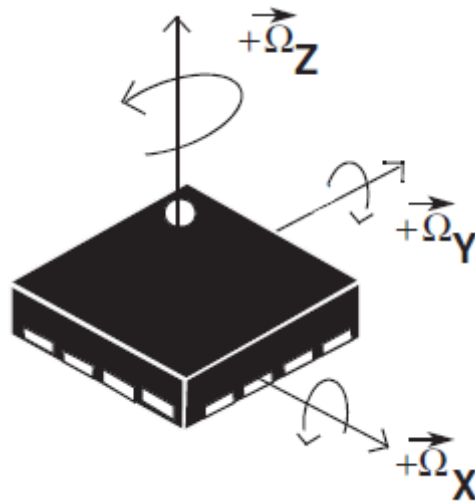


Figura 3.7: Eixos de sensibilidade do giroscópio [31].

Magnetômetro tri-axial: **LSM303DLHC** [32]

- Faixa de alimentação: 2,16 V a 3,6 V;
- Consumo de corrente: 0,11 mA;
- Fundo de escala:  $\pm 1.3$  /  $\pm 1.9$  /  $\pm 2.5$  /  $\pm 4.0$  /  $\pm 4.7$  /  $\pm 5.6$  /  $\pm 8.1$  gauss (selecionável).
- Precisão da leitura: 16 *bits*.

Os eixos de sensibilidade do magnetômetro são mostrados na Figura 3.8:

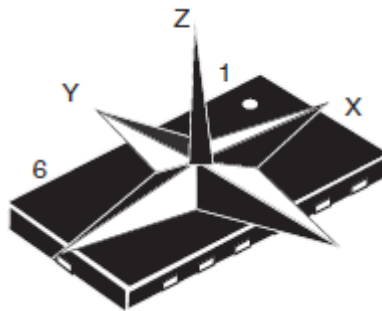


Figura 3.8: Eixos de sensibilidade do magnetômetro [32].

Observação: O dispositivo LSM303DLHC contém, além do magnetômetro, um acelerômetro tri-axial, que não foi utilizado, pois a escolha deste dispositivo foi feita após a implementação do acelerômetro LIS302DL estar completa.

### 3.3 Placas *breakout* para os sensores

Os sensores utilizados neste trabalho não são disponíveis em encapsulamentos adequados para prototipação, pois não possuem pinos *through-hole* (pinos que atravessam a placa de circuito impresso), e, portanto, não são facilmente inseridos em *proto-boards* nem em conectores do tipo barra de pinos fêmea. O encapsulamento dos sensores utilizados é do tipo LGA (*Land Grid Array*), visto na Figura 3.9, que apenas possui *pads* em sua parte inferior. Assim sendo, há a necessidade de construir placas adaptadoras contendo pinos, onde os sensores são soldados, possibilitando a prototipação rápida (sem necessidade de soldar os componentes numa placa final). Este tipo de placa adaptadora é comumente chamado *breakout*.

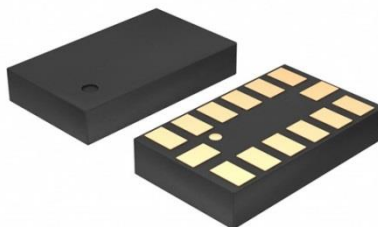


Figura 3.9: Encapsulamento LGA14 [33].

Para cada sensor foi feita uma placa *breakout* independente (O método de fabricação das placas de circuito impresso empregado neste projeto é discutido no APÊNDICE A). A placa do acelerômetro é a mais simples das três, consistindo simplesmente no CI do sensor **LIS302DL** e em dois capacitores de desacoplamento (C1 e C2). Capacitores de desacoplamento são ligados entre o terra e o positivo da alimentação, visando eliminar ruídos e oscilações na alimentação.

As figuras 3.10, 3.11 e 3.12 mostram o diagrama esquemático, o *layout* e uma foto da *breakout* do acelerômetro, respectivamente.

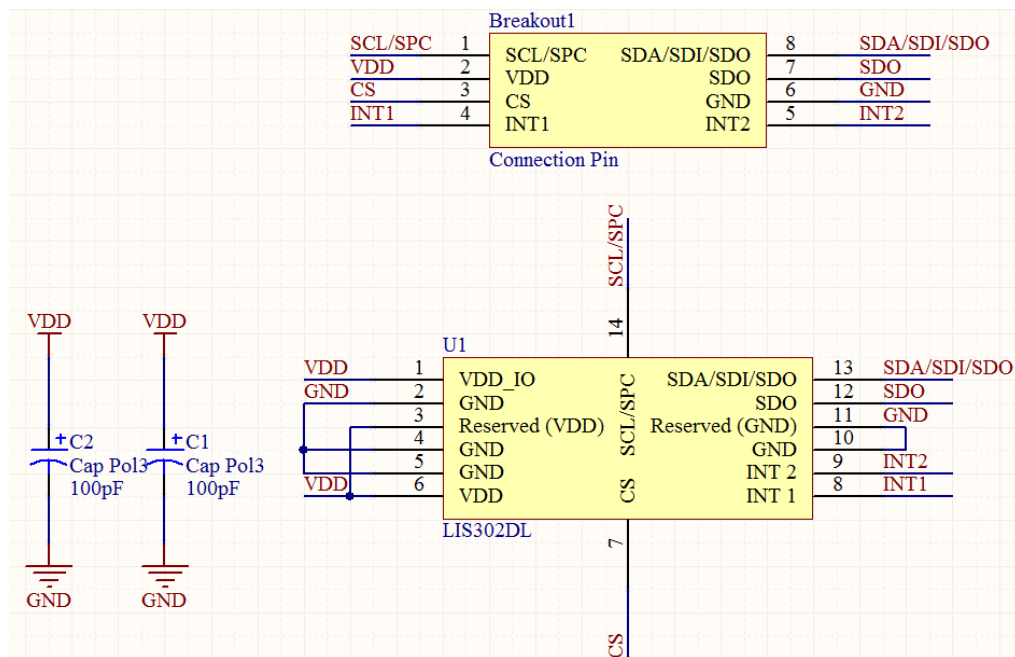


Figura 3.10: Diagrama esquemático da placa *breakout* para o acelerômetro.

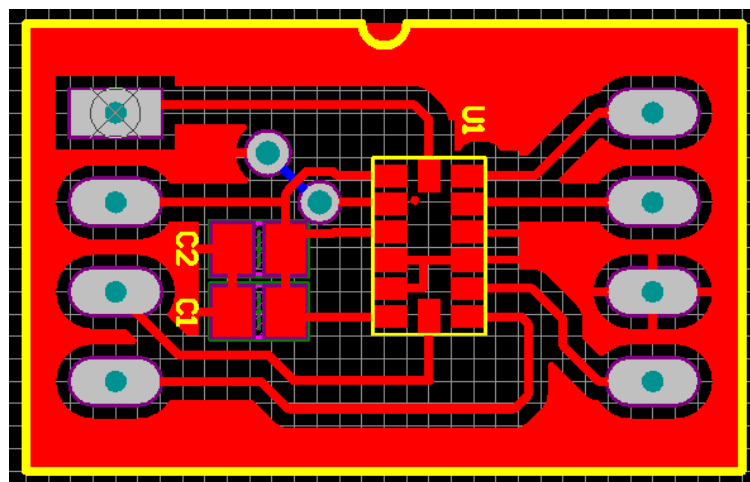


Figura 3.11: *Layout* da placa *breakout* para o acelerômetro.

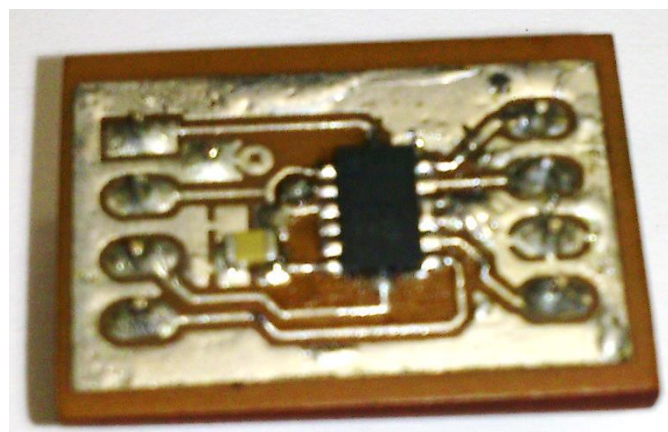


Figura 3.12: Foto da *breakout* do acelerômetro.

A placa do giroscópio **L3G4200D** também possui capacitores de desacoplamento (C1 e C2), além de um resistor (R1) e dois capacitores (C3 e C4) que formam um filtro exigido pelo giroscópio. O valor destes componentes foi escolhido em conformidade com a folha de dados do fabricante.

As figuras 3.13, 3.14 e 3.15 mostram o diagrama esquemático, o *layout* e uma foto da *breakout* do giroscópio, respectivamente.

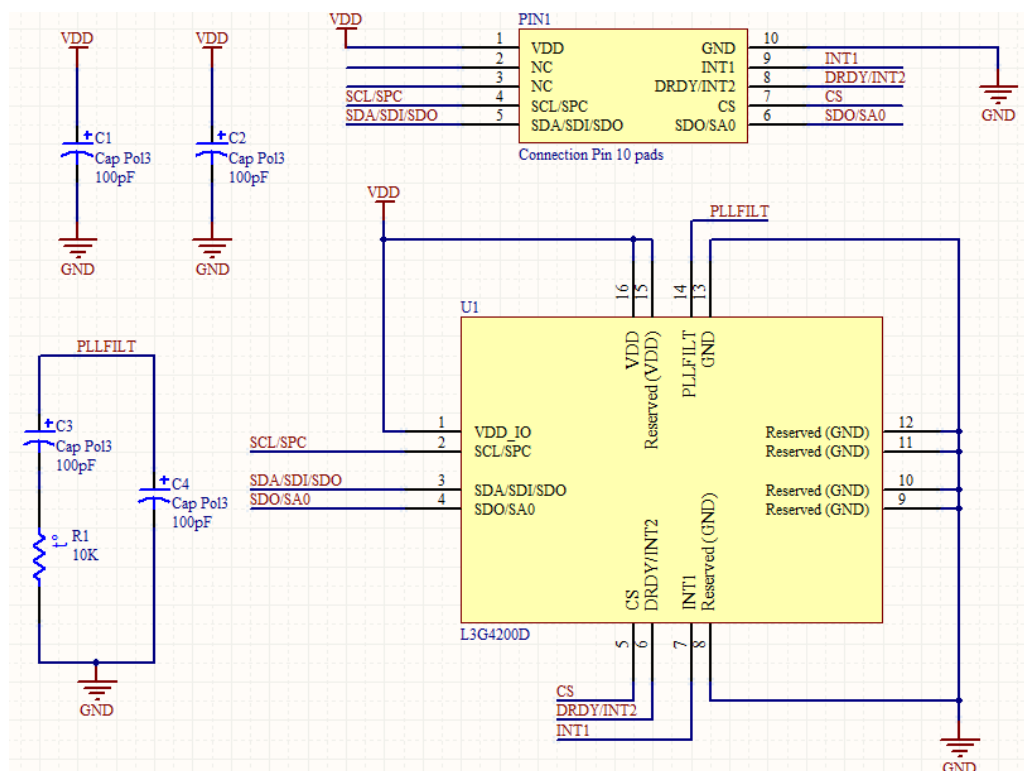


Figura 3.13: Diagrama esquemático da placa *breakout* para o giroscópio.

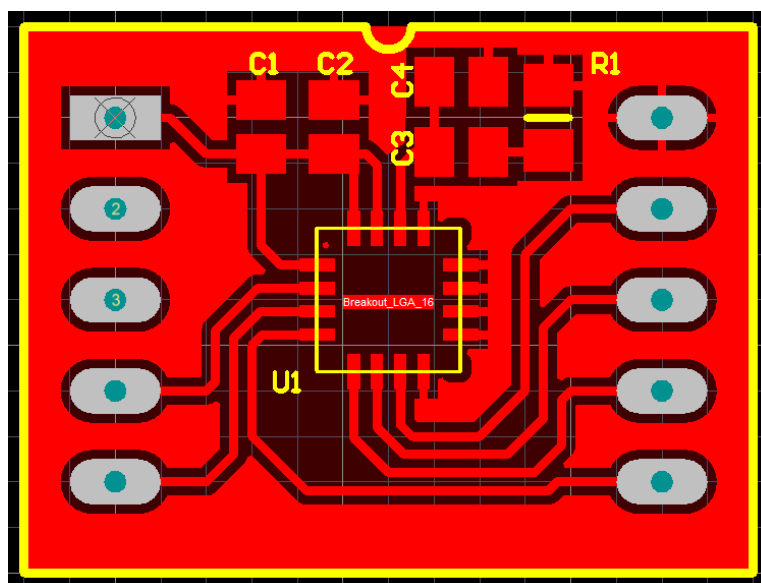


Figura 3.14: *Layout da placa breakout para o giroscópio.*

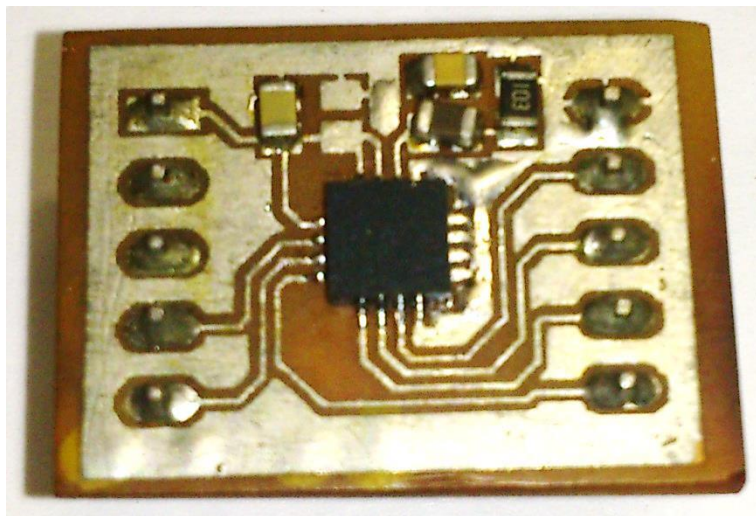


Figura 3.15: Foto da *breakout* do giroscópio.

A placa *breakout* do magnetômetro possui, também, capacitores de desacoplamento (C3 e C4), porém a adição de um CI conversor de protocolos de comunicação foi necessária. Este CI (**CP2120**, do fabricante Silicon Labs) faz a conversão entre os protocolos SPI e I2C, e é necessário pois o magnetômetro **LSM303DLHC** só possui um módulo de comunicação I2C, não sendo capaz de utilizar o protocolo SPI, que foi o escolhido para a comunicação entre os sensores e o microcontrolador. Os dois resistores (R1 e R2) promovem o *pull-up* das duas linhas seriais do protocolo I2C.

As figuras 3.16, 3.17 e 3.18 mostram o diagrama esquemático, o *layout* e uma foto da *breakout* do magnetômetro, respectivamente.

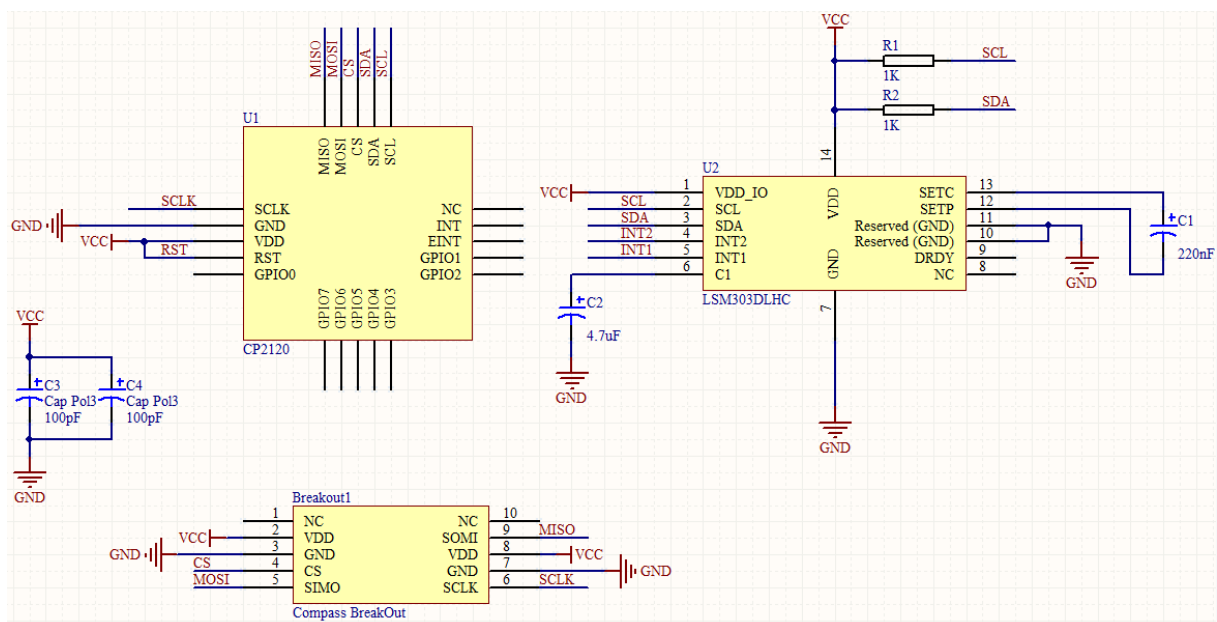


Figura 3.16: Diagrama esquemático da placa *breakout* para o magnetômetro.

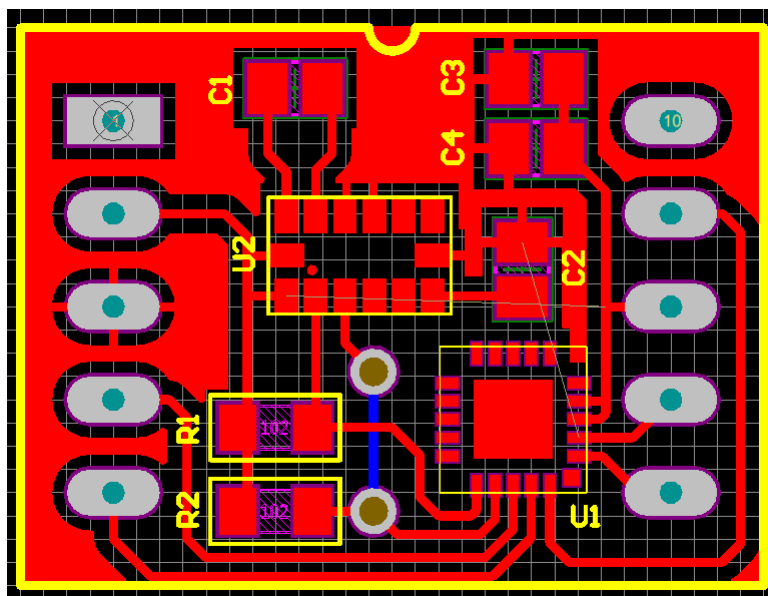


Figura 3.17: *Layout* da placa *breakout* para o magnetômetro.

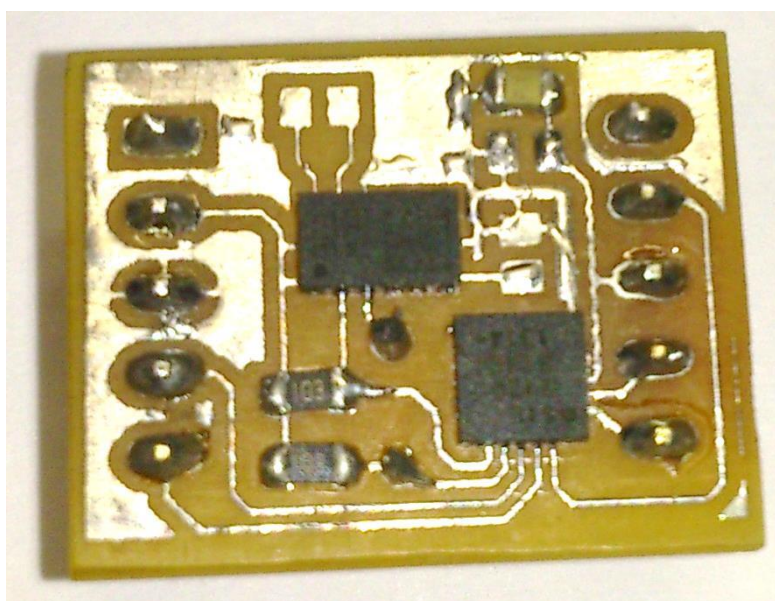


Figura 3.18: Foto da *breakout* do magnetômetro.

### 3.4 Placa acopladora (*Shield*)

Para fazer a interface entre o *kit* LaunchPad e as *breakouts* foi projetada e manufaturada uma placa acopladora. Esta placa é conectada ao LaunchPad, as *breakouts* e o cabo de comunicação com

o computador são conectados a ela. No mercado existem vários módulos de expansão para *kits* de desenvolvimento como o LaunchPad. Estes módulos, que incluem desde simples placas com matrizes de contatos até módulos de comunicação WiFi e GPS, são popularmente conhecidos como *shields* (escudos). As figuras 3.19, 3.20 e 3.21 mostram o diagrama esquemático, o *layout* e fotos do *shield* V1.2, respectivamente. A Figura 3.22 mostra o sistema com as *breakouts* inseridas.

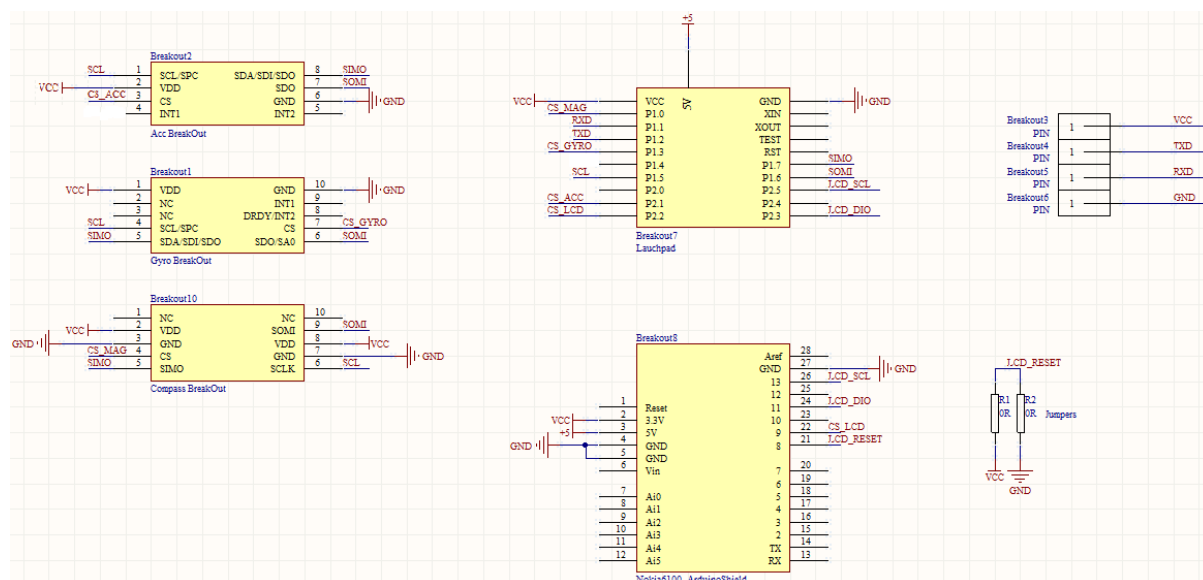


**Figura 3.19: Diagrama esquemático do *shield* V1.2.**

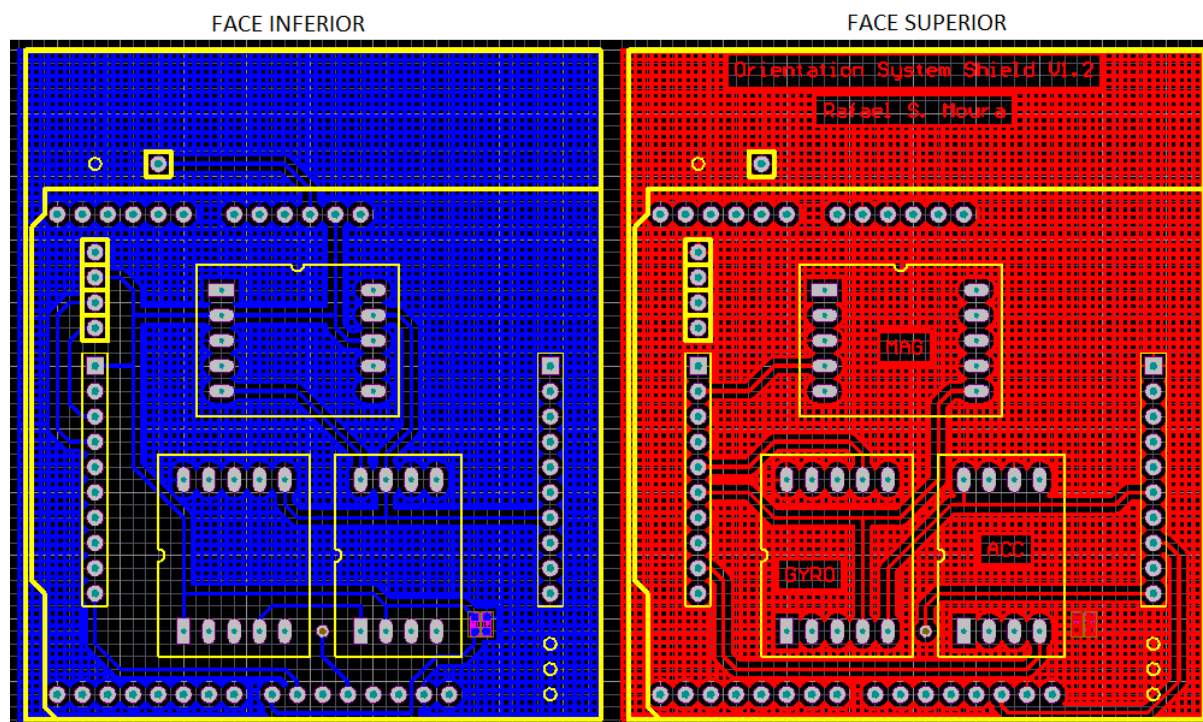
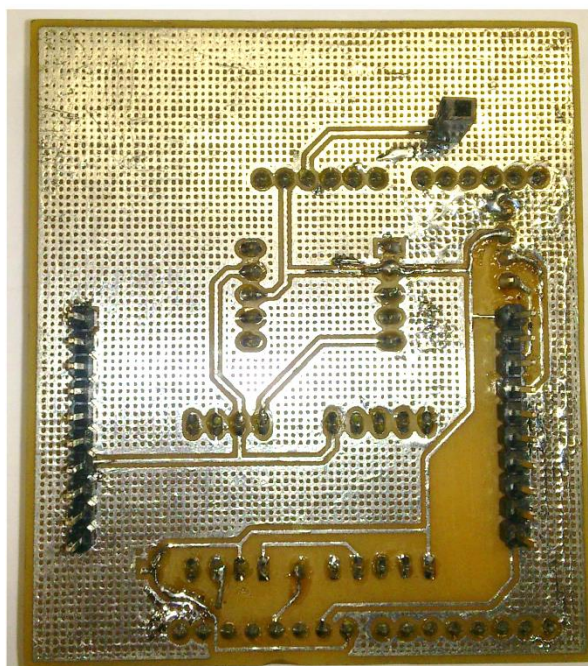
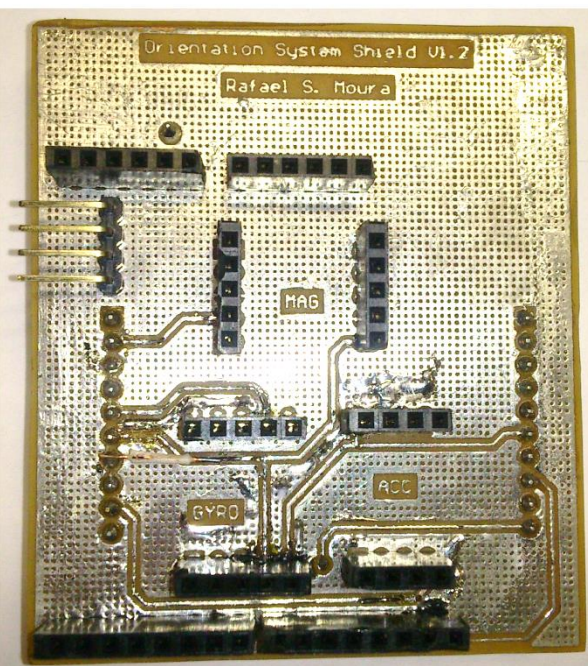
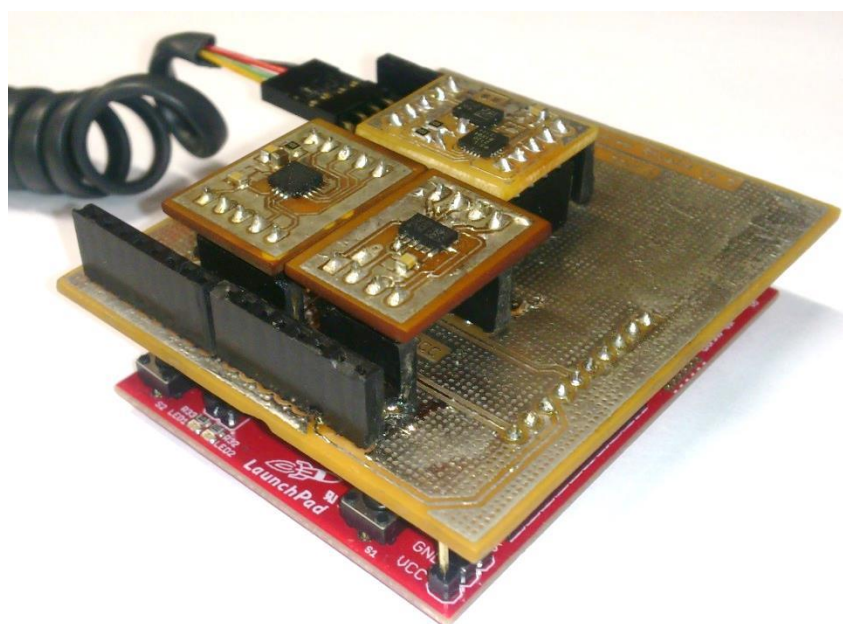


**Figura 3.20: Layout do shield V1.2.**

FACE INFERIOR



FACE SUPERIOR

Figura 3.21: Fotos do *shield* V1.2.Figura 3.22: Sistema de orientação com as *breakouts* inseridas.

Foi projetada e manufaturada, também, uma versão do *shield* (versão 1.3) com dois conectores destinados para a inserção de servomotores. Os servomotores podem ser utilizados, por exemplo, para fazer a estabilização de uma câmera fixada a eles. As figuras 3.23, 3.24 e 3.25 mostram o diagrama esquemático, o *layout* e fotos do *shield* V1.3, respectivamente.

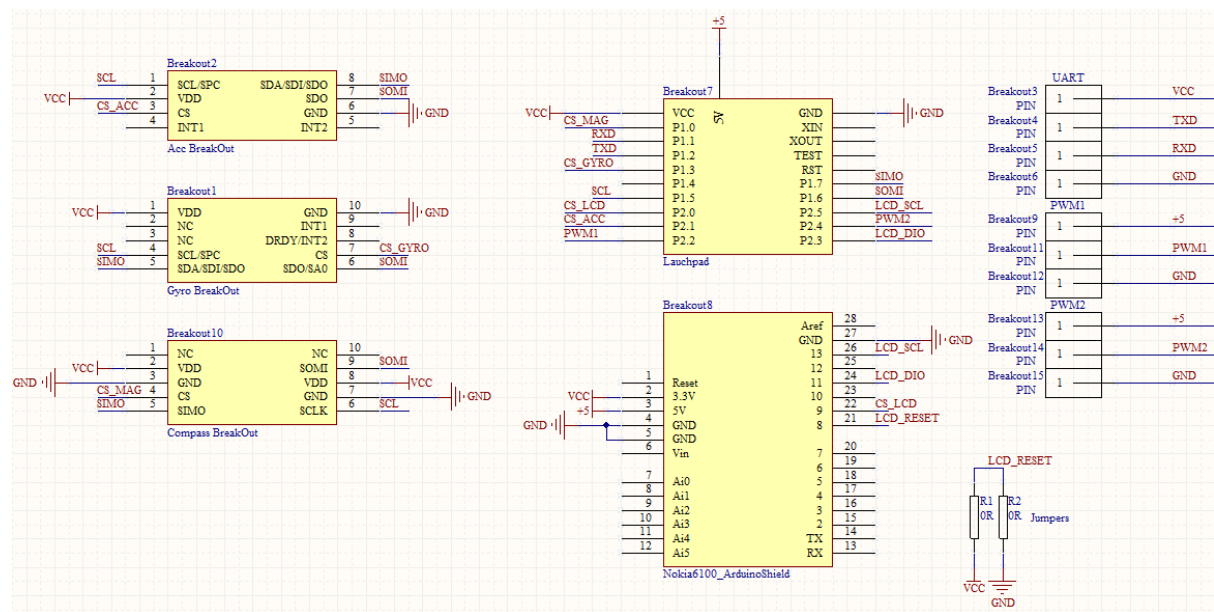


Figura 3.23: Diagrama esquemático do shield V1.3.

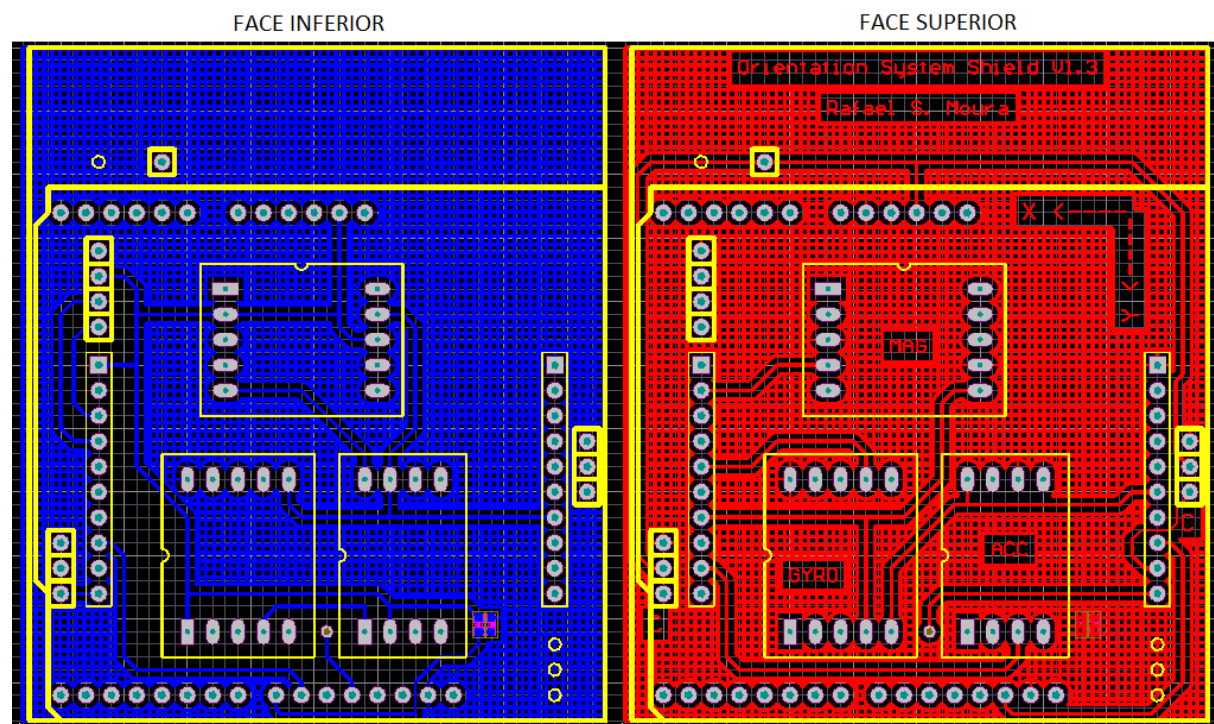


Figura 3.24: Layout do shield V1.3.

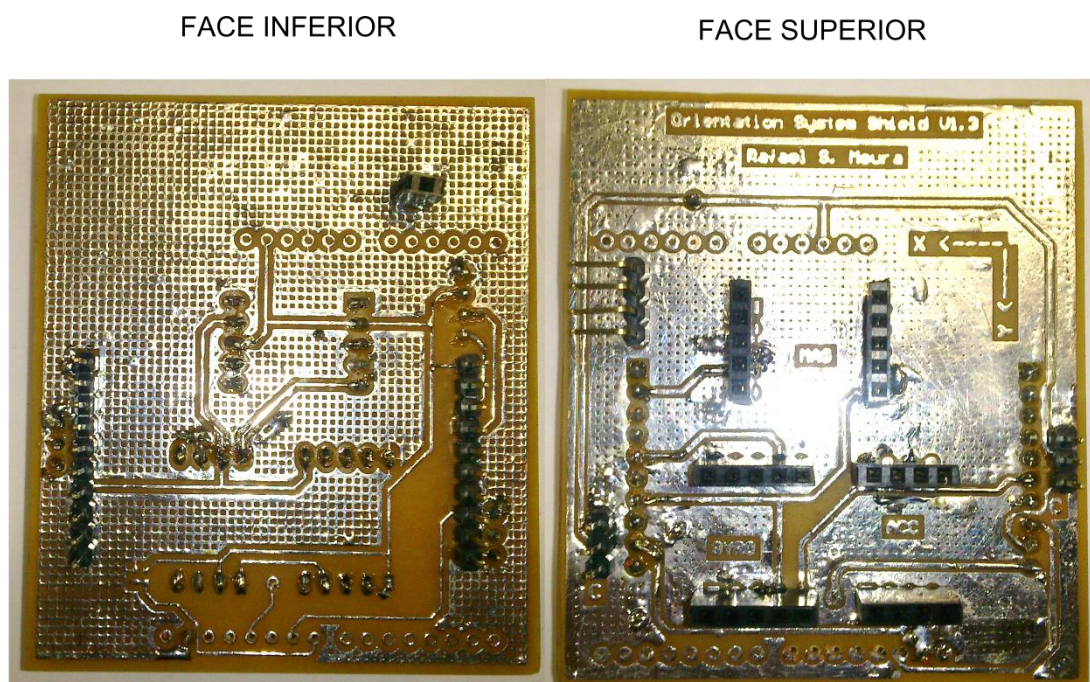


Figura 3.25: Fotos do *shield* V1.3.

No *shield* existem também conectores reservados para a inserção de uma *breakout* comercial projetada e comercializada pela loja SparkFun Electronics [34], vista na Figura 3.26, a qual possui uma tela de cristal líquido que era utilizada nos celulares Nokia modelo 6100, bem como o circuito *boost* para alimentar os LEDs de iluminação da tela. Este módulo foi escolhido por ser facilmente encontrado no mercado, e por ter custo reduzido. Não foi possível implementar o controle desta tela, como será explicado na seção 4.2. A foto da Figura 3.27 mostra o módulo da tela acoplado ao sistema. Na versão 1.3 os conectores reservados para a tela não foram soldados, pois quando da fabricação desta versão foi levada em consideração a impossibilidade de uso da tela com o microcontrolador utilizado.



Figura 3.26: Breakout da tela de cristal líquido.

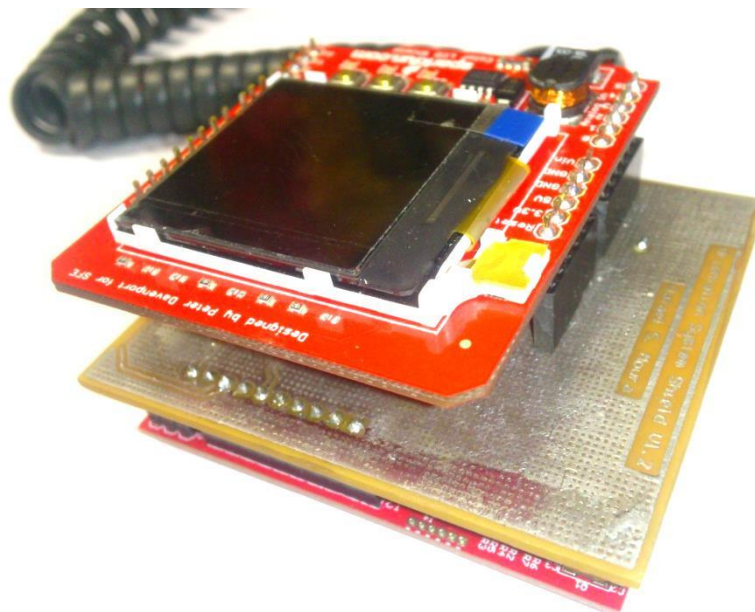


Figura 3.27: Módulo da tela de cristal líquido acoplado ao sistema de orientação.

Para conectar o sistema ao computador, foi utilizado um cabo espiralado de telefone, mostrado na Figura 3.28, com conectores do tipo MODU, mostrados em detalhe na Figura 3.29. O sistema conectado ao computador durante testes é mostrado na Figura 3.30. Foi escolhido um cabo deste tipo por conta de sua flexibilidade, que é necessária durante os testes onde o sistema é rotacionado rapidamente na mesa de testes.



Figura 3.28: Cabo de comunicação.



Figura 3.29: Conectores MODU.

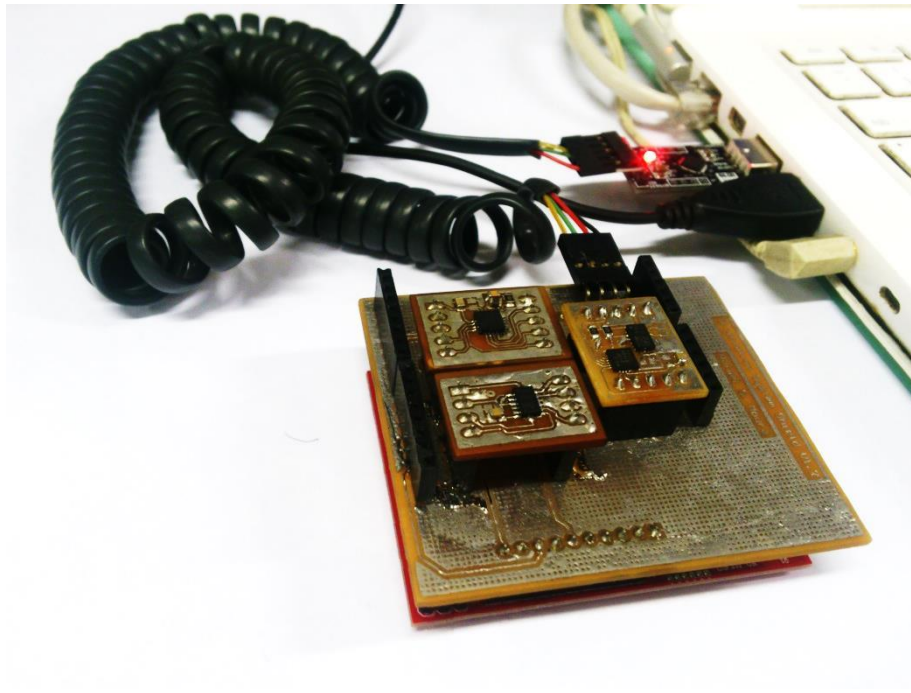


Figura 3.30: Sistema de orientação conectado ao computador via cabo de comunicação e conversor TTL/USB.

### 3.5 Mesa de testes rotativa

Para auxiliar no processo de testes durante o desenvolvimento, foi construída uma mesa giratória onde o sistema pode ser fixado e rotacionado. A mesa pode ser girada manualmente ou com o auxílio de um motor de corrente contínua de 12 V, utilizado em limpadores de para-brisa automotivos, acoplado à plataforma. Esta mesa de testes foi utilizada principalmente na fase de calibração do ganho dos sensores, pois o movimento proporcionado por ela é muito mais estável que o movimento manual. Os desenhos do projeto da mesa de testes e fotos são apresentados no APÊNDICE D.

A mesa é construída de modo que ofereça uma plataforma nivelada e paralela ao chão, proporcionando condições adequadas para a medição do sistema em condições estáticas e dinâmicas. Um tacômetro digital foi utilizado para medir a velocidade de rotação da mesa durante os testes com o giroscópio.

### 3.6 Software

Tomando como base conceitos fundamentais na área de Engenharia de *Software*, como organização e estruturação do fluxo de dados e códigos fonte, o código foi separado em módulos, levando em conta a função de cada bloco. Foram criados arquivos *header* (cabeçalho, extensão \*.h) contendo os protótipos das funções e/ou definições (de valores constantes e de endereços de registradores, por exemplo), e arquivos *source* (fonte, extensão \*.c) contendo o código fonte das funções de cada módulo. Todos os códigos fonte do projeto são apresentados no APÊNDICE B.

#### 3.6.1 Visão geral

O diagrama mostrado na Figura 3.31 exibe os módulos empregados, agrupados por função.

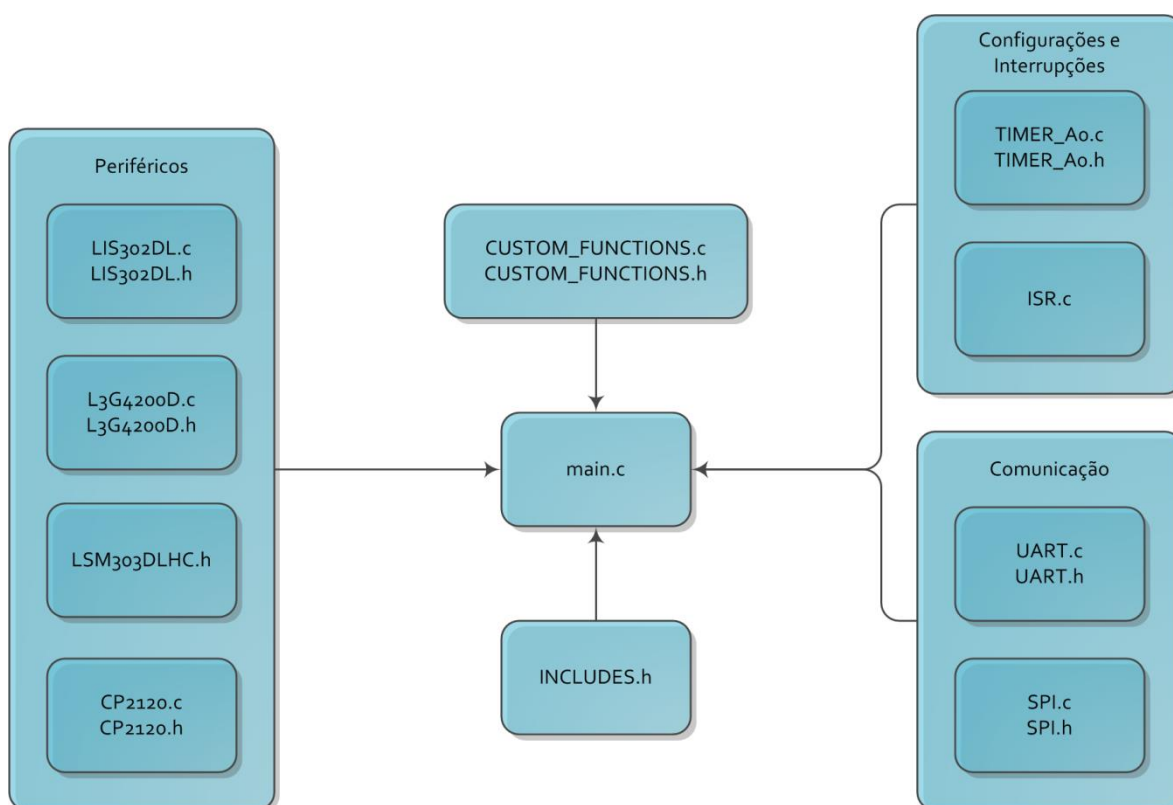


Figura 3.31: Módulos componentes do *software*.

Breve explanação sobre cada módulo:

- **Módulo TIMER\_A0**

Contém a rotina de inicialização do temporizador utilizado no sincronismo da amostragem, e a constante com o valor da frequência de amostragem. Contém também a inicialização do temporizador gerador de PWM.

- **Módulo ISR**

Contém a rotina de serviço à interrupção (*Interrupt Service Routine*) do timer de amostragem.

- **Módulo UART**

Contém a rotina de inicialização do módulo periférico UART do microcontrolador, e as rotinas de envio de um *byte* pela porta serial.

- **Módulo SPI**

Contém as rotinas de inicialização, envio e recepção do módulo periférico SPI do microcontrolador.

- **Módulos LIS302DL, L3G4200D, LSM303DLHC e CP212**

Contém as rotinas de leitura dos registradores internos de dados dos sensores, assim como as definições de endereços dos registradores.

- **Módulo CUSTOM\_FUNCTIONS**

Contém as funções não compreendidas pelos outros módulos.

- **Módulo INCLUDES**

Contém as diretivas de inclusão dos demais módulos.

- **Módulo Main**

Contempla o programa principal, que controla a execução do *software*.

O fluxograma presente na Figura 3.32 explicita simplificadaamente os passos que são seguidos pelo *software*.

Embora os módulos LSM303DLHC e CP212, responsáveis pela leitura do magnetômetro, estejam inclusos na estrutura deste projeto, eles não são utilizados pelo programa principal, pois não foi possível incluir o magnetômetro no projeto, como será explicado na seção 4.2.

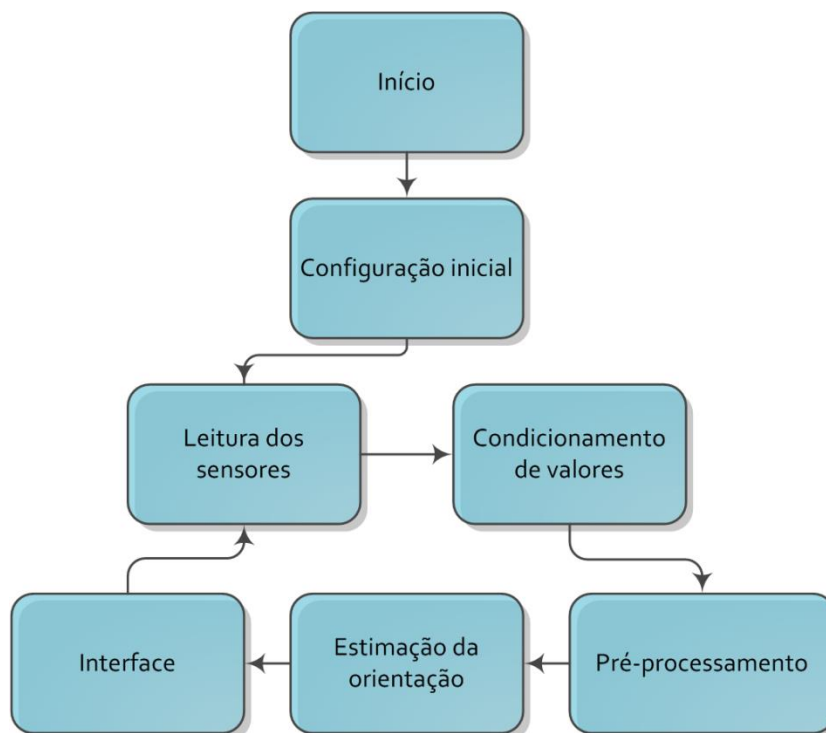


Figura 3.32: Fluxograma do *software*.

Após a configuração inicial ser concluída, o programa entra em um *loop* infinito, cujo primeiro passo é a leitura dos sensores, seguida do condicionamento dos valores adquiridos aos formatos adequados, pré-processamento dos dados condicionados (aplicação de filtros para “limpar” o sinal) e processamento principal (estimação da orientação), finalizando com a interface com o mundo externo. Esta interface pode ser realizada de várias formas, que serão discutidas na seção 3.6.4. Após a conclusão da interface, o ciclo recomeça.

Uma vez que os cálculos envolvidos neste trabalho envolvem integração dos sinais com relação ao tempo, é de vital importância que o período entre uma leitura completa dos dados de saída dos sensores e a seguinte seja constante e preciso. Sendo assim, o fluxo do programa é controlado por um temporizador interno, ajustado para permitir o início de um novo ciclo somente em intervalos específicos. A frequência de amostragem,  $f_a$ , é definida no módulo TIMER\_A0. Quanto maior o valor de  $f_a$ , melhores são os resultados obtidos, pois uma maior taxa de amostragem implica em menores erros de integração, visto que o intervalo entre uma amostra e a próxima é reduzido. Inicialmente,  $f_a$  foi definido em 400 Hz, que é a maior frequência de amostragem permitida pelos sensores; no entanto,  $f_a$  precisou ser reduzido para 50 Hz, pelos motivos descritos na seção 4.2. Esta redução diminui a qualidade da estimação em termos de queda de precisão e aumento do tempo de resposta (a estimação oscila mais e leva mais tempo para responder a estímulos externos).

### 3.6.2 Aquisição de dados e processamento preliminar

O primeiro passo do *loop* é a aquisição dos dados. A aquisição é feita através da leitura dos sensores, pelo barramento SPI. A Figura 3.33 mostra as linhas MOSI (canal 1) e SCLK (canal 2) do barramento SPI durante o requerimento de leitura de um dos eixos do acelerômetro.

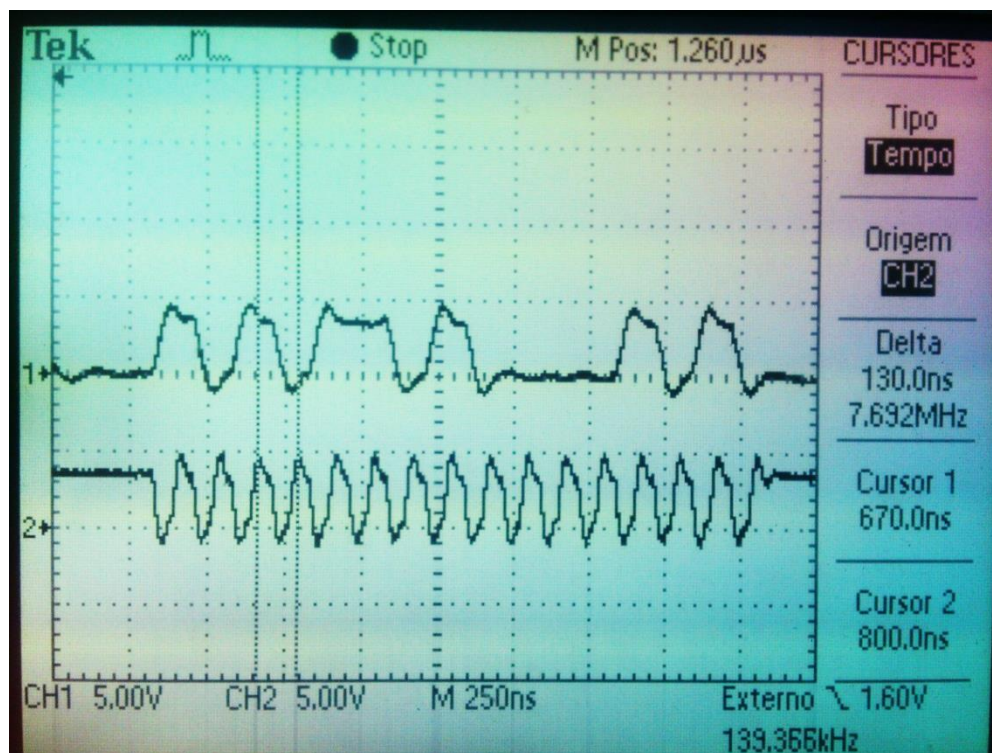


Figura 3.33: Um dos eixos do acelerômetro sendo lido com o protocolo SPI.

Após a aquisição dos dados, os valores são convertidos para as suas respectivas unidades (ver Tabela 3.3), e corrigidos para compensar os erros de *offset* e desvio no valor do ganho dos sensores. Idealmente, uma rotina de calibração deveria ser realizada durante a inicialização do sistema, calculando o valor médio de cada sinal e adotando este valor médio como o valor de *offset*, porém, por insuficiência de memória RAM disponível, este passo foi excluído, sendo substituído por um valor fixo de *offset* obtido experimentalmente, por meio de medições em condições estáticas.

Tabela 3.3: Unidades dos sinais condicionados dos sensores.

| Sensor       | Grandeza                  | Unidade                  |
|--------------|---------------------------|--------------------------|
| Acelerômetro | <b>Aceleração</b>         | <b>[m/s<sup>2</sup>]</b> |
| Giroscópio   | <b>Velocidade angular</b> | <b>[°/s]</b>             |
| Magnetômetro | <b>Campo magnético</b>    | <b>[G]</b>               |

Com a finalidade de filtrar os ruídos de alta frequência inerentes aos sensores, é aplicado um filtro passa-baixas digital a cada sinal corrigido. O filtro implementado é um filtro de **média móvel** temporal ponderado. Para uma amostra  $A(n)$  no instante  $n$ , é calculada uma amostra filtrada  $Af(n)$ , de acordo com a Equação 3.1, considerando 3 valores passados de  $A(n)$ .

$$Af(n) = \frac{W \cdot A(n) + A(n-1) + A(n-2) + A(n-3)}{W + 3} \quad \text{Equação 3.1}$$

$W$  é uma constante de ponderação, obtida empiricamente, utilizada para melhorar a resposta dinâmica do sinal filtrado. Sem essa constante de ponderação, a resposta dos sensores se torna muito lenta. Se o valor de  $W$  for muito alto, o filtro se torna supérfluo, pois a contribuição dos valores passados da leitura fica desprezível. Sendo assim, o valor determinado para  $W$  é um compromisso entre a velocidade de resposta e a qualidade da suavização do sinal. Na Figura 3.34 é mostrada a leitura dos três eixos do acelerômetro (eixo X: curva azul, eixo Y: curva vermelha, eixo Z: curva verde) antes (parte superior) e depois (parte inferior) da aplicação do filtro de média, com  $W = 4$ . O eixo vertical é dado em  $\text{m/s}^2$  e o eixo horizontal em número da amostra.

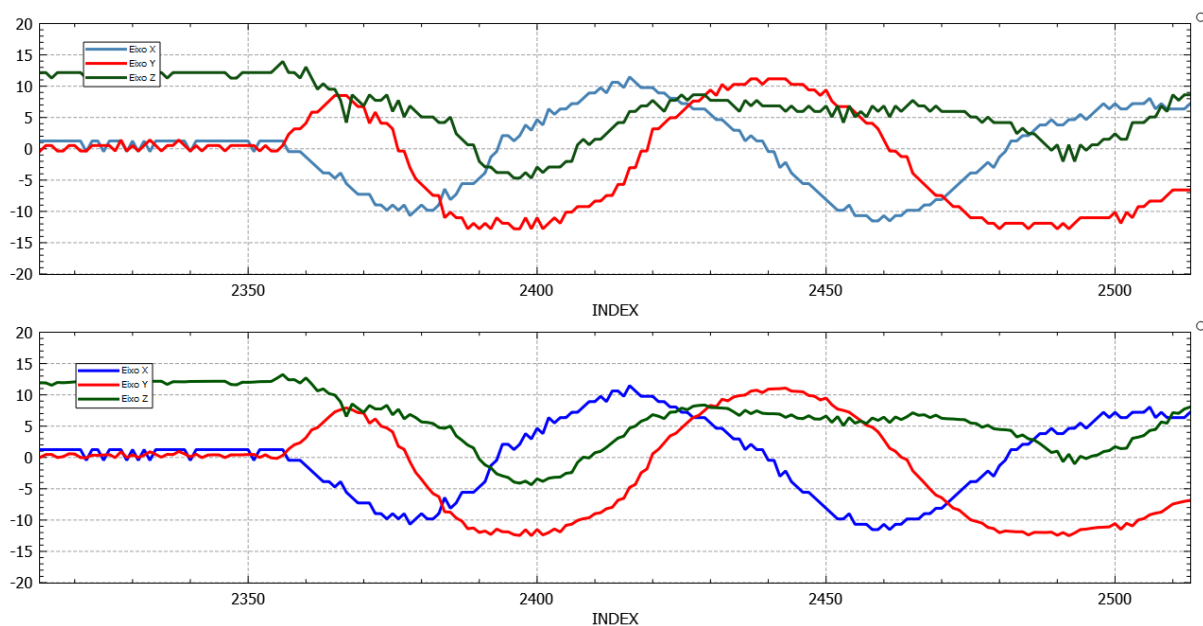


Figura 3.34: Sinais antes e depois da aplicação do filtro de média móvel.

Além do filtro de média móvel, outro artifício é necessário para a obtenção de um sinal adequado ao uso na estimativa da orientação: a **supressão de ruído próximo ao nível zero**. Os ruídos de alta frequência nas saídas dos sensores também estão presentes quando o nível do sinal está próximo de zero. Isto pode causar a modificação da orientação mesmo quando o sistema se

encontra em repouso, e para eliminar este efeito, o valor do sinal é forçado a zero se este valor estiver dentro de uma faixa estipulada em torno de zero, como pode ser visto na Equação 3.2, onde  $Afr(n)$  é o sinal filtrado pelo filtro de média móvel e pelo filtro de rejeição, e  $R$  é o limiar de rejeição.

$$Afr(n) = \begin{cases} Af(n), & \text{se } |Af(n)| > R \\ 0, & \text{se } |Af(n)| \leq R \end{cases} \quad \text{Equação 3.2}$$

O valor de  $R$  é definido empiricamente, grande o suficiente para eliminar a maior parte do ruído em torno do zero, porém pequeno o suficiente para não degradar demais o sinal, pois uma grande região “morta” implica em perda de sensibilidade do sensor. O efeito gerado por este filtro pode ser visto da Figura 3.35, onde a curva de cima mostra o sinal do eixo Y do acelerômetro antes de ser filtrado, e, a figura de baixo, após a supressão do ruído, com  $R = 0,9$ . O eixo vertical é dado em  $m/s^2$  e o eixo horizontal em número da amostra.

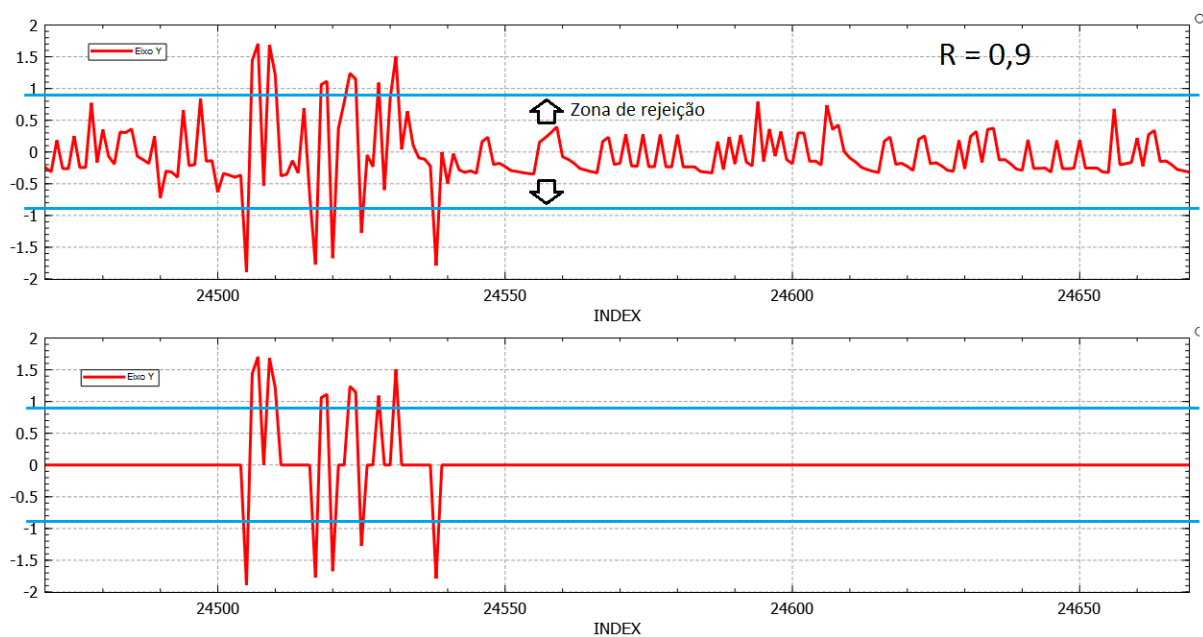


Figura 3.35: Sinal antes e depois da supressão de ruído próximo ao nível zero.

### 3.6.3 Estimação da orientação

Definem-se três vetores de estimação, que representam os eixos do sistema de coordenadas do sistema de orientação: H1, H2 e Vert. H1 é equivalente ao vetor  $X_C$ , H2 é equivalente ao vetor  $Y_C$ , e Vert equivale ao vetor  $Z_C$  (ver Figura 2.2).

O algoritmo de estimação toma as leituras do giroscópio como fonte principal de orientação, sendo que o acelerômetro é usado para fazer a correção da inclinação nos eixos *pitch* e *roll*. O diagrama de blocos da Figura 3.36 expõe simplificada o processo. Todos os sinais apresentados são vetores tridimensionais.

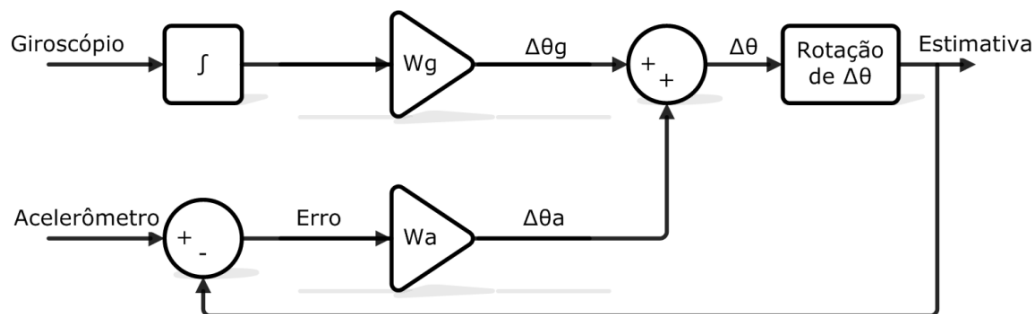


Figura 3.36: Diagrama de blocos simplificado do algoritmo de estimação.

Os sinais de velocidade angular do giroscópio são multiplicados pelo período de amostragem (integração) e multiplicados por uma constante de ponderação, o que resulta em uma variação dos ângulos (*pitch*, *roll* e *yaw*) com respeito à última amostra, representada por  $\Delta\theta_g$ . Os sinais do acelerômetro, por sua vez, que representam o vetor da aceleração gravitacional em relação ao referencial do sistema, são comparados com a estimativa anterior da orientação do vetor  $Z_C$  (eixo vertical em relação ao corpo), gerando um sinal de erro. Este sinal de erro é multiplicado por uma constante de ponderação, gerando outra variação dos ângulos,  $\Delta\theta_a$ , que será somada a  $\Delta\theta_g$ , compondo a variação total,  $\Delta\theta$  (como o acelerômetro só é capaz de corrigir os ângulos de *pitch* e *roll*, a componente referente ao *yaw* em  $\Delta\theta_a$  é nulificada). Os vetores de estimação são, então, rotacionados pelos três valores de variação de ângulo, gerando uma nova estimativa da orientação.

Relembrando, o vetor gerado pelo acelerômetro representa a aceleração da gravidade apenas se o dispositivo estiver em repouso ou com velocidade constante. Se houver outra aceleração agindo sobre o acelerômetro, o vetor gerado representa a resultante da soma entre a gravidade e a outra aceleração, o que, obviamente, não representa a aceleração da gravidade. É por este motivo que o giroscópio deve ser usado em conjunto com o acelerômetro para prover estimativas mais corretas sobre a orientação.

O algoritmo de estimação da orientação implementado neste trabalho estima corretamente apenas as inclinações do sistema em relação ao eixo  $Z_T$  (eixo vertical em relação à Terra), não

levando em conta as rotações em torno deste eixo, ou seja, estima apenas o *pitch* e o *roll*. Isto se deve ao fato de que, como o magnetômetro não foi utilizado, por razões explicadas na seção 4.2, não seria possível corrigir o ângulo de *yaw*. O diagrama de blocos simplificado do algoritmo incluindo o magnetômetro é mostrado na Figura 3.37, onde  $\Delta\theta_m$  possui apenas a componente referente ao *yaw*.

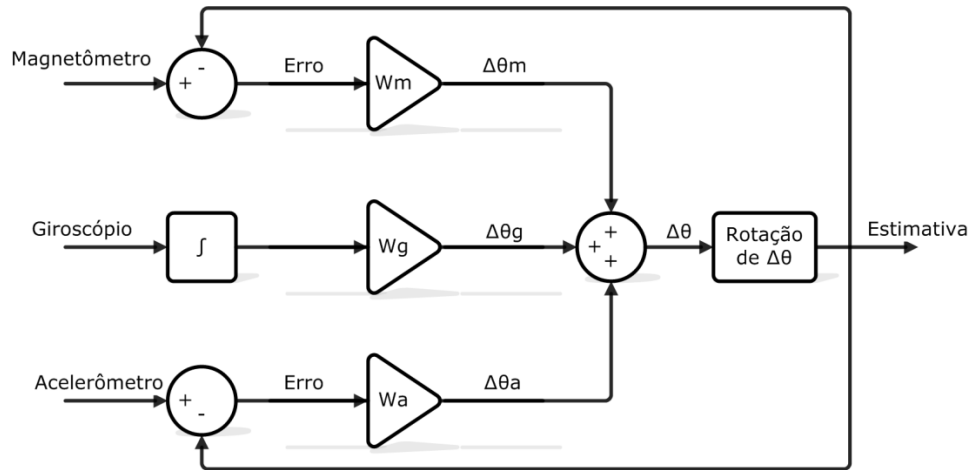


Figura 3.37: Diagrama de blocos simplificado, magnetômetro incluso.

O cálculo do vetor de erro, cujo nome no algoritmo é “PR\_Correct”, é feito pelo produto vetorial entre o vetor de estimação  $Z_c$  atual (“Vert” no algoritmo), e o vetor de referência da gravidade gerado pelo acelerômetro (VAcc no algoritmo), nesta ordem. As três componentes de PR\_Correct são calculadas pela Equação 3.3, onde a constante de valor 15 amplifica o erro, para que o sistema apresente convergência mais rápida.

$$PR\_Correct = 15 * (Vert \times VAcc)$$

Equação 3.3

Em seguida, a terceira componente de PR\_Correct, referente a rotações em *yaw*, é nulificada, ou seja,  $PR\_Correct[2] = 0$ . O próximo passo executado é o cálculo do vetor total de variação dos ângulos, calculado pela Equação 3.4:

$$\text{deltaAng} = \frac{\frac{T * Wg}{\text{SampleRate}} - Wa * PR\_Correct}{(Wa + Wg)} * \frac{\pi}{180}$$

Equação 3.4

Onde deltaAng é o vetor total de variação dos ângulos, T é o vetor de leitura do giroscópio, Wg é a constante de ponderação do giroscópio, SampleRate é a frequência de amostragem ( $f_a$ ), Wa é a constante de ponderação do acelerômetro e PR\_Correct é o vetor de correção de *pitch* e *roll*. O fator multiplicativo  $\frac{\pi}{180}$  converte os ângulos de graus para radianos.

Obtidos os ângulos de rotação, os vetores de estimação (Vert, H1 e H2) são atualizados por meio da aplicação da função “**rotate3D**”, que aplica nestes vetores uma matriz de rotação tridimensional em torno do eixo  $X_T$  e do eixo  $Y_T$ , denotada por  $M_{xy}$ . A matriz de aplicada é mostrada na Figura 3.38.

$$\begin{pmatrix} \cos(ROLL) & 0 & -\sin(ROLL) \\ \sin(PITCH) * \sin(ROLL) & \cos(PITCH) & \sin(PITCH) * \cos(ROLL) \\ \cos(PITCH) * \sin(ROLL) & -\sin(PITCH) & \cos(PITCH) * \cos(ROLL) \end{pmatrix}$$

**Figura 3.38: Matriz de rotação  $M_{xy}$ .**

Neste ponto, o algoritmo já calculou uma estimação vetorial da orientação do sistema, que pode ser exibida graficamente em um computador ou em uma tela de cristal líquido acoplada ao sistema. Para se fazer uso de atuadores mecânicos (no caso deste trabalho, servomotores), porém, é necessário calcular o valor numérico dos ângulos *pitch* e *roll*. Tal cálculo é feito de acordo com a Equação 3.5 e a Equação 3.6.

$$PITCH = \text{atan}\left(\frac{\text{Vert}[0]}{\text{Vert}[2]}\right) \quad \text{Equação 3.5}$$

$$ROLL = \text{atan}\left(\frac{\text{Vert}[1]}{\text{Vert}[2]}\right) \quad \text{Equação 3.6}$$

Onde Vert[0], Vert[1] e Vert[2] são as componentes X, Y e Z, respectivamente, de Vert. Estes cálculos são baseados nas propriedades trigonométricas dos vetores, que podem ser observadas na Figura 3.39.

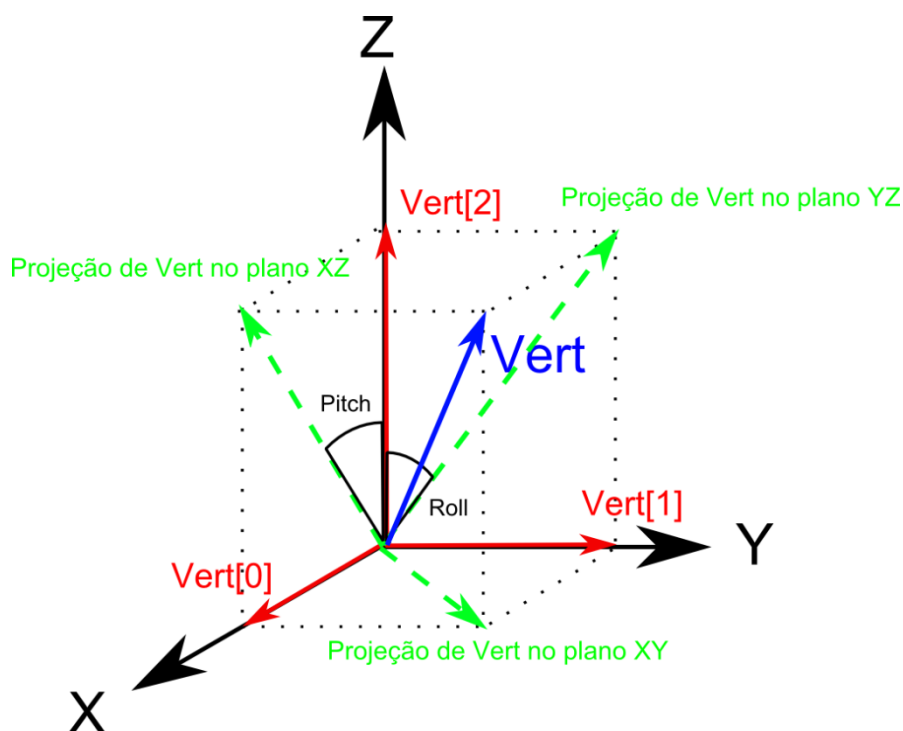


Figura 3.39: Propriedades trigonométricas do vetor Vert.

Contudo, quando o eixo  $Z_c$  fica paralelo ao plano horizontal, o denominador das frações tende a zero, resultando em valores incorretos, pois a fração tende a mais ou menos infinito, independentemente do valor dos numeradores. Para contornar este efeito, são levados em conta os eixos não presentes em cada equação, da seguinte forma:

$$\text{PITCH} = \text{atan}\left(\frac{\text{Vert}[0]}{\text{sin}(\text{Vert}[2]) * \sqrt{\text{Vert}[2]^2 + \text{Vert}[1]^2}}\right) \quad \text{Equação 3.7}$$

$$\text{ROLL} = \text{atan}\left(\frac{\text{Vert}[1]}{\text{sin}(\text{Vert}[2]) * \sqrt{\text{Vert}[2]^2 + \text{Vert}[0]^2}}\right) \quad \text{Equação 3.8}$$

Assim, o numerador das frações nunca se aproxima de zero. No código em C foi utilizada a função “**atan2**”, que leva em conta o sinal dos operandos, e é capaz de fornecer os ângulos corretos em todos os octantes.

### 3.6.4 Comunicação externa

A interface com o ambiente externo pode ser feita de diversas formas, dependendo da aplicação desejada. Um modo é utilizar a estimação da orientação para controlar servomotores fixados ao sistema, de modo que, por exemplo, uma câmera acoplada ao mecanismo fique sempre na horizontal, independentemente da orientação do sistema. Outro modo seria utilizar um visor acoplado ao sistema para exibir informações de orientação, ou direções para pontos de referência determinados anteriormente. Neste trabalho, durante o desenvolvimento, a evolução do projeto e os testes realizados foram observados utilizando um computador para mostrar os dados de forma gráfica. Durante a fase inicial do projeto, foi utilizado o *software* KST para exibir gráficos em tempo real das variáveis pertinentes. Na fase final do projeto, quando já se dispunha de estimações sobre a orientação do sistema, os dados passaram a ser avaliados utilizando o *software* MATLAB, pois este possui ferramentas gráficas prontas que tornaram possível a exibição de vetores tridimensionais, ficando imediata a interpretação dos resultados.

Houve interesse em utilizar o *software* MATLAB desde o início do projeto, para facilitar e acelerar a prototipação dos algoritmos, porém foi fator limitante a necessidade de uma frequência de amostragem fixa, pois o *software* é deficiente em ferramentas para sistemas em tempo real, isto é, sistemas com demandas de temporização controlada. Desta forma, foi utilizado o *software* KST, que é uma ferramenta leve e simples para traçar gráficos. Este programa é capaz de verificar continuamente se há alterações em um arquivo de texto no formato **CSV** (*Comma Separated Values* – Valores Separados por Vírgula) contendo os valores a serem traçados, e atualizar o gráfico sem intervenção humana. O *software* de terminal virtual RealTerm, visto na Figura 3.40, por sua vez, tem a capacidade de gravar continuamente em um arquivo de texto os dados recebidos pela porta serial do computador. Desta maneira, é possível visualizar em tempo real as variáveis de interesse.

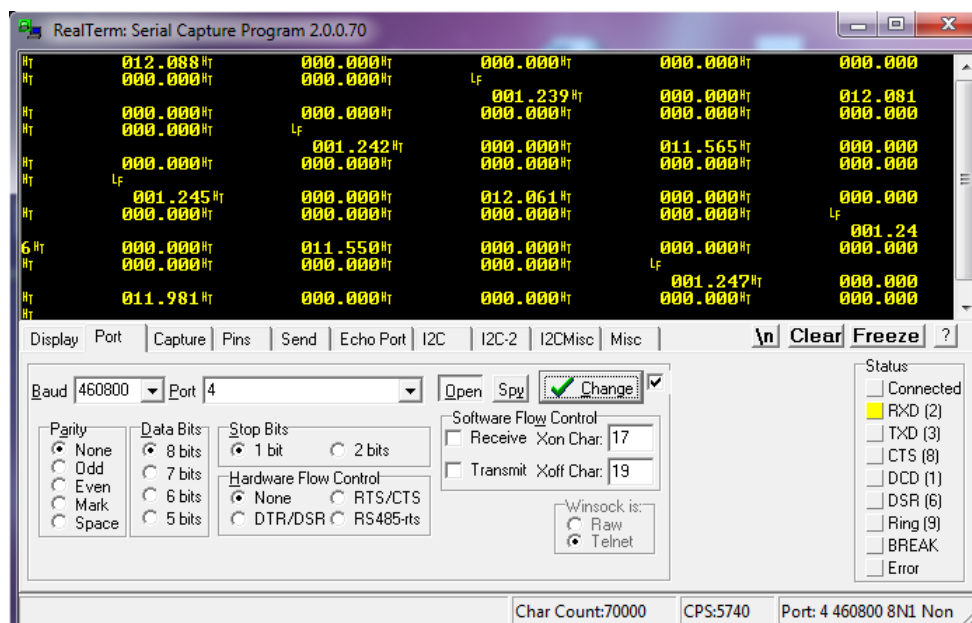


Figura 3.40: *Software* RealTerm, durante captura de dados enviados pelo microcontrolador.

O módulo `CUSTOM_FUNCTIONS` contém a função “`float2BCD`”, que converte uma variável do tipo ponto flutuante (*float*) em dois vetores do tipo caractere (*char*), um contendo a parte inteira da variável, e outro contendo a parte fracionária, com três dígitos cada um, em formato BCD (*Binary Coded Decimal* – Decimal Codificado em Binário). O vetor da parte inteira inclui, ainda, um quarto elemento, contendo o sinal da variável, já em código ASCII. Como se deseja apenas visualizar os resultados, três casas decimais é precisão suficiente. De posse das variáveis já convertidas para BCD, a função “`SendValuesUART`”, do módulo `UART`, converte as variáveis para código ASCII e as envia pela porta serial. Esta função envia três valores separados por tabulações, e após o último valor, pode haver um caractere de controle de nova linha (*line feed*), ou não, segundo a necessidade. O pacote enviado é da forma:

(sinal)XXX.XXX(tabulação)(sinal)YYY.YYY(tabulação)(sinal)ZZZ.ZZZ(tabulação)(nova linha[opcional])

Este pacote conforma com o padrão CSV, pois, de fato, o separador pode ser uma tabulação, não apenas uma vírgula. Um excerto do arquivo de texto é mostrado a seguir, para efeito de ilustração.

```
001.250      001.412      012.173      001.066      000.980      002.153
001.250      001.412      012.173      001.180      001.066      002.158
001.250      -000.362      012.173      001.190      000.150      002.166
001.250      000.524      011.286      001.205      000.613      001.660
001.250      000.524      011.286      001.225      000.561      001.589
001.250      001.412      011.286      001.231      000.996      001.508
001.250      -000.362      012.173      001.237      000.103      001.921
```

Neste exemplo, as três primeiras colunas representam os três eixos do acelerômetro, e as outras três colunas representam os três eixos do giroscópio.

Quando do envio dos valores para o MATLAB, o mesmo procedimento é usado. O MATLAB dispõe de funções de manipulação de arquivos CSV, tornando a leitura dos dados do arquivo de texto simples. No entanto, neste caso, apenas a última linha do arquivo é considerada, e, visto que não é possível forçar a leitura do arquivo em intervalos regulares de tempo, os valores lidos servem apenas para representação visual, sendo de pouca valia para prototipação, por motivos discutidos previamente.

O código fonte do programa interpretado pelo MATLAB para a exibição gráfica da orientação estimada é apresentado no APÊNDICE C.

O módulo periférico UART do microcontrolador MSP430G2553 gera e aceita sinais em nível lógico TTL (0 a 5 V), em protocolo UART, porém o computador possui apenas portas USB. Assim sendo, é necessária a utilização de algum dispositivo conversor. O módulo conversor utilizado é um conversor TTL/RS232 para USB comercial (fabricante desconhecido), que emprega o CI **CP2102**, do fabricante Silicon Labs. A Figura 3.41 mostra uma foto do módulo.



Figura 3.41: Módulo conversor USB.

A fim de controlar os dois servomotores responsáveis pela estabilização vertical, os valores de *pitch* e *roll* calculados pelas equações Equação 3.7 e Equação 3.8 são convertidos em sinais de PWM. O gerador de PWM do microcontrolador é configurado de tal forma que o intervalo [1,2], em milissegundos, necessário para o controle dos servos, é equivalente ao intervalo [2000,4000] para os valores inteiros que determinam o *duty-cycle*. Como os servos só podem ser movimentados no intervalo [90,-90] graus, esta faixa deve ser transformada para a faixa [2000,4000]. A conversão é feita segundo a Equação 3.9.

$$\text{duty} = 2000 + (\text{ângulo} + 90) * 11.111111$$

**Equação 3.9**

Verifica-se que, quando “ângulo = 0”, o resultado da Equação 3.9 é “duty = 2999,9999”, que é aproximado para 3000, o que equivale a 1,5 milissegundos. Quando “ângulo” vale 90 ou -90, os

resultados são “duty = 3999,99998  $\approx$  4000”, e “duty = 2000”, que equivalem, respectivamente, a 1 milissegundo e 2 milissegundos. Como medida de segurança, os valores que “duty” pode receber são forçosamente limitados à faixa [2000,4000], ignorando valores fora desta faixa.

## 4 Resultados e Discussões

### 4.1 Resultados do funcionamento

O funcionamento do projeto foi verificado através da movimentação do sistema e observação da resposta fornecida através do computador e dos servomotores.

- **Acelerômetro versus Acelerômetro + Giroscópio**

A estimação da orientação vertical utilizando apenas o acelerômetro apresentou resultados corretos enquanto o sistema esteve em repouso ou durante movimentos lentos, porém bastante ruidosos, isto é, a estimação oscila em torno do valor correto. Durante movimentos translacionais, no entanto, a estimação é prejudicada pela aceleração linear do conjunto somada à aceleração da gravidade, fazendo com que a inclinação seja momentaneamente incorreta.

Quando a informação do giroscópio é levada em consideração, por outro lado, a estimação fica quase livre de ruídos, e os erros oriundos da aceleração linear do sistema são bastante amortecidos. Pode-se observar na Figura 4.1 o efeito da inclusão do giroscópio no algoritmo: o gráfico superior mostra as componentes em cada eixo (X, Y e Z) do vetor unitário de orientação durante um movimento translacional como na Figura 4.2. Observa-se que o vetor oscila, acompanhando a translação. O gráfico inferior mostra as mesmas componentes do vetor de orientação, porém corrigido pelo giroscópio. Observa-se que a oscilação do vetor é bastante reduzida, em comparação com a oscilação do vetor não corrigido.

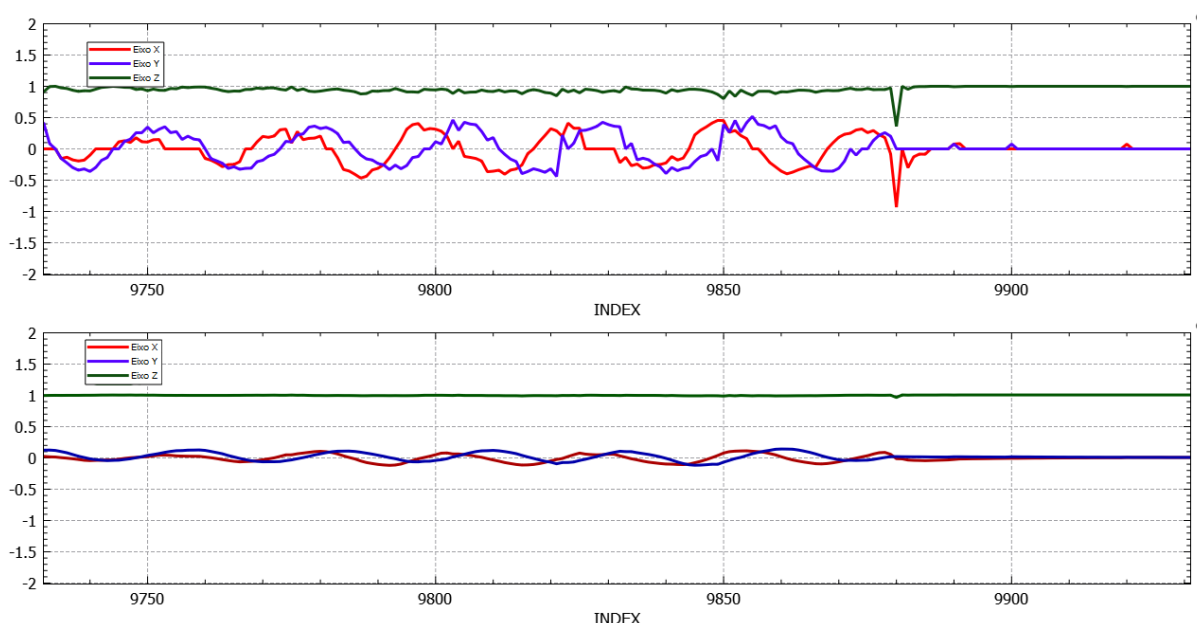


Figura 4.1: Estimação da orientação, durante movimento translacional circular, sem (superior) e com (inferior) o giroscópio.

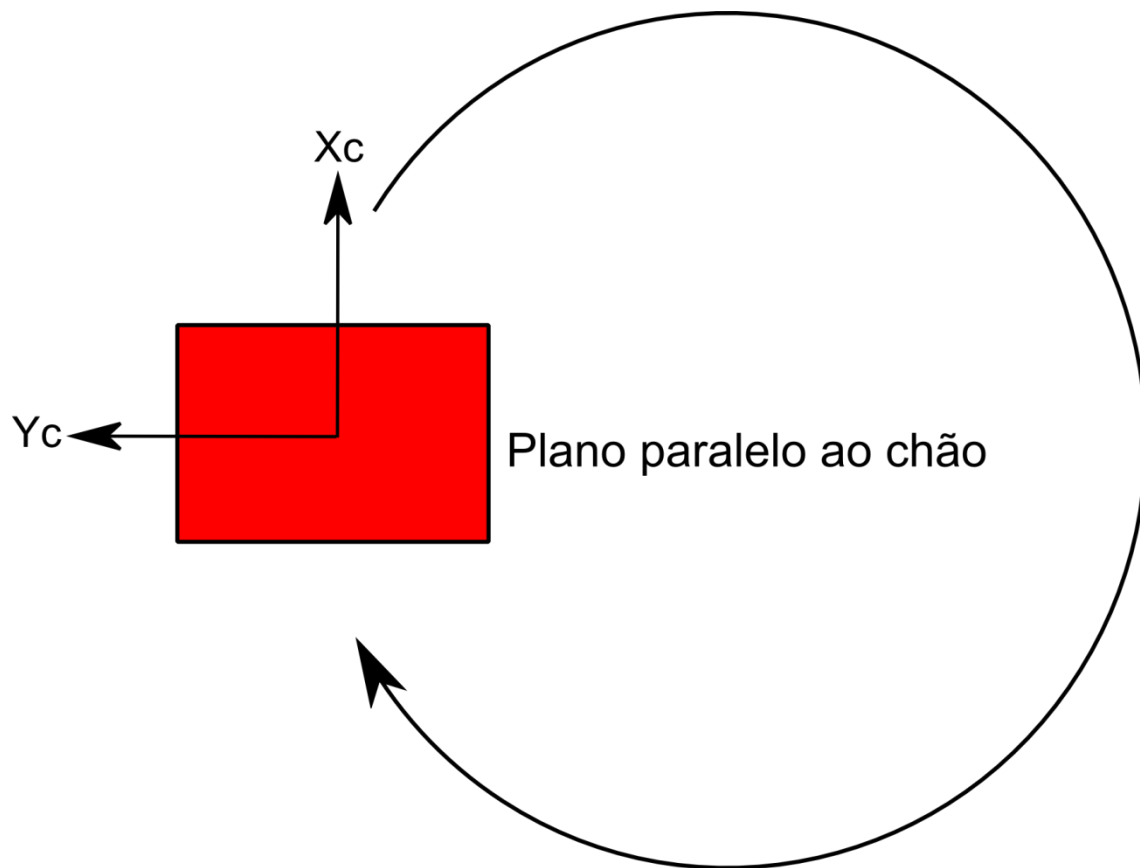


Figura 4.2: Movimento realizado para teste de supressão da aceleração linear.

- **Resposta do algoritmo com visualização gráfica no MATLAB**

Os testes utilizando a representação gráfica dos eixos do sistema obtida através do MATLAB apresentaram os resultados esperados. O eixo vertical (azul) sempre converge para a inclinação correta, em todos os octantes, como pode ser visto nas figuras 4.3 a 4.6.

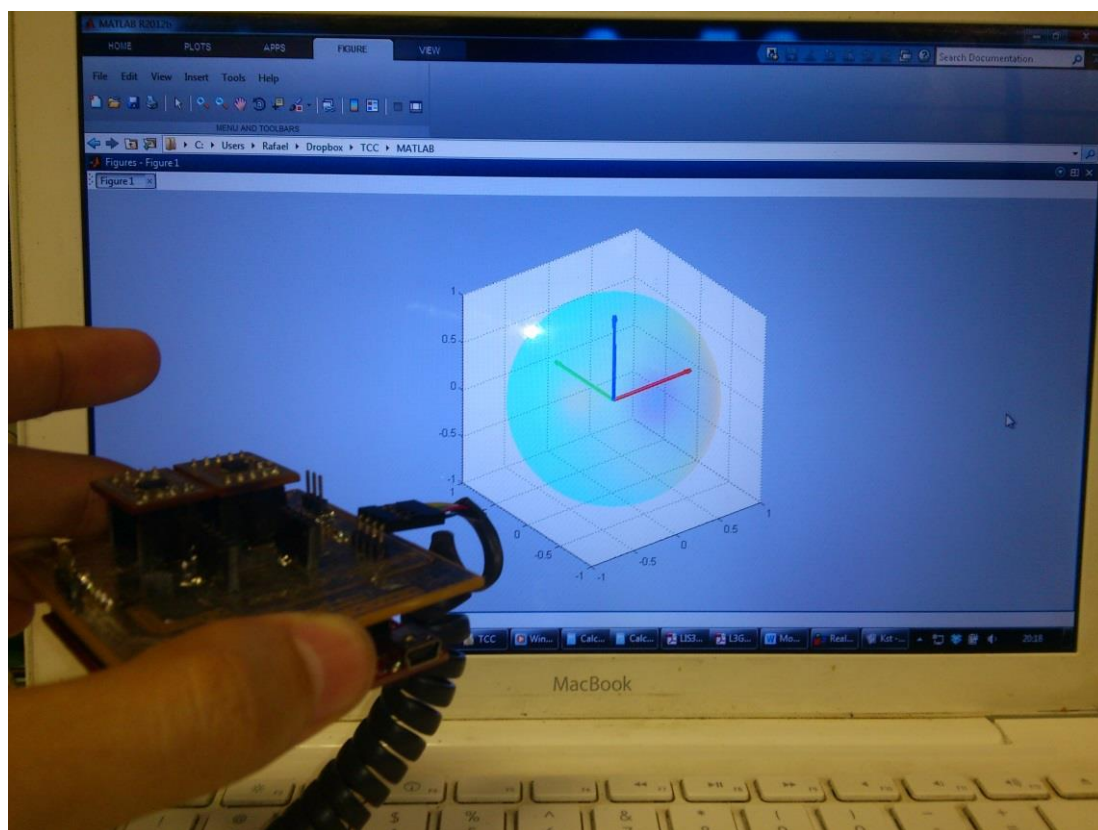


Figura 4.3: Posição vertical positiva.

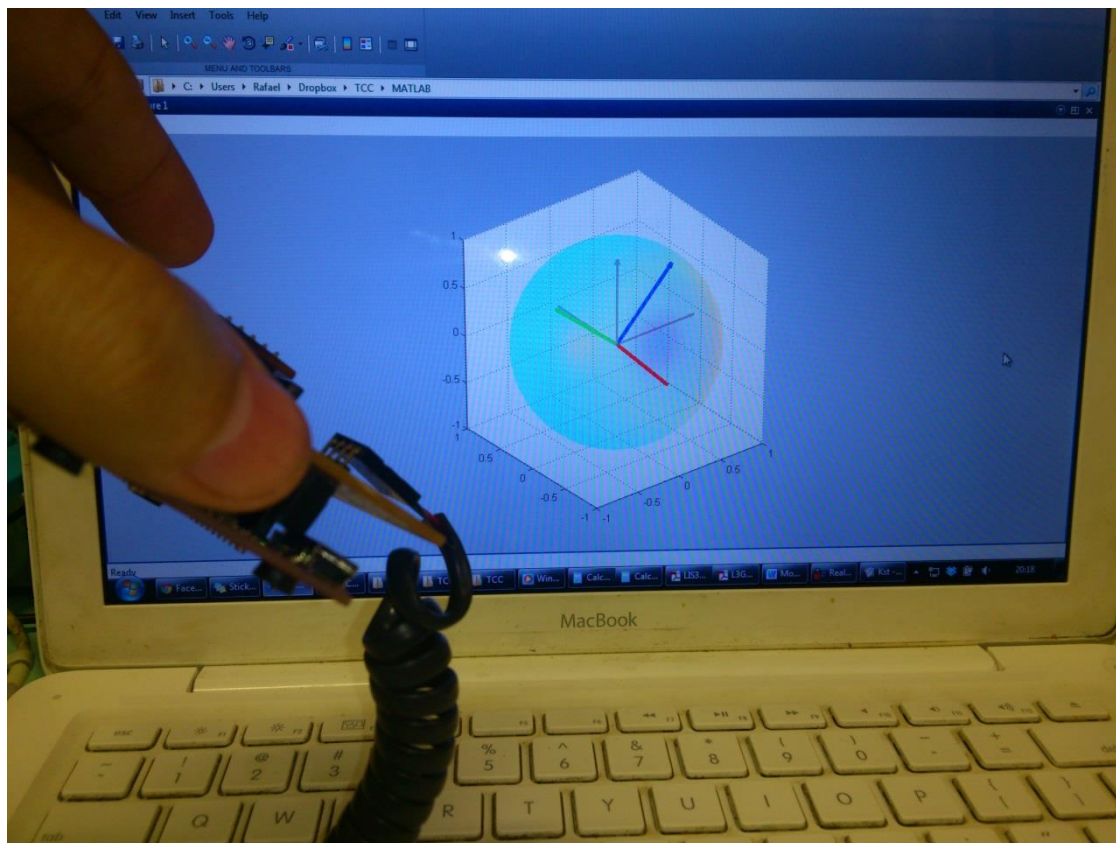


Figura 4.4: Inclinação com *pitch* igual a 45 graus.

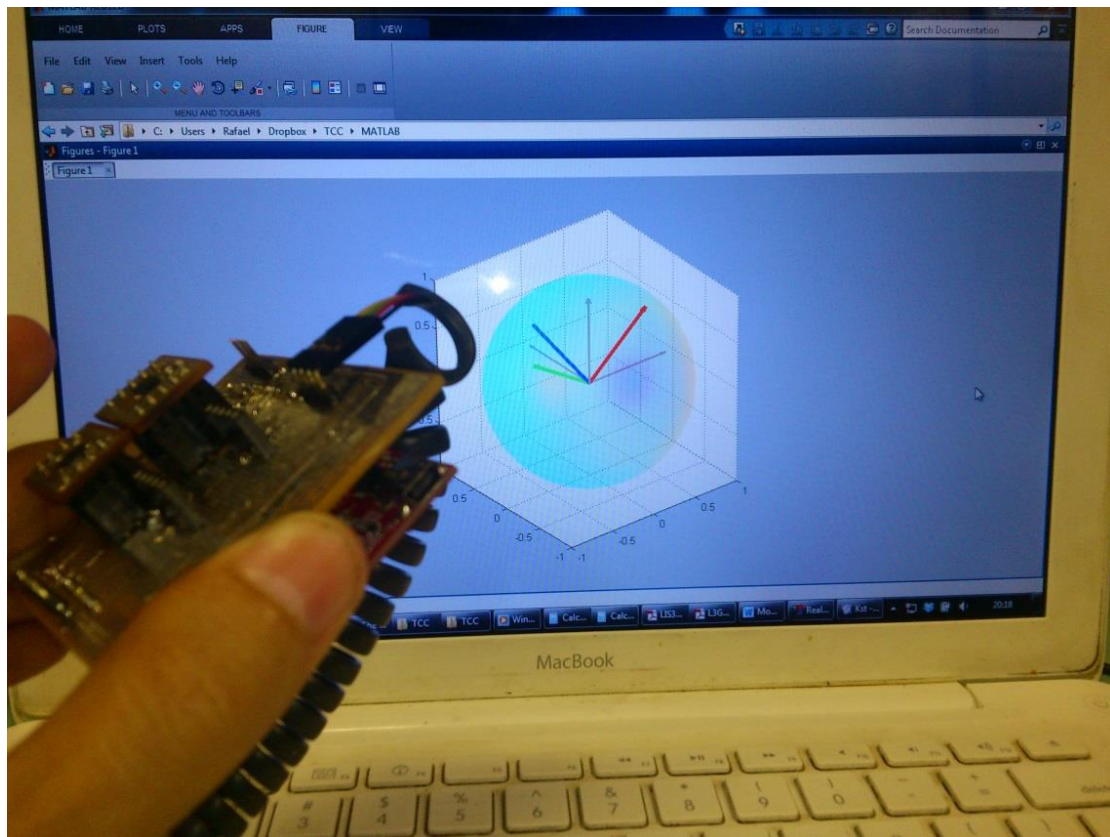


Figura 4.5: Inclinação com *pitch* e *roll* iguais a -45 graus.

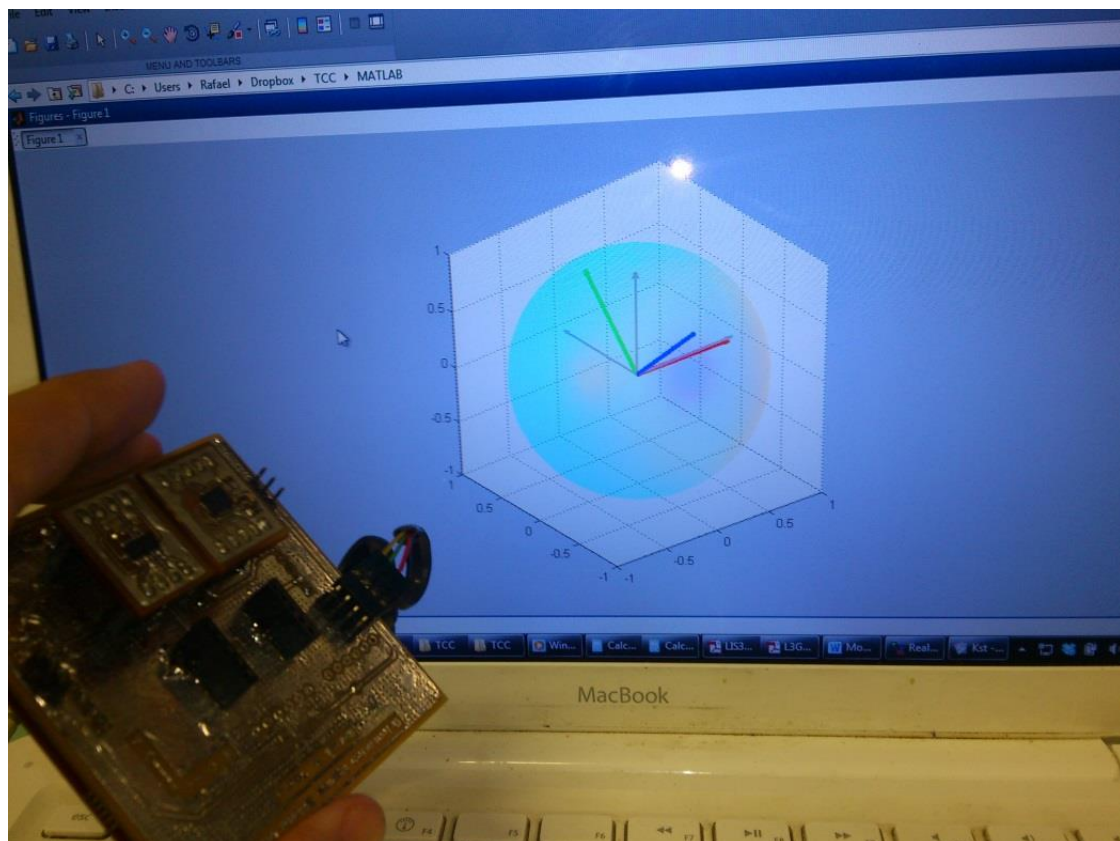


Figura 4.6: Inclinação com *roll* igual a 45 graus.

Mesmo se o sistema for rotacionado muito rapidamente (mais de 180 graus em 20 milissegundos, ou seja, mais de 9000 graus por segundo), o algoritmo sempre proporciona a convergência do vetor de estimação sobre o vetor real da inclinação quando o sistema entrar em repouso. No entanto, neste caso, o vetor pode girar na direção oposta ao giro, pois, se entre uma amostragem e seguinte o vetor real houver rotacionado mais que 180 graus, o menor caminho entre ele e o vetor estimador é no sentido oposto de giro, como pode ser visto na Figura 4.7.

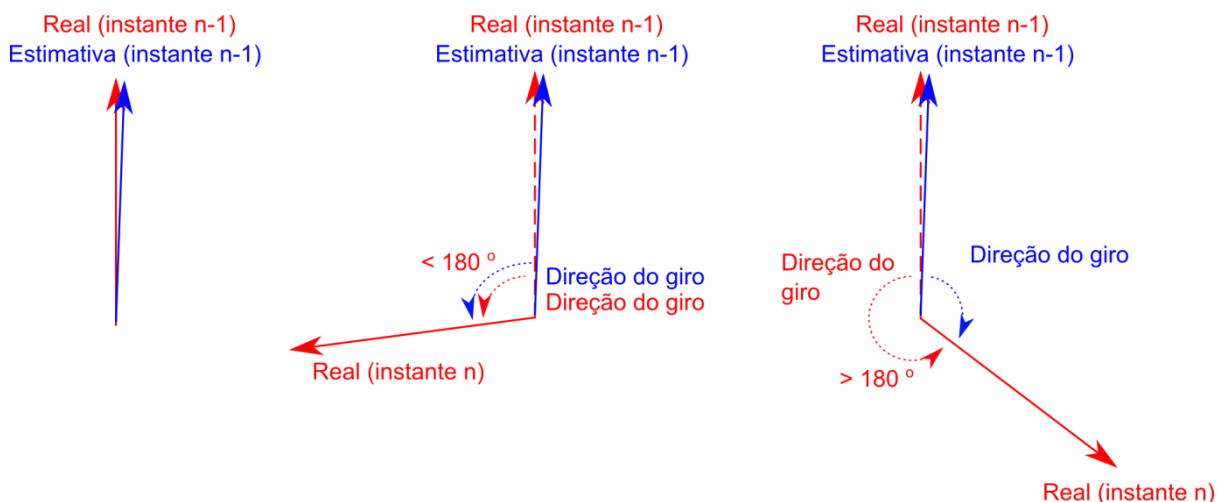


Figura 4.7: Sentidos de giro do vetor estimador.

- **Resposta do algoritmo com visualização pelos servomotores**

A utilização dos servomotores para mostrar a orientação do sistema também apresentou resultados dentro do esperado. O eixo do servomotor livre sempre converge para a posição vertical, desde que nenhuma inclinação seja maior que  $\pm 90$  graus. Caso alguma inclinação seja maior que  $\pm 90$  graus, os servomotores mantêm suas últimas posições angulares (com respeito a si mesmos).

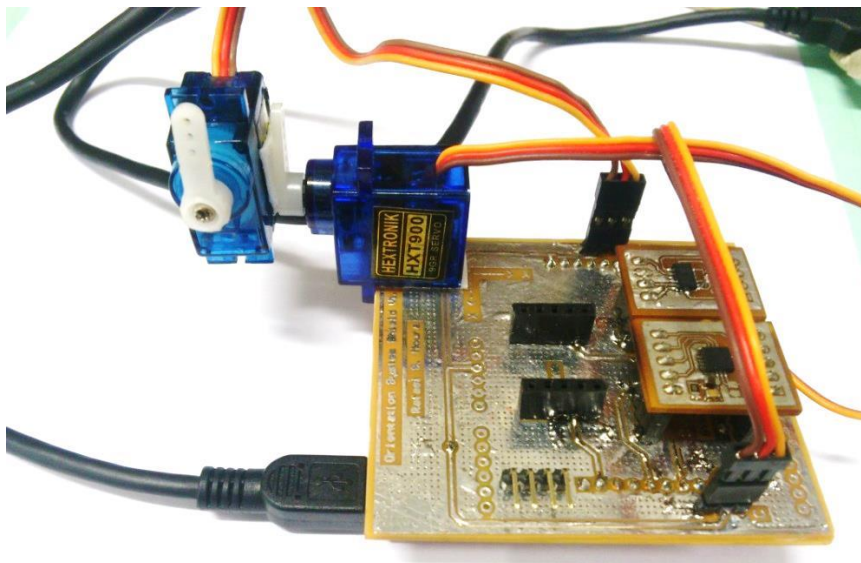


Figura 4.8: Sistema com servomotores em repouso.

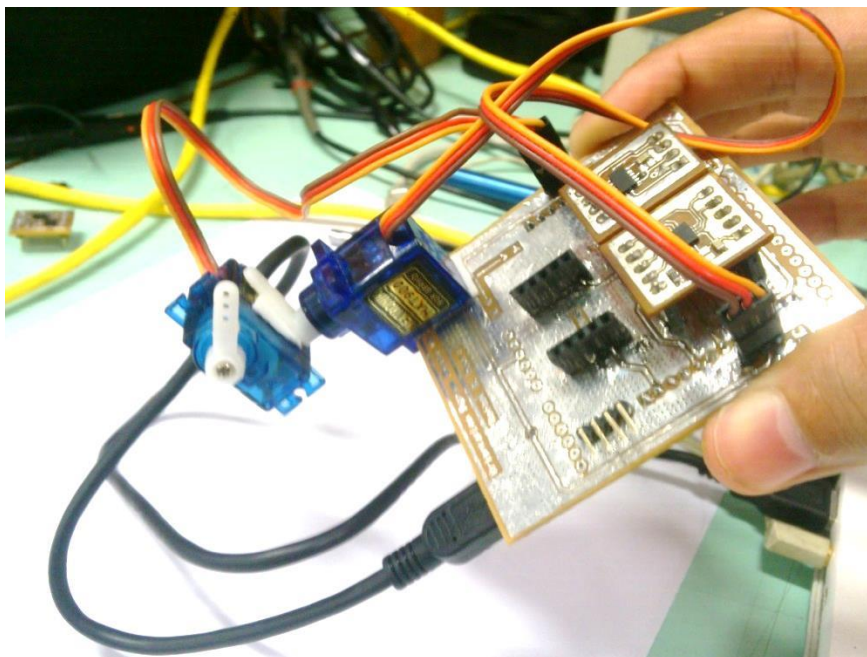


Figura 4.9: Resposta dos servomotores para uma inclinação arbitrária.

## 4.2 Problemas

Durante o desenvolvimento deste projeto foram encontrados diversos obstáculos. Os problemas mais importantes e não solucionados são descritos a seguir, bem como possíveis soluções. As soluções indicadas não foram aplicadas por falta de recursos financeiros e de tempo.

- **SPI de 9 bits**

O controlador embutido na tela de cristal líquido se comunica com outros dispositivos por meio de protocolo SPI com palavras de 9 *bits*. No entanto, o microcontrolador MSP430G2553 não possui a habilidade de geração de palavras de 9 *bits*, sendo limitado a 7 e 8 *bits*. Para contornar este obstáculo, foram criadas rotinas para gerar o protocolo SPI em *software*, isto é, sem usar o periférico dedicado embutido no microcontrolador. À implementação de protocolos de comunicação via *software* é dado o nome *bit banging* [45]. Apesar do sucesso obtido na implementação do protocolo, esta se tornou inviável, pois a taxa máxima de transmissão dos *bits* é proibitivamente baixa ao se utilizar o protocolo controlado por *software*, de modo que o tempo adicional requerido pela comunicação com a tela ultrapassa o período de amostragem dos sensores. Por esta razão, a interface gráfica utilizando a tela foi abandonada. Este problema seria facilmente sanado utilizando-se um microcontrolador com um módulo de comunicação SPI mais flexível, capaz de gerar palavras de 9 *bits*.

- **Frequência de operação do microcontrolador**

Outro obstáculo fortemente limitante para o funcionamento do sistema é a frequência máxima de operação do microcontrolador. Como este é um sistema em tempo real, i.e., um sistema com restrições de tempo, onde determinadas tarefas devem ser realizadas dentro de períodos controlados de tempo, a baixa frequência de operação do microcontrolador limita a quantidade de instruções que podem ser realizadas em tempo hábil. Foi colocado considerável esforço na codificação de funções que fizessem operações em variáveis do tipo ponto fixo, pois operações em ponto fixo são apreciavelmente mais rápidas que operações em ponto flutuante; não obstante, devido à necessidade abundante de operações trigonométricas e de radiciação, foi necessário usar variáveis em ponto flutuante para todas as grandezas fracionárias. A quantidade de operações em ponto flutuante tornou o sistema lento, limitando a capacidade deste de fazer estimações mais exatas. Utilizar-se microcontroladores habilitados a operar em frequências maiores resolveria tais limitações de tempo, permitindo um programa mais elaborado de estimação.

- **Falta de memória RAM do microcontrolador**

Ainda no domínio dos obstáculos gerados pela utilização de variáveis do tipo ponto flutuante, a falta de memória RAM também agiu como fator limitante à complexidade do algoritmo estimador. Variáveis do tipo *float*, nesta família de microcontroladores, ocupam 32 *bits* (4 bytes) de memória. A título de exemplo, o número de amostras passadas consideradas na Equação 3.1 foi escolhido como três, porém inicialmente eram considerados quinze valores passados, o que, por si só, ocupava 360 bytes de RAM ( $[4 \text{ bytes}] * [15 \text{ amostras}] * [6 \text{ eixos}] = 360$ ). À medida que o algoritmo crescia em tamanho e complexidade, deparou-se com o transbordo de memória, i.e., não havia mais espaço para variáveis na memória RAM do microcontrolador, criando a necessidade de racionamento de variáveis. Este obstáculo seria superado com o emprego de um microcontrolador com mais RAM disponível.

- **Lentidão do protocolo UART**

Assim como a implementação do protocolo SPI em *software* causou atrasos no processamento, a comunicação serial pelo módulo UART também diminui o tempo disponível para o algoritmo de estimação dentro do período de amostragem. O *kit* LaunchPad possui um módulo conversor TTL/USB que pode ser usado para fazer a comunicação serial entre o microcontrolador e o computador, porém este é limitado em um *baudrate* de 9.600, taxa esta proibitivamente baixa. Para abrandar o atraso causado por esta comunicação, foi empregado o conversor comercial descrito na seção 3.6.4, o qual permitiu que o *baudrate* fosse aumentado até o máximo valor permitido pelo microcontrolador, que é de 460.800. Apesar do elevado *baudrate* obtido, o tempo perdido durante a comunicação ainda é sensivelmente alto, compondo cerca de 14,0% do período de amostragem ( $\frac{2,8 \text{ ms}}{20,0 \text{ ms}} = 0,14$ ), como pode ser visto nas figuras 4.10 e 4.11. Contribui para este atraso, ainda, o tempo gasto durante a conversão das variáveis enviadas para código ASCII. Além de um microcontrolador capaz de taxas mais altas de transferência, uma possível solução para estes empecilhos seria a utilização de *hardware* externo dedicado à comunicação serial.

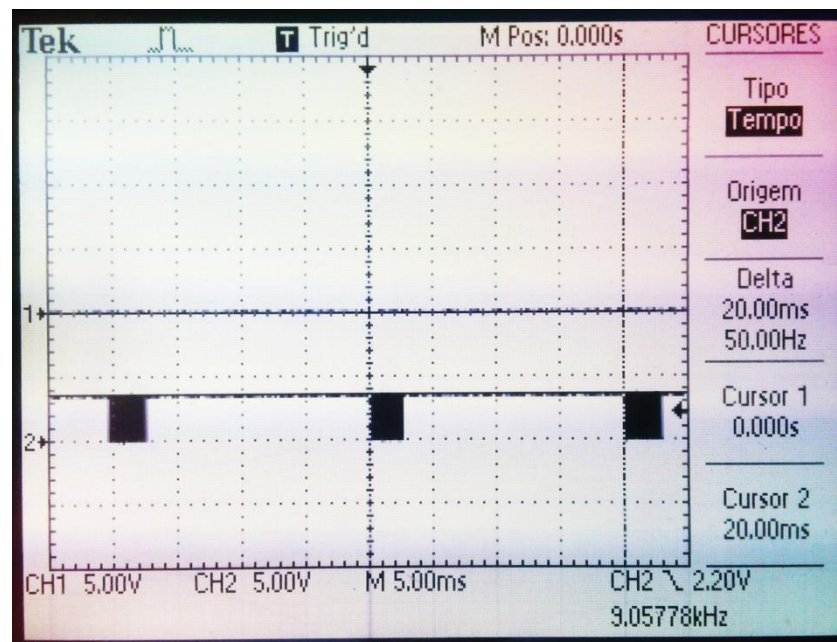


Figura 4.10: Canal 2: Linha de transmissão serial, cursores marcando o período de amostragem.

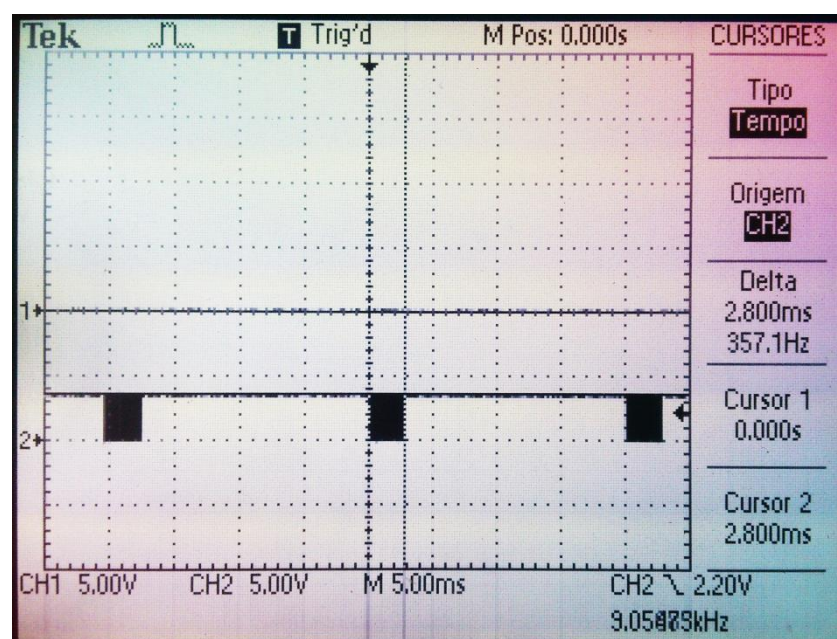


Figura 4.11: Canal 2: Linha de transmissão serial, cursores marcando o tempo gasto pela comunicação.

Os resultados da lentidão do sistema foram a diminuição da frequência de amostragem e da sofisticação do algoritmo de estimação, incluindo a exclusão da correção, por meio do magnetômetro, do desvio acumulativo da rotação em torno do eixo Z no sistema de coordenadas da Terra.

## 5 Conclusões

Pode-se concluir que o trabalho teve sucesso, a despeito das limitações impostas pelo microcontrolador. O objetivo de construir um sistema de orientação inercial foi alcançado. O *hardware* projetado e montado funcionou corretamente, e a comunicação com os sensores foi implementada com sucesso. A estimação da orientação vertical apresentou os resultados esperados, respondendo rapidamente e com pouca presença de ruídos. Devido à não utilização do magnetômetro, o objetivo inicial de estimar a orientação espacial inclusive levando em conta rotações ao redor do eixo Z, e não apenas estimar corretamente a inclinação, não foi alcançado, embora o algoritmo esteja pronto para a inclusão do magnetômetro, sendo necessário apenas desenvolver o algoritmo de correção baseado na referência ao norte magnético da Terra proporcionado pelo magnetômetro.

O projeto, em seu nível de desenvolvimento atual, pode ser prontamente utilizado em diversas aplicações, como estabilização vertical de plataformas/câmeras, controle de estabilização de pêndulo invertido, controle de jogos, e indicadores de inclinação, que podem ser usados em aplicações de navegação.

Os objetivos parciais alcançados foram:

- Integração dos sensores (acelerômetro, giroscópio e magnetômetro) ao sistema;
- Estimação da inclinação vertical;
- Uso da estimação da inclinação vertical para implementar um sistema de estabilização de plataformas;
- Exibição gráfica de vetores em um computador para representar as estimativas de orientação;

O sucesso destes objetivos permite que se conclua que o projeto como um todo teve sucesso, pois são as partes mais importantes do sistema. A comunicação com os sensores é a primeira etapa do projeto, sendo crucial para o sistema; além disso, a partir das rotinas de comunicação desenvolvidas, outros sensores podem ser imediatamente utilizados com o sistema. A estimação da inclinação vertical e seu uso no controle de servomotores permite que o sistema seja utilizado em aplicações de controle de estabilização de câmeras e plataformas, entre outras. Finalmente, a comunicação e exibição gráfica em computadores proporciona a possibilidade de se fazer interfaces com o usuário, tanto para aplicações quanto para depurações e testes do sistema.

Os objetivos parciais não alcançados foram:

- Fusão do magnetômetro e do acelerômetro para implementar uma bússola digital;
- Estimação da rotação do sistema ao redor de todos os eixos utilizando a compensação proporcionada pela bússola.

Estes objetivos, embora de menor importância para o sistema, são componentes de um sistema capaz de estimar corretamente rotações em torno do eixo vertical, o qual permitiria que a atitude do sistema fosse estimada completamente, e não apenas a inclinação vertical. As aplicações para tais sistemas incluem versões mais sofisticadas de jogos, assistentes de navegação e sistemas de controle de orientação de objetos, por exemplo.

O estudo feito neste projeto, como citado na seção 1.1, serve de base para a criação futura de um sistema que estime corretamente, além da orientação do objeto, sua posição no espaço, em relação a um ponto de origem. Os resultados obtidos se mostram promissores, e incentivam o prosseguimento do projeto, utilizando microcontroladores mais poderosos, sensores mais sensíveis e técnicas e algoritmos mais refinados, a fim de alcançar a meta final, que é projetar um sistema de navegação puramente inercial, portátil.

## 5.1 Sugestões para trabalhos futuros

Para trabalhos futuros baseados no projeto apresentado, a principal sugestão é o uso de um microcontrolador com memória RAM de maior capacidade, maior frequência de operação máxima (por exemplo, o microcontrolador MSP430F5171, com 2 kB de memória RAM e frequência máxima de operação de 25 MHz), e módulo SPI capaz de gerar palavras de 9 *bits*, pois assim seria possível realizar os objetivos não alcançados neste trabalho:

- Implementação da comunicação com a tela de cristal líquido apresentada, bem como codificação de um *driver* para fazer desenhos/textos na tela;
- Inclusão do magnetômetro para se obter mais um vetor de referência;
- Projeto de um algoritmo de estimação mais complexo e mais robusto, capaz de calcular com maior precisão e acurácia a orientação do sistema.

Outras sugestões:

- Utilização do protocolo I2C, para eliminar a necessidade de um conversor SPI/I2C;
- Projeto de uma única placa de circuito impresso que acomode o circuito como um todo, incluindo os sensores, conectores e microcontrolador;
- Utilização de bateria para alimentar o sistema, eliminando a necessidade de cabos de alimentação.

## 6 Bibliografia

1. SMART phone note. **Gizfactory**. Disponível em: <<http://gizfactory.com/article/smart-phone-note/>>. Acesso em: 26 outubro 2013.
2. COORDINATE system. **Wikipedia, The Free Encyclopedia**. Disponível em: <[http://en.wikipedia.org/wiki/Coordinate\\_system](http://en.wikipedia.org/wiki/Coordinate_system)>. Acesso em: 29 outubro 2013.
3. SANTOS, F. J. D.; SILVIMAR, F. F. Coordenadas cartesianas. In: BOOKMAN **Geometria Analítica**. [S.l.]: [s.n.], 2009. Cap. 1.
4. SANTOS, F. J. D.; SILVIMAR, F. F. Rotação. In: BOOKMAN **Geometria analítica**. [S.l.]: [s.n.], 2009. Cap. 5.
5. SANTOS, F. J. D.; SILVIMAR, F. F. Produto Vetorial. In: BOOKMAN **Geometria analítica**. [S.l.]: [s.n.], 2009. Cap. 9.
6. INERTIAL measurement unit. **Wikipedia, The Free Encyclopedia**. Disponível em: <[http://en.wikipedia.org/wiki/Inertial\\_measurement\\_unit](http://en.wikipedia.org/wiki/Inertial_measurement_unit)>. Acesso em: 26 outubro 2013.
7. SICILIANO, B.; KHATIB, O. **Springer Handbook of Robotics**. [S.l.]: [s.n.], 2008.
8. MODELS. **Segway**. Disponível em: <<http://www.segway.com/individual/models/>>. Acesso em: 29 outubro 2013.
9. MOLHABILIDADE. **Wikipedia, A Enciclopédia Livre**. Disponível em: <<http://pt.wikipedia.org/wiki/Molhabilidade>>. Acesso em: 14 out. 2013.
10. MICROELECTROMECHANICAL systems. **Wikipedia, The Free Encyclopedia**. Disponível em: <[http://en.wikipedia.org/wiki/Microelectromechanical\\_systems](http://en.wikipedia.org/wiki/Microelectromechanical_systems)>. Acesso em: 14 out. 2013.
11. MEMS. **Electronics and instrumentation engineering**. Disponível em: <<http://einstrumentation.blogspot.com.br/2012/11/mems.html>>. Acesso em: 26 outubro 2013.
12. **Physics to go**. Disponível em: <<http://www.compadre.org/informal/index.cfm?Issue=5>>. Acesso em: 26 outubro 2013.
13. MEMS Image Gallery. **MEMX**. Disponível em: <[http://www.memx.com/image\\_gallery.htm](http://www.memx.com/image_gallery.htm)>. Acesso em: 26 outubro 2013.
14. SONIC Nirvana: MEMS Accelerometers as Acoustic Pickups in Musical Instruments. **Sensors Magazine**. Disponível em: <<http://www.sensormag.com/sensors/acceleration-vibration/sonic-nirvana-mems-accelerometers-acoustic-pickups-musical-i-5852>>. Acesso em: 26 outubro 2013.
15. MEMS Motion Sensors: The Technology Behind the Technology. **Low-Power Design**. Disponível em: <[http://low-powerdesign.com/article\\_mems\\_032711.html](http://low-powerdesign.com/article_mems_032711.html)>. Acesso em: 26 outubro 2013.
16. GYROSCOPE. **Wikipedia, The Free Encyclopedia**. Disponível em:

- <<http://en.wikipedia.org/wiki/Gyroscope>>. Acesso em: 26 outubro 2013.
17. **Northwestern University.** Disponível em: <<http://clifton.mech.northwestern.edu/~me381/project.html>>. Acesso em: 26 outubro 2013.
  18. IMPLEMENTING your MCU-based system's serial UART in software. **Embedded.** Disponível em: <<http://www.embedded.com/design/other/4025995/Implementing-your-MCU-based-system-s-serial-UART-in-software>>. Acesso em: 29 outubro 2013.
  19. UNIVERSAL Asynchronous Receiver/Transmitter. **Wikipedia, The Free Encyclopedia.** Disponível em: <[http://en.wikipedia.org/wiki/Universal\\_asynchronous\\_receiver/transmitter](http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter)>. Acesso em: 29 outubro 2013.
  20. SRINATH, N. K. **8085 Microprocessor Programming and Interfacing.** [S.l.]: [s.n.], 2005.
  21. FRACAROLLI, J. P. V.; FRACAROLLI, J. P. V. **Implementação do controle remoto de uma miniatura de carro operado via computador utilizando comunicação wireless.** São Carlos: [s.n.], 2012.
  22. STEINER, C. **8051/8052 Microcontroller.** [S.l.]: Universal Publishers, 2005.
  23. SERIAL Peripheral Interface Bus. **Wikipedia, The Free Encyclopedia.** Disponível em: <[http://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface\\_Bus](http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)>. Acesso em: 26 outubro 2013.
  24. SERVOMOTOR. **Wikipedia, The Free Encyclopedia.** Disponível em: <<http://en.wikipedia.org/wiki/Servomotor>>. Acesso em: 29 outubro 2013.
  25. HXT900. **Solarbotics.** Disponível em: <<https://solarbotics.com/product/25500/>>. Acesso em: 29 outubro 2013.
  26. HOME: Hextronik. **Hextronik.** Disponível em: <<http://www.hextronik.com/>>. Acesso em: 29 outubro 2013.
  27. BAKSHI, U. A.; BAKSHI, V. U. **Electrical Circuits and Machines.** [S.l.]: Technical Publications Pune, 2009.
  28. MOTOR Speed Control. **Robot Platform.** Disponível em: <[http://www.robotplatform.com/knowledge/motion\\_control/motor\\_speed\\_control.html](http://www.robotplatform.com/knowledge/motion_control/motor_speed_control.html)>. Acesso em: 29 outubro 2013.
  29. DOWNLOAD CCS. **Texas Instruments.** Disponível em: <[http://processors.wiki.ti.com/index.php/Download\\_CCS](http://processors.wiki.ti.com/index.php/Download_CCS)>. Acesso em: 26 outubro 2013.
  30. HOME: Altium. **Altium.** Disponível em: <<http://www.altium.com/>>. Acesso em: 27 outubro 2013.
  31. HOME: RealTerm. **SourceForge.** Disponível em: <<http://realterm.sourceforge.net/>>. Acesso em: 27 outubro 2013.

32. MATLAB. **MathWorks**. Disponível em: <<http://www.mathworks.com/products/matlab/>>. Acesso em: 27 outubro 2013.
33. DOWNLOAD Kst. **Kst**. Disponível em: <<http://kst-plot.kde.org/download/>>. Acesso em: 27 outubro 2013.
34. HOME: SketchUp. **SketchUp**. Disponível em: <<http://www.sketchup.com/>>. Acesso em: 27 outubro 2013.
35. MSP430 LaunchPad Value Line Development kit. **Texas Instruments**. Disponível em: <<http://www.ti.com/tool/msp-exp430g2>>. Acesso em: 26 outubro 2013.
36. ACCELEROMETERS. **STMicroelectronics**. Disponível em: <[http://www.st.com/web/en/catalog/sense\\_power/FM89/SC444](http://www.st.com/web/en/catalog/sense_power/FM89/SC444)>. Acesso em: 01 dezembro 2013.
37. GYROSCOPES. **STMicroelectronics**. Disponível em: <[http://www.st.com/web/en/catalog/sense\\_power/FM89/SC1288](http://www.st.com/web/en/catalog/sense_power/FM89/SC1288)>. Acesso em: 01 dezembro 2013.
38. E-COMPASSES. **STMicroelectronics**. Disponível em: <[http://www.st.com/web/en/catalog/sense\\_power/FM89/SC1449](http://www.st.com/web/en/catalog/sense_power/FM89/SC1449)>. Acesso em: 01 dezembro 2013.
39. **STMicroelectronics**. Disponível em: <<http://www.st.com/web/en/home.html>>. Acesso em: 26 outubro 2013.
40. LIS302DLMEMS motion sensor 3-axis  $\pm 2g/\pm 8g$  smart digital output 'piccolo' accelerometer. **STMicroelectronics**. Disponível em: <[http://www.st.com/web/catalog/sense\\_power/FM89/SC444/PF152913](http://www.st.com/web/catalog/sense_power/FM89/SC444/PF152913)>. Acesso em: 26 outubro 2013.
41. L3G4200DMEMS motion sensor: three-axis digital output gyroscope. **STMicroelectronics**. Disponível em: <[http://www.st.com/web/catalog/sense\\_power/FM89/SC1288/PF250373](http://www.st.com/web/catalog/sense_power/FM89/SC1288/PF250373)>. Acesso em: 26 outubro 2013.
42. LSM303DLHC Ultra compact high performance e-compass: 3D accelerometer and 3D magnetometer module. **STMicroelectronics**. Disponível em: <[http://www.st.com/web/catalog/sense\\_power/FM89/SC1449/PF251940](http://www.st.com/web/catalog/sense_power/FM89/SC1449/PF251940)>. Acesso em: 26 outubro 2013.
43. **DigiKey**. Disponível em: <<http://media.digikey.com/Renders/STMicro%20Renders/14-LGA%20PKG.jpg>>. Acesso em: 26 outubro 2013.

44. **Sparkfun Electronics**. Disponível em: <<https://www.sparkfun.com/>>. Acesso em: 26 outubro 2013.
45. MATHIVANAN, N. **PC-Based Instrumentation: Concepts and Practice**. [S.l.]: Prentice-Hall, 2007.
46. KHANDPUR, R. S. **Printed Circuit Boards - Design, Fabrication, Assembly and Testing**. [S.l.]: Tata McGraw-Hill, 2005.
47. BOSSHART, W. C. **Printed Circuit Boards - Design and Technology**. [S.l.]: Tata McGraw-Hill, 1983.

## **APÊNDICE A - Método de fabricação das placas de circuito impresso**



## Introdução

Há diversos métodos de fabricação de placas de circuito impresso (PCIs) [46] [47]. Entre eles, estão:

- Termo transferência;
- Fotolitográfico;
- Fresamento automático com uso de CNCs (*Computer Numerical Control* – Controle numérico computadorizado).

Um dos métodos mais utilizados por iniciantes é o da termo transferência, onde a máscara do circuito é impressa em um papel especial, utilizando uma impressora a *laser*, e o *tonner* é transferido para a PCI por meio de prensagem a quente. No entanto, este método apresenta resultados pouco consistentes, isto é, sua reprodutibilidade é baixa, resultando em falhas nas trilhas frequentemente, mesmo ao se fazer um trabalho esmeroso. Além disso, este método não alcança resolução suficiente para trilhas muito estreitas (recomenda-se trilhas de pelo menos 16 milésimos de polegada). Outras desvantagens deste método incluem o grande consumo de energia elétrica para esquentar a prensa, e o risco de danos à placa devido ao calor.

Os sensores utilizados neste trabalho são fornecidos em encapsulamentos do tipo LGA, com terminais de tamanho muito reduzido, e na parte inferior do *chip*. Pelo tamanho destes terminais, é inviável o emprego do método da termo transferência, pois são requeridas trilhas muito estreitas (foram utilizadas trilhas de 10 milésimos de polegada). Assim, o método fotolitográfico foi escolhido para a fabricação das PCIs.

No método fotolitográfico, uma camada de material sensível à luz ultravioleta é aplicada à chapa de cobre, e, por cima desta camada, é posicionada uma máscara que deixa passar luz apenas em certas regiões. O material sensível à luz UV pode ser de dois tipos: **positivo** e **negativo**.

Num material positivo, as regiões expostas à luz UV sofrem uma reação que as torna solúveis em certos solventes, e as regiões que não foram expostas continuam resistentes ao solvente. Num material negativo, o oposto acontece, isto é, as regiões expostas sofrem polimerização, se tornando insolúveis, e as regiões cobertas continuam solúveis. A máscara é chamada “positiva” se as regiões o que reterão o cobre são impressas, e é chamada “negativa” se as regiões cujo cobre será retirado são impressas.

Para a fabricação das PCIs deste trabalho foram utilizados dois materiais fotossensíveis. Um deles é um *spray* do tipo positivo, comumente conhecido como *photoresist*. O outro material é um

filme fotossensível negativo, conhecido como *dryfilm*. O uso de *dryfilm* se apresentou muito superior, pois é facilmente aplicável sobre a chapa de cobre, enquanto que a aplicação do *spray* deve ser feita num ambiente isento de poeira, e é extremamente difícil se fazer uma aplicação manual uniforme e com espessura adequada (dois fatores determinantes para o sucesso do procedimento). Para o *spray*, o solvente utilizado é hidróxido de sódio (NaOH, conhecido como soda cáustica), e para o *dryfilm* o solvente utilizado é carbonato de sódio (Na<sub>2</sub>CO<sub>3</sub>, conhecido como barrilha). O material das placas é uma chapa de fenolite recoberta com cobre.



Figura A.1: Placa de circuito com *dryfilm* aplicado.

Este método proporciona resultados muito superiores ao método da termo transferência, tanto em qualidade como em largura das trilhas (é possível se obter trilhas de 2 milésimos de polegada facilmente com este método), apresentando taxa de falhas muito menor. O fator limitante na qualidade de uma PCI fabricada por este método é, na maioria das vezes, a qualidade da impressora utilizada na impressão da máscara.

## Procedimento de fabricação

Os métodos da transferência térmica e fotolitográfico seguem as etapas mostradas no fluxograma da Figura A.2.

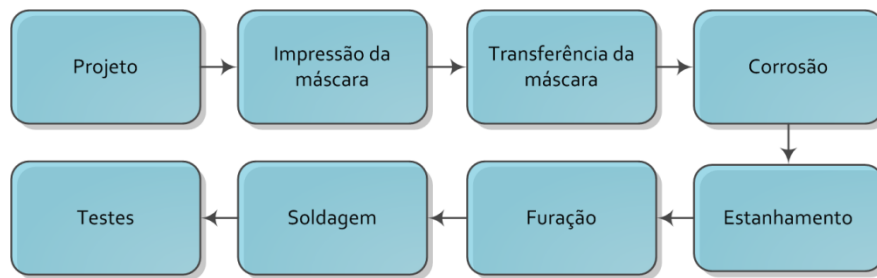


Figura A.2: Procedimento de fabricação de PCIs.

Explicação das etapas:

- **Projeto**

O primeiro passo é o projeto do esquema elétrico do circuito. Uma vez desenvolvido o circuito desejado, é feito o desenho do *layout* da PCI, determinando a disposição dos componentes, dimensões da placa, furação de fixação, roteamento das trilhas, e, eventualmente, colocação de textos e desenhos indicativos, entre outros detalhes. Neste trabalho foram desenhados, inclusive, os modelos mecânicos dos sensores (o desenho com a localização e tamanho dos terminais e do encapsulamento), pois não foi encontrada nenhuma biblioteca pronta contendo estes modelos.

- **Impressão da máscara**

De posse do projeto da PCI, é impressa a máscara que será aplicada à chapa de cobre. Esta máscara tem a função de determinar as regiões de cobre que serão retiradas, e as regiões que reterão o cobre. Na termo transferência, a máscara (positiva) é impressa a *laser* em um papel especial do tipo “*glossy*”, semelhante ao papel utilizado em fotografias. No método fotolitográfico, a máscara (positiva ou negativa) é impressa em papel de transparência, utilizado em retroprojetores. Neste método, a impressão pode ser feita a *laser* ou a tinta, desde que a impressora tenha qualidade suficiente para efetuar uma impressão sem falhas e suficientemente opaca.

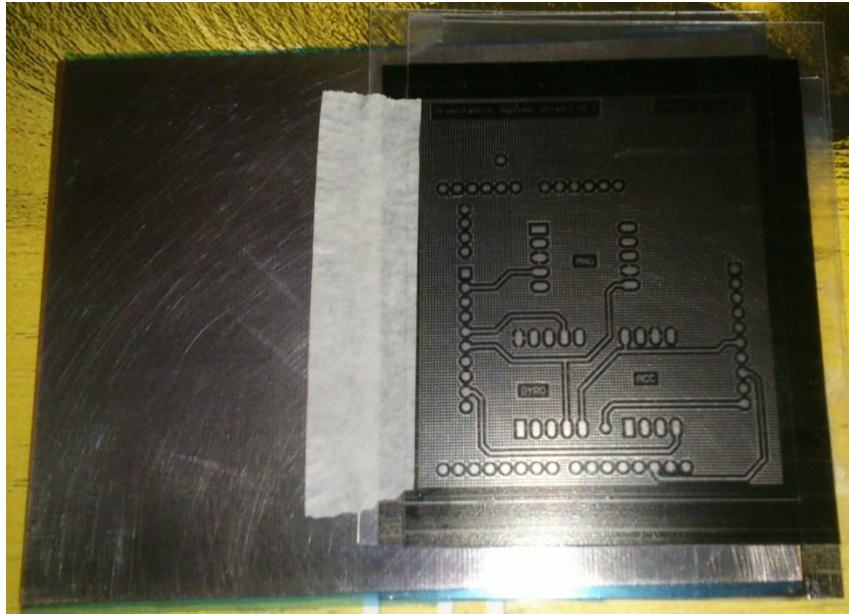


Figura A.3: Máscara para o *shield* colocada sobre a placa.

- **Transferência da máscara**

No método térmico, a máscara é transferida fisicamente para a chapa de cobre, cobrindo com *tonner* as regiões que reterão o cobre. No método com luz UV, a máscara é posicionada sobre a chapa de cobre já coberta com o material fotossensível, e o conjunto é submetido a uma exposição de luz ultravioleta. O tempo de exposição depende de vários fatores, incluindo a distância, intensidade e uniformidade da fonte luminosa. Neste trabalho a exposição ideal encontrada foi de 25 minutos. Finda a exposição, a placa é levada a um banho na solução reveladora, que dissolverá as regiões solúveis do material fotossensível, restando apenas as regiões que devem cobrir o cobre.

Na Figura A.4 é mostrada a PCI recoberta com *dryfilm* durante a exposição a UV.

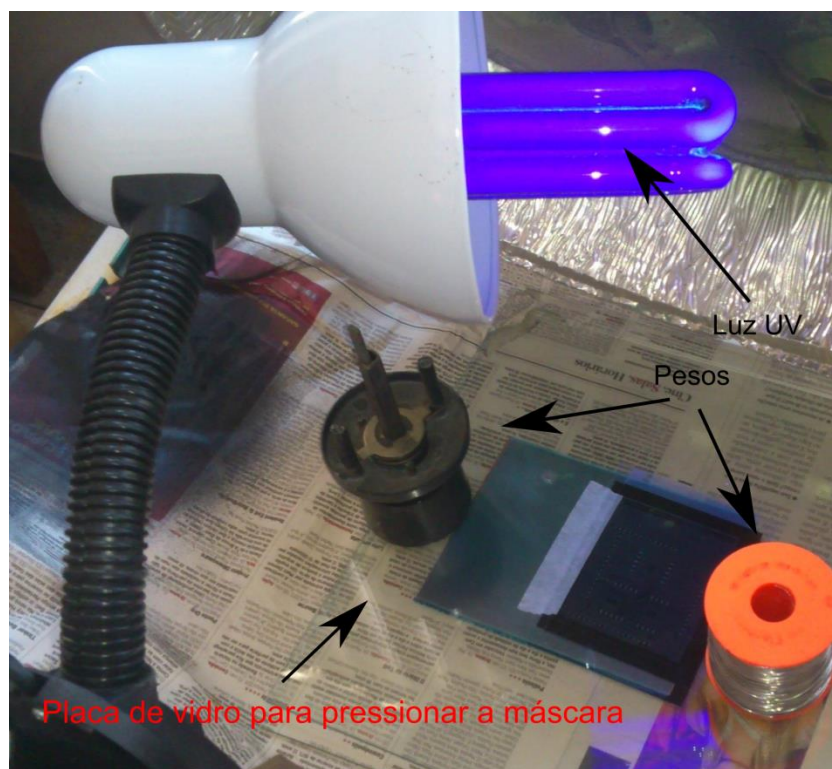


Figura A.4: Exposição a luz ultravioleta.

Na Figura A.5 pode ser vista uma placa de cobre com várias *breakouts* após a exposição a UV. Note que as regiões com azul mais escuro são as regiões polimerizadas, que não serão retiradas durante a revelação. Esta placa foi utilizada durante os testes para determinação do tempo de exposição ideal para o *dryfilm*, utilizando máscaras positivas, as quais seriam inapropriadas para a fabricação da placa, visto que, para o *dryfilm*, as máscaras devem ser negativas.

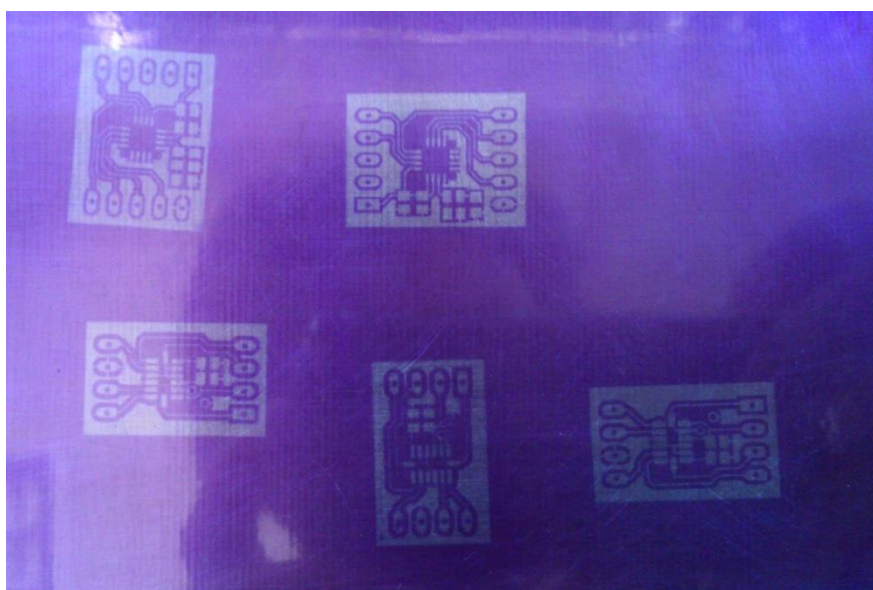


Figura A.5: *Breakouts* do acelerômetro e giroscópio após exposição UV.

Na Figura A.6 pode ser observada a placa do *shield* após revelação em solução de carbonato de sódio.

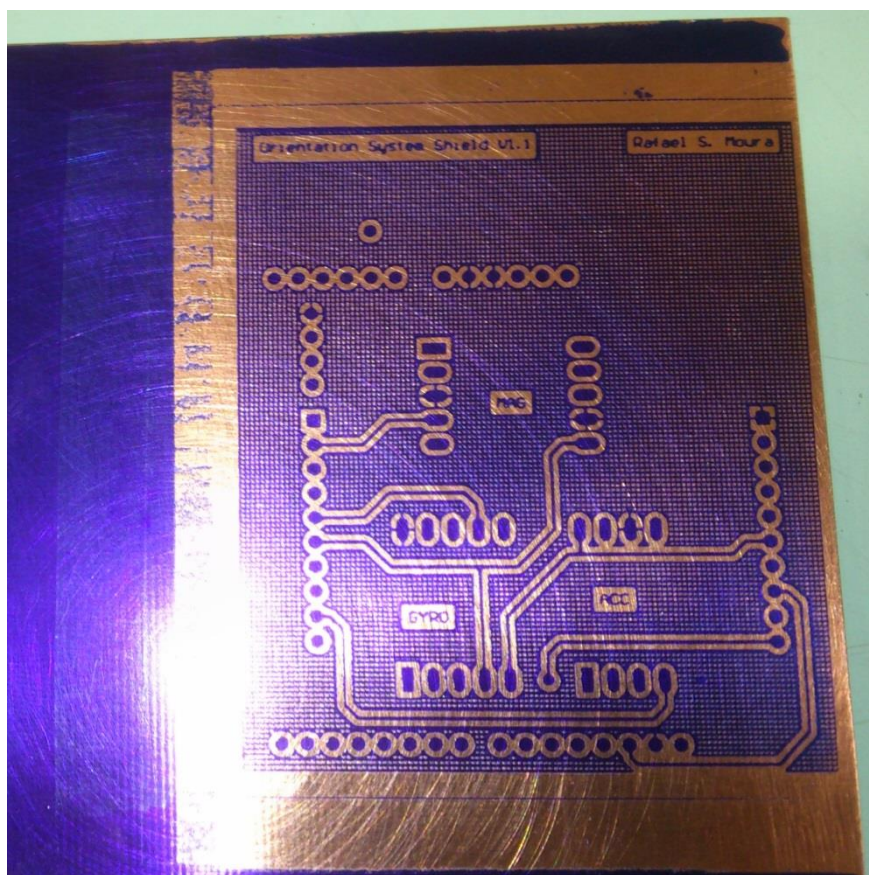


Figura A.6: Placa do *shield* após revelação.

Observação: É de suma importância que a placa esteja limpa antes da aplicação do material fotossensível e antes da corrosão, principalmente isenta de gorduras. Por esta razão, o uso de luvas durante todo o procedimento é recomendado.

- **Corrosão**

Completa a etapa de transferência, a placa de circuito impresso é levada a um banho em solução corrosiva, para retirar o cobre desnecessário. A solução aquosa utilizada neste trabalho é preparada com cloreto de ferro (III) ( $\text{FeCl}_3$ , conhecido popularmente como percloroeto de ferro). Na indústria é utilizada uma solução de ácido clorídrico ( $\text{NaCl}$ , conhecido como ácido muriático), porém a solução é altamente corrosiva, e libera grande quantidade de fumos tóxicos. Por razões de segurança, o percloroeto de ferro é empregado. Após a corrosão, a máscara é retirada mecanicamente (abrasão) ou quimicamente (aplicação de solvente).

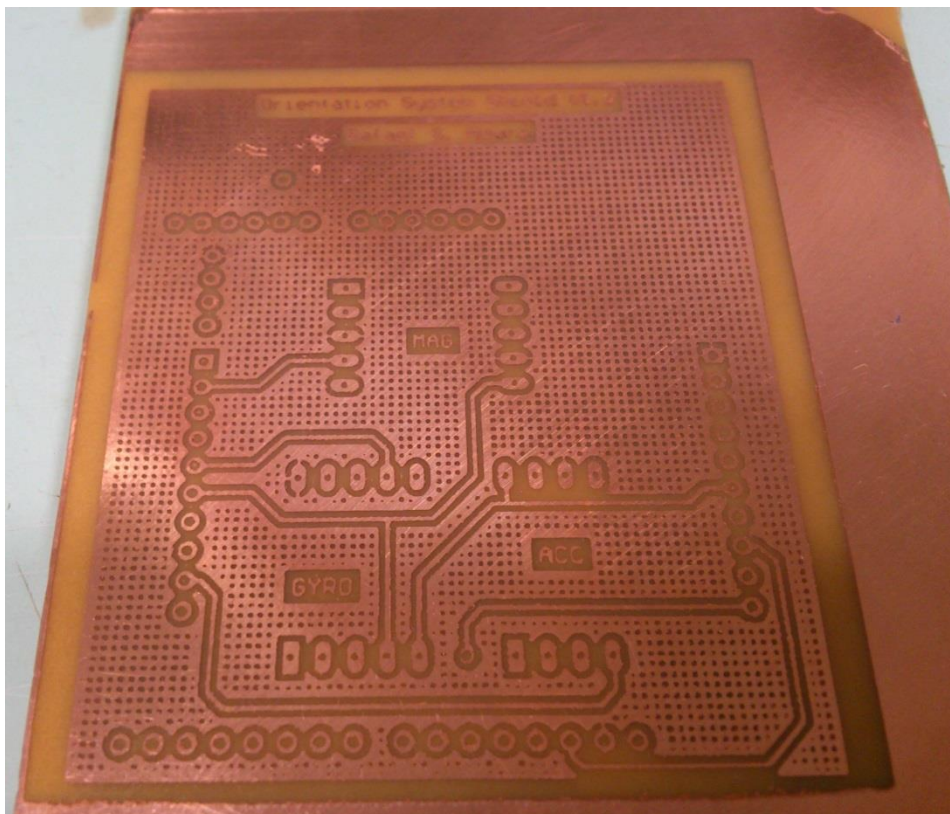


Figura A.7: *Shield* após corrosão.

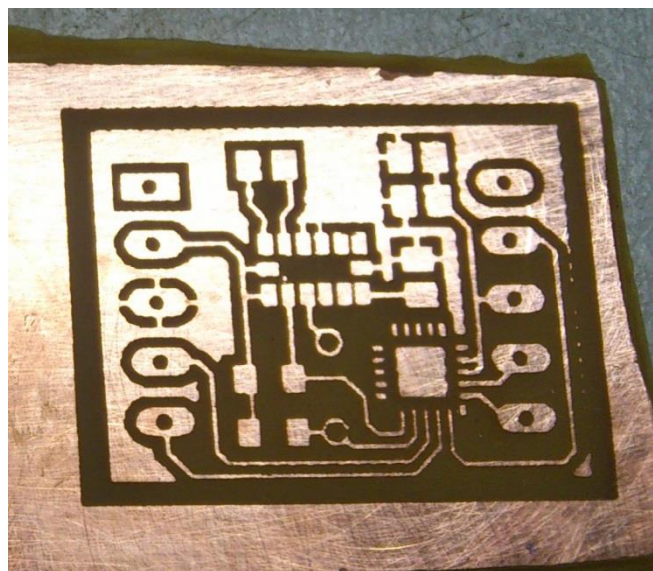


Figura A.8: *Breakout* do magnetômetro após corrosão.

- **Estanhamento**

Tendo em mãos a placa já corroída, é indicada a aplicação de algum tipo de camada protetora. Na indústria são utilizadas tintas resinosas, que são responsáveis pela cor característica (geralmente verde, vermelha ou azul) das PCBs comerciais. Uma alternativa menos dispendiosa e mais rápida é o

“estanhamento” das trilhas, isto é, a aplicação de solda sobre todo o cobre. Isto impede a oxidação do cobre, e tem a vantagem de melhorar a condutividade das trilhas. O padrão quadriculado empregado na grande região de terra presente na PCI do *shield* serve para facilitar o processo de estanhamento, pois um grande plano preenchido de cobre transmite o calor inserido pelo ferro de solda muito rapidamente, dificultando o derretimento da solda.

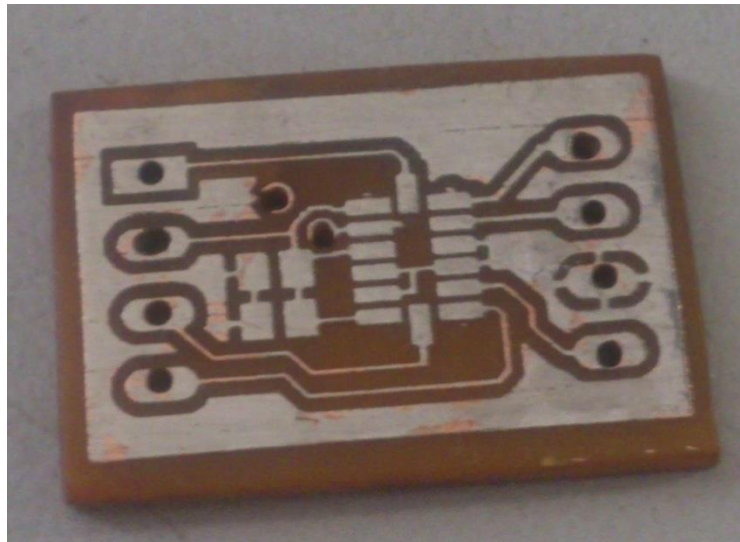


Figura A.9: *Breakout* do acelerômetro estanhada.

- **Furação**

Para se colocar os componentes *through-hole* na PCI, são feitos os furos utilizando uma furadeira de bancada. Geralmente é nesta etapa, também, em que o corte da placa para esta ficar com o formato final é realizado.

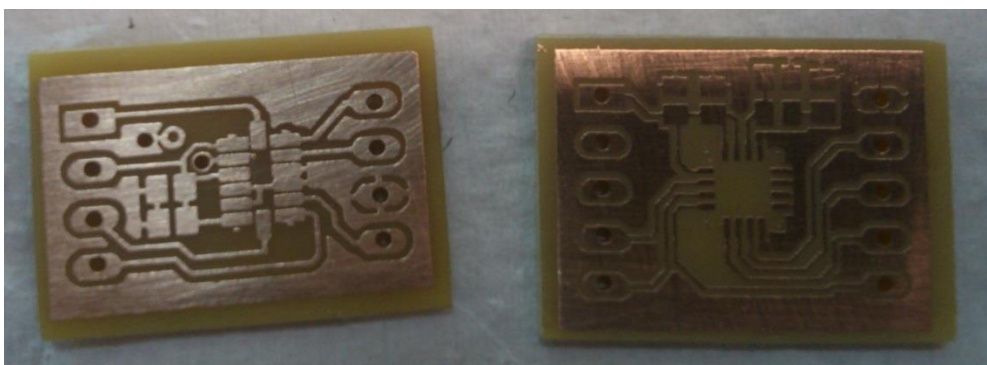


Figura A.10: *Breakouts* do acelerômetro e do giroscópio após furação.

- **Soldagem**

A etapa final da fabricação das PCIs é a soldagem dos componentes. Basicamente há dois tipos de componentes: *through-hole* e SMD (*surface mount device* – dispositivo de montagem superficial). Vários encapsulamentos de componentes SMD podem ser soldados manualmente, mas há tipos que só podem ser soldados utilizando máquinas ou equipamentos especiais. Industrialmente, há vários métodos de soldagem, incluindo soldagem manual com ferros de solda, e soldagem por **refluxo**. A soldagem por refluxo é assim chamada pois a PCI, após receber pequenas quantidades de solda em todos seus terminais e ter os componentes colocados em seus respectivos lugares, é colocada em um forno, e aquecida até que a solda nos terminais entre em ponto de fusão, ou seja, até “fluir” novamente.

Os sensores empregados neste projeto, por serem fornecidos em encapsulamento LGA, não podem ser soldados com um ferro de solda (visto que não possuem terminais expostos). Como não se dispunha de um forno de refluxo, estes componentes foram soldados com ar quente oriundo de um soprador térmico. O ar é direcionado sobre o componente, cujos terminais foram previamente estanhados, a princípio de uma distância grande, sendo a pistola gradativamente aproximada até aproximadamente 8 centímetros do componente. Após alguns segundos, a solda derrete e o componente, devido às tensões superficiais da solda derretida, se alinha sem intervenção humana; uma vez alinhado, a pistola é afastada gradativamente, permitindo um resfriamento uniforme e de boa qualidade.

Observação: Apesar de não ser necessário, um produto de grande auxílio durante a soldagem, principalmente de componentes SMD, é o fluxo de solda. O fluxo é uma substância levemente corrosiva que “ativa” a solda, abaixando seu ponto de fusão e melhorando suas propriedades físicas, o que torna a soldagem mais rápida e de melhor qualidade. Fluxo de solda “*no clean*” (sem necessidade de limpeza após o uso) foi utilizado na soldagem dos encapsulamentos LGA empregados neste trabalho.

- **Testes**

Terminada a fabricação da PCI, há uma etapa adicional opcional, porém altamente recomendada, que é a etapa de testes de continuidade. Estes testes têm como objetivo encontrar falhas nas trilhas, que podem ser o curto-circuito indevido entre trilhas ou o rompimento de uma trilha. Para é empregado um multímetro no modo de teste de continuidade (“bipador”).



## **APÊNDICE B – Códigos fonte em C**



- **main.c**

```

/* ***** */
/*      Sistema de orientação      */
/*      Programa principal          */
/*      Autor: Rafael Santos Moura  */
/* ***** */

#include "includes.h"

#define Wa          1      // Peso de confiança do acelerômetro
#define Wg          15     // Peso de confiança do giroscópio
#define servo1_duty TA1CCR1 // Pseudônimo do PWM1
#define servo2_duty TA1CCR2 // Pseudônimo do PWM2

extern volatile unsigned int Timer_A0_IF; // Indicação de timer estourado

float VAcc[3], T[3]; // Leitura do acelerômetro e do giroscópio

float VAccx_Samples[3] = {0,0,0}; // Armazenam valores passados de
float VAccy_Samples[3] = {0,0,0}; // leitura dos eixos do acelerômetro
float VAccz_Samples[3] = {0,0,0}; //

float Tx_Samples[3] = {0,0,0}; // Armazenam valores passados de
float Ty_Samples[3] = {0,0,0}; // leitura dos eixos do giroscópio
float Tz_Samples[3] = {0,0,0}; //

float H1[3] = {1,0,0}; // Vetor de orientação estimada X
float H2[3] = {0,1,0}; // Vetor de orientação estimada Y
float Vert[3] = {0,0,1}; // Vetor de orientação estimada Z

float deltaAng[3]; // Variação de ângulos

float PR_Correct[3]; // Vetor de correção de PITCH e ROLL

int signVert; // Armazena o sinal de Vert[2]

float PITCH, ROLL; // Ângulos de pitch e roll calculados para os servos

unsigned int aux_duty; // Variáveis auxiliares
float Norm;           //

int main(void) // Início do programa
{
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    BCSCTL1 = CALBC1_16MHZ;    // Set DCO to 16MHz
    DCOCTL = CALDCO_16MHZ;    // Set DCO to 16MHz

    USCI_A0_UART_Init();      // Initialize UART Module
    USCI_B0_SPI_Init();       // Initialize SPI Module

    TIMER_A0_Init();           // Initialize Timer A0
    TIMER_A1_Init();           // Initialize Timer A1

    __enable_interrupt();      // Interrupts Enabled

    __delay_cycles(10000);     // Delay

    while (USCI_B0_SPI_Read16(CS_ACC, ACC_WHO_AM_I, low) != 0x3B); // Lê os
    registradores de identificação
    while (USCI_B0_SPI_Read16(CS_GYRO, GYRO_WHO_AM_I, low) != 0xD3); // dos
    sensores, e pára o programa

    // se algum deles estiver incorreto

    /* Configuração do acelerômetro */

```

```

// Modo ligado, fundo de escala +-8g, taxa de amostragem em 400 Hz,
// eixos X, Y e Z ativos, filtro passa-alta em 1 Hz
USCI_B0_SPI_Writel6(CS_ACC, ACC_CTRL_REG1, ACC_POWER_DOWN|ACC_DATA_RATE|ACC_FUL
L_SCALE|0x87);
USCI_B0_SPI_Writel6(CS_ACC, ACC_CTRL_REG2, ACC_HP_Coeff_2|ACC_HP_Coeff_1);

/* Configuração do giroscópio */
// Modo ligado, fundo de escala em +-500graus/s, taxa de amostragem em 400
Hz,
// eixos X, Y e Z ativos
USCI_B0_SPI_Writel6(CS_GYRO, GYRO_CTRL_REG1, 0x8F);
USCI_B0_SPI_Writel6(CS_GYRO, GYRO_CTRL_REG4, 0x10);

while(1) // Início do loop principal
{
    if (Timer_A0_IF == 1) // Verifica se o timer já estourou,
// o início de um novo ciclo
    permitindo
    apenas após
    amostragem
    // decorrido o período de

    {
        /* ***** */
        /* ***** Acelerômetro ***** */
        /* ***** */

        READ_ACC_OUTPUT(VAcc);

        // Conversão para m/s^2
        VAcc[0] = VAcc[0]/1.40861;
        VAcc[1] = VAcc[1]/1.40861;
        VAcc[2] = VAcc[2]/1.40861;

        // Correção na função de transferência
        VAcc[0] = -(VAcc[0]*1.20 + 1.25);
        VAcc[1] = -(VAcc[1]*1.25 - 1.25);
        VAcc[2] = +(VAcc[2]*1.25 - 0.25);

        // Filtro média móvel
        VAcc[0] = (VAccx_Samples[0] + VAccx_Samples[1] +
VAccx_Samples[2] + 4*VAcc[0])/7;

        VAccx_Samples[0] = VAccx_Samples[1];
        VAccx_Samples[1] = VAccx_Samples[2];
        VAccx_Samples[2] = VAcc[0];

        VAcc[1] = (VAccy_Samples[0] + VAccy_Samples[1] +
VAccy_Samples[2] + 4*VAcc[1])/7;

        VAccy_Samples[0] = VAccy_Samples[1];
        VAccy_Samples[1] = VAccy_Samples[2];
        VAccy_Samples[2] = VAcc[1];

        VAcc[2] = (VAccz_Samples[0] + VAccz_Samples[1] +
VAccz_Samples[2] + 4*VAcc[2])/7;

        VAccz_Samples[0] = VAccz_Samples[1];
        VAccz_Samples[1] = VAccz_Samples[2];
        VAccz_Samples[2] = VAcc[2];

        // Supressão de ruído dentro da margem
        if (fabs(VAcc[0]) < 0.9) VAcc[0] = 0;
        if (fabs(VAcc[1]) < 0.9) VAcc[1] = 0;
        if (fabs(VAcc[2]) < 0.9) VAcc[2] = 0;

        // Cálculo da norma de VAcc

```

```

VAcc[2]*VAcc[2]);

Norm = sqrt(VAcc[0]*VAcc[0] + VAcc[1]*VAcc[1] +
VAcc[2]*VAcc[2]);

// Normalização do vetor Acelerômetro
VAcc[0] = VAcc[0]/Norm;
VAcc[1] = VAcc[1]/Norm;
VAcc[2] = VAcc[2]/Norm;

/* ***** */
/* ***** Giroscópio ***** */
/* ***** */

READ_GYRO_OUTPUT(T);
// Conversão para graus por segundo

T[0] = T[0]/65.534;
T[1] = T[1]/65.534;
T[2] = T[2]/65.534;

T[0] = -(T[0]*1.10 + 0.0);
T[1] = +(T[1]*1.10 + 4.5);
T[2] = -(T[2]*1.10 + 4.5);

// Filtro média móvel
T[0] = (Tx_Samples[0] + Tx_Samples[1] + Tx_Samples[2] +
5*T[0])/8;

Tx_Samples[0] = Tx_Samples[1];
Tx_Samples[1] = Tx_Samples[2];
Tx_Samples[2] = T[0];

T[1] = (Ty_Samples[0] + Ty_Samples[1] + Ty_Samples[2] +
5*T[1])/8;

Ty_Samples[0] = Ty_Samples[1];
Ty_Samples[1] = Ty_Samples[2];
Ty_Samples[2] = T[1];

T[2] = (Tz_Samples[0] + Tz_Samples[1] + Tz_Samples[2] +
5*T[2])/8;

Tz_Samples[0] = Tz_Samples[1];
Tz_Samples[1] = Tz_Samples[2];
Tz_Samples[2] = T[2];

// Supressão de ruído dentro da margem
if (fabs(T[0]) < 6) T[0] = 0;
if (fabs(T[1]) < 6) T[1] = 0;
if (fabs(T[2]) < 6) T[2] = 0;

/* ***** Cálculos ***** */

// Cálculo do vetor de correção de Pitch e Roll
cross_product(Vert, VAcc, PR_Correct);
PR_Correct[0] = +PR_Correct[0]*15;
PR_Correct[1] = +PR_Correct[1]*15;
PR_Correct[2] = 0;

// Cálculo do vetor de variação de ângulos
deltaAng[0] = (T[0]*Wg/SampleRate - Wa*PR_Correct[0])/(Wa +
Wg)*PI/180;
deltaAng[1] = (T[1]*Wg/SampleRate - Wa*PR_Correct[1])/(Wa +
Wg)*PI/180;
deltaAng[2] = (T[2]*Wg/SampleRate - Wa*PR_Correct[2])/(Wa +
Wg)*PI/180;

// Rotação dos versores de orientação estimada
rotate3D(deltaAng[0], deltaAng[1], Vert);

```

```

rotate3D(deltaAng[0], deltaAng[1], H1);
rotate3D(deltaAng[0], deltaAng[1], H2);

// Cálculo de Pitch e Roll a partir do acelerômetro
if (Vert[2] < 0) signVert = -1; else signVert = 1;
PITCH = atan2(Vert[0], signVert*sqrt(Vert[1]*Vert[1] +
Vert[2]*Vert[2]))*180/PI;
ROLL = atan2(Vert[1], signVert*sqrt(Vert[0]*Vert[0] +
Vert[2]*Vert[2]))*180/PI;

// Calcula o duty cycle a ser enviado para os servos, baseado
// em seus respectivos ângulos. Se o tempo em alto sair da
faixa

// de 1 ms a 2 ms, o duty cycle não é atualizado
aux_duty = (unsigned int)(1000 + (PITCH + 90)*21.1111111);
if (aux_duty > 1000 && aux_duty < 5000) servo1_duty = aux_duty;

aux_duty = (unsigned int)(1000 + (ROLL + 90)*21.1111111);
if (aux_duty > 1000 && aux_duty < 5000) servo2_duty = aux_duty;

// Conversão para ASCII e Transmissão

SendValuesUART(Vert[0], Vert[1], Vert[2], 0);
SendValuesUART(H1[0], H1[1], H1[2], 0);
SendValuesUART(H2[0], H2[1], H2[2], 1);

Timer_A0_IF = 0; // Limpa a indicação de que o timer estourou
}
} /* Fim do loop principal */
} // Fim do programa

```

- **INCLUDES.h**

```
#include "msp430g2553.h"           // Definições do microcontrolador

#include "math.h"                   // Funções matemáticas padrão

#include "SPI.h"                    // Módulo SPI
#include "UART.h"                   // Módulo UART

#include "LIS302DL.h"               // Módulo acelerômetro
#include "L3G4200D.h"              // Módulo giroscópio

#include "TIMER_A0.h"               // Módulo timer e PWM

#include "CUSTOM_FUNCTIONS.h"       // Módulo misto
```

- **TIMER\_A0.c**

```
#include "msp430g2553.h"

unsigned volatile int Timer_A0_IF; // Flag que indica que o timer estourou

/* TIMER_A0_Init */
// Inicializa o módulo TIMER_A0 em modo temporizador, contador para cima
// Entrada: Nada
// Retorna: Nada
void TIMER_A0_Init()
{
    TA0CTL = ID_3 + TASSEL_2 + MC_1;    // Source = SMCLK, Count up to
    TACCR0, Divide by 8

    TA0CCR0 = 0x9C3F;                    /* Source freq. / Sample Rate =
                                         /* 16/8 MHz/50 Hz = 40000 =
0x9C40 -> TACCR0 = 0x9C3F

    TA0CCTL0 = CCIE;                     // Habilita interrupção do timer

    Timer_A0_IF = 0;                     // Timer não estourou

    return;
}

/* TIMER_A1_Init */
// Inicializa o módulo TIMER_A1 em modo PWM
// Entrada: Nada
// Retorna: Nada
void TIMER_A1_Init()
{
    P2DIR |= BIT2 + BIT4;    // Configura os pinos P2.2 e P2.4
    P2SEL |= BIT2 + BIT4;    // como saídas de PWM

    TA1CCTL1 = OUTMOD_7;     // Configura o modo do PWM
    TA1CCTL2 = OUTMOD_7;     // como reset/set

    TA1CTL = ID_3 + TASSEL_2 + MC_1;    // Source = SMCLK, Count up to
    TACCR0, divide by 8

    TA1CCR0 = 0x9C3F; /* Source freq. / 50 Hz = (TACCR0 + 1)
                     /* 16/8 MHz / 50 Hz = 40000 = 0x9C40 -
> TACCR0 = 0x9C3F

    TA1CCR1 = 0x0BB7; // 0x9C3F = 20 ms, 0x0F9F = 2 ms, 0x07CF = 1 ms
    TA1CCR2 = 0x0BB7; // Tempo alto inicial = 1,5 ms

    return;
}
```

- **TIMER\_A0.h**

```
#define SampleRate 50 // Frequência de amostragem

/* Protótipos das funções */
void TIMER_A0_Init();

void TIMER_A1_Init();
```

- **SPI.c**

```
#include "msp430g2553.h"
#include "SPI.h"

/* USCI_B0_SPI_Init */
// Inicializa o módulo USCI B0 em modo SPI
// Entrada: Nada
// Retorna: Nada
void USCI_B0_SPI_Init()
{
    /* Configuração dos pinos */
    P1SEL |= UCB0SIMO + UCB0SOMI + UCB0CLK;
    P1SEL2 |= UCB0SIMO + UCB0SOMI + UCB0CLK;

    P2DIR |= BIT1; // CS_ACC
    P2OUT |= BIT1; //
    P1DIR |= BIT3; // CS_GYRO
    P1OUT |= BIT3; //
    P1DIR |= BIT0; // CS_MAG
    P1OUT |= BIT0; //
// P2DIR |= BIT2; // CS_LCD
// P2OUT |= BIT2; //

    /* Configuração da fonte de clock */
    UCB0CTL1 |= UCSSEL_2; // SMCLK - SubMainClock

    /* Configuração de controle */
    UCB0CTL0 |= 0x69; // CPHA = 0, CPOL = 1, MSB
    FIRST, 8-BIT DATA, // MASTER MODE, 3-PIN
    SPI, SYNCHRONOUS

    /* Configuração do gerador de baud rate */
    UCB0BR1=0x00; UCB0BR0=0x02; // SMCLK dividido por 2

    /* Inicializa a máquina de estados */
    UCB0CTL1 &= ~UCSWRST;

    return;
}

/* USCI_B0_SPI_Writel6 */
// Escreve um dado de 8 bits em registrador interno de dispositivo no
barramento SPI
// Entrada: cs_device: ID do dispositivo alvo
// address: Endereço do registrador interno do dispositivo
// data: Dados a serem escritos
// Retorna: Nada
void USCI_B0_SPI_Writel6(unsigned int cs_device, unsigned int address,
unsigned int data)
{
    unsigned int TX_BYTE_1;

    TX_BYTE_1 = address & ~(READ_BIT + MULT_BIT); // Endereço, Leitura
Simples

    switch (cs_device) // Coloca o CS do dispositivo em nível baixo
    {
        case 1:
```

```

        P2OUT &= ~BIT1;
        break;
    case 2:
        P1OUT &= ~BIT3;
        break;
    case 3:
        P1OUT &= ~BIT0;
        break;
    case 4:
        P2OUT &= ~BIT2;
        break;
}

UCB0TXBUF = TX_BYTE_1;          // Transmite o endereço
UCB0TXBUF = data;                // Trasmite o dado
while(UCB0STAT & UCBUSY);        // Espera término da transação

switch (cs_device) // Coloca o CS do dispositivo em nível alto
{
    case 1:
        P2OUT |= BIT1;
        break;
    case 2:
        P1OUT |= BIT3;
        break;
    case 3:
        P1OUT |= BIT0;
        break;
    case 4:
        P2OUT |= BIT2;
        break;
}

return;
}

/* USCI_B0_SPI_Read16 */
// Lê um dado de 8 bits de registrador interno de dispositivo no barramento
// SPI
// Entrada: cs_device: ID do dispositivo alvo
//          address: Endereço do registrador interno do dispositivo
//          highB: Indica se o valor lido é o byte mais significativo
//          (usado para verificar sinal do valor)
// Retorna: Palavra de 16 bits, com sinal
signed int USCI_B0_SPI_Read16(unsigned int cs_device, unsigned int address,
unsigned int highB)
{
    unsigned int RX_BYTE_1;
    signed int receive_SPI;

    RX_BYTE_1 = (address | READ_BIT) & ~MULT_BIT;

    switch (cs_device) // Coloca o CS do dispositivo em nível baixo
    {
        case 1:
            P2OUT &= ~BIT1;
            break;
        case 2:
            P1OUT &= ~BIT3;
            break;

```

```

        case 3:
            P1OUT &= ~BIT0;
            break;
        case 4:
            P2OUT &= ~BIT2;
            break;
    }
    UCB0TXBUF = RX_BYTE_1;           // Transmite endereço
    UCB0TXBUF = 0x0A;                // Byte aleatório
    while(UCB0STAT & UCBUSY);

    switch (cs_device) // Coloca o CS do dispositivo em nível alto
    {
        case 1:
            P2OUT |= BIT1;
            break;
        case 2:
            P1OUT |= BIT3;
            break;
        case 3:
            P1OUT |= BIT0;
            break;
        case 4:
            P2OUT |= BIT2;
            break;
    }

    receive_SPI = UCB0RXBUF;         // Recebe dado lido

    /* Transformação de 8 bits com sinal para 16 bits com sinal */
    if (highB) // Se o byte é o high byte, verifica o bit de sinal do
dado de 8 bits
    {
        if (receive_SPI & 0x0080)    // Se MSB = 1, então o número é
negativo.
        {
            // Se é negativo, o
sinal é colocado no MSB
            receive_SPI ^= 0x00FF;    // da palavra de 16 bits
            receive_SPI++;
            receive_SPI ^= 0xFFFF;
        }
    }

    return receive_SPI;
}

```

- SPI.h

```
#include "msp430g2553.h"

/* Definições do módulo USCI_B0 em modo SPI*/

#define READ_BIT 0x80 // Leitura(1) ou escrita(0) de dados
#define MULT_BIT 0x40 // Operação simples(0) ou múltipla(1)

#define high 1 // Indica byte mais significativo
#define low 0 // Indica byte não mais significativo

/* IDs dos dispositivos */
#define CS_ACC 1 // Chip Select Acelerômetro
#define CS_GYRO 2 // Chip Select Giroscópio
#define CS_MAG 3 // Chip Select Magnetômetro
#define CS_LCD 4 // Chip Select LCD

/* Definição dos bits do barramento */
#define UCB0CLK BIT5 // USCI B0 Master CLK
#define UCB0SOMI BIT6 // USCI B0 SOMI - Slave Out Master In
#define UCB0SIMO BIT7 // USCI B0 SIMO - Slave In Master Out

#define LCD_DIO BIT3 // LCD SIMO (P2.3)
#define LCD_SCL BIT5 // LCD SCLK (P2.5)

/* Protótipo das funções */

void USCI_B0_SPI_Init();

void USCI_B0_SPI_Writel6(unsigned int cs_device, unsigned int address,
unsigned int data);

signed int USCI_B0_SPI_Readl6(unsigned int cs_device, unsigned int address,
unsigned int highB);
```

- **ISR.c**

```
#include "msp430g2553.h"

extern volatile unsigned int Timer_A0_IF; // Flag que indica que o timer já
estourou

// Tipo "volatile" indica que a variável pode ser alterada sem sincronismo
com o
// programa principal. "volatile" evita otimizações indesejadas no código
pelo compilador

// Tipo "extern" indica que a variável já foi declarada em outro arquivo

// Rotina de interrupção do Timer A0
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{
    if (TA0CTL & TAIFG)          // Verifica se a flag de interrupção
está ativa
    {
        Timer_A0_IF = 1; // Indica que o timer estourou
        TA0CTL &= ~TAIFG; // Limpa a flag de interrupção
    }

    return;
}
```

- CUSTOM\_FUNCTIONS.c

```
#include "INCLUDES.h"

/* float2BCD */
// Converte uma variável float em vetores BCD no formato (sinal)XXX.XXX;
// Esta função modifica diretamente os valores dos vetores fornecidos
// Entrada: numero: Variável float a ser convertida
//          *BCD_i: Ponteiro do vetor que conterà o sinal e a parte
inteira
//          *BCD_f: Ponteiro do vetor que conterà a parte fracionária
// Retorna: Nada
void float2BCD(float numero, char *BCD_i, char *BCD_f)
{
    int i;
    signed int integer;
    signed int fraction;

    integer = (signed int)numero;
    fraction = (unsigned int)(numero*1000 - integer*1000);

    if (numero < 0)    // Se o número for negativo
    {
        *BCD_i = 0x2D;    // Sinal negativo
    }
    else
    {
        *BCD_i = 0x20;    // Espaço
    }

    integer = abs(integer);    // Separa a parte inteira em "integer"
    fraction = abs(fraction);    // Separa a parte fracionária em "fraction"

    BCD_i++;    // Incrementa ponteiro do vetor

    for (i=0;i<3;i++)
    {
        *BCD_i = (char)(integer % 10);    // Armazena um dígito no
vetor
        integer = integer / 10;    // Divide por 10
        BCD_i++;    // Incrementa ponteiro
do vetor

        *BCD_f = (char)(fraction % 10);    // Armazena um dígito no vetor
        fraction = fraction / 10;    // Divide por 10
        BCD_f++;    // Incrementa ponteiro
do vetor
    }

    return;
}

/* integrate_data */
// Calcula a integral discreta (somatório) de uma variável float entre seu
valor atual e seu valor imediatamente passado.
// Cálculo utilizando aproximação trapezoidal para diminuir erro de
integração.
// Entrada: data_now: Variável float a ser integrada, valor atual
//          data_before: Variável float a ser integrada, valor
passado
```

```

//          initial_integrated_data: valor passado da integral
// Retorna: integrated_data: Valor calculado atual da integral

// Fórmula: IntegralAtual = IntegralPassada + (ValorAtual + |(ValorAtual -
// ValorPassado)|/(2*Frequência de amostragem)
float integrate_data(float data_now, float data_before, float
initial_integrated_data)
{
    float integrated_data;

    integrated_data = ((data_now + (abs(data_now - data_before) >>
1))/SampleRate + initial_integrated_data);

    return integrated_data;
}

/* cross_product */
// Calcula o produto vetorial V3 = V1 X V2
// Esta função modifica diretamente os valores de V3
// Entrada: V1: Primeiro vetor da multiplicação
//          V2: Segundo vetor da multiplicação
//          V3: Resultado da multiplicação
// Retorna: Nada
void cross_product(float *V1, float *V2, float *V3)
{
    *(V3+0) = ( *(V1+1) * *(V2+2)) - (*(V1+2) * *(V2+1)) );
    *(V3+1) = - ( *(V1+0) * *(V2+2)) - (*(V1+2) * *(V2+0)) );
    *(V3+2) = ( *(V1+0) * *(V2+1)) - (*(V1+1) * *(V2+0)) );
}

/* rotate3D */
// Rotaciona o vetor em X e Y
// Esta função modifica diretamente os valores de "vector"
// Entrada: p: pitch
//          r: roll
//          vector: Vetor a ser rotacionado
// Retorna: Nada
void rotate3D(float p, float r, float *vector)
{
    float temp1, temp2;

    temp1 = cos(r)**(vector+0) -sin(r)**(vector+2);
    temp2 = sin(p)*sin(r)**(vector+0) + cos(p)**(vector+1) +
sin(p)*cos(r)**(vector+2);
    *(vector+2) = cos(p)*sin(r)**(vector+0) - sin(p)**(vector+1) +
cos(p)*cos(r)**(vector+2);

    *(vector+0) = temp1;
    *(vector+1) = temp2;
}

```

- **CUSTOM\_FUNCTIONS.h**

```
#define PI 3.14159265

/* Protótipos das funções */
void float2BCD(float numero, char *BCD_i, char *BCD_f);

float integrate_data(float data_now, float data_before, float
initial_integrated_data);

void cross_product(float *V1, float *V2, float *V3);

void rotate3D(float p, float r, float *vector);
```

- **UART.c**

```
#include "msp430g2553.h"
#include "uart.h"
#include "custom_functions.h"

// Obs.: Se TXIE = 1, GIE = 1 e TXFG = 1, isto é, se a interrupção de TX
// e as interrupções globais estão ligadas e a flag de TX indica buffer
pronto,
// Então será gerado um pedido de interrupção.

// Tipo "volatile" indica que a variável pode ser alterada sem sincronismo
com o
// programa principal. "volatile" evita otimizações indesejadas no código
pelo compilador

volatile unsigned int tx_flag;      // Flag que indica transmissão não
efetuada
volatile unsigned char tx_char;     // Armazena o char a ser transmitido
pela UART

volatile unsigned int uart_f; // Flag que indica se há transmissão pendente

/* USCI_B0_SPI_Init */
// Inicializa o módulo USCI A0 em modo UART
// Entrada: Nada
// Retorna: Nada
void USCI_A0_UART_Init()
{
    /* Configuração dos pinos */
    P1SEL |= RXD + TXD;
    P1SEL2 |= RXD + TXD;

    UCA0CTL1 |= UCSSEL_2;          // Clock source: SMCLK - SubMainClock

    UCA0BR0 = 0x02;                // Division factor of clock source to
generate baud rate
    UCA0BR1 = 0x00;                // BaudRate = 460800
    UCA0MCTL = 0x26|UCOS16;        // Modulation Control and Divide by 16

    UCA0CTL1 &= ~UCSWRST;          // USCI state machine on

    tx_flag = 0;                   // Set tx_flag to 0
    uart_f = 0;                    // Set tx_flag to 0

    return;
}

/* USCI_A0_UART_puthex */
// Envia um dado de 8 bits pelo módulo UART
// Entrada: hex: Dado de 8 bits
// Retorna: Nada
void USCI_A0_UART_puthex(unsigned int hex)
{
    uart_f = 1;
    tx_char = hex;                 // Copia o dado em tx_char
}
```

```

        IE2 |= UCA0TXIE;           // Liga interrupção de transmissão UART
        while(tx_flag == 1);       // Espera eventual transmissão em andamento
ser finalizada
        tx_flag = 1;               // Indica que nova transmissão começou
        return;
}

/* SendValuesUART */
// Conversão para ASCII, na forma XXX.XXX, e transmissão de 3 valores float
// pelo módulo UART
// Entrada: value1: Primeiro float
//           value2: Segundo float
//           value3: Terceiro float
//           linefeed: Indica a necessidade de nova linha (1 = sim, 0
// = não)
// Retorna: Nada
void SendValuesUART(float value1, float value2, float value3, char
linefeed)
{
    char BCD_i[4], BCD_f[3];

    float2BCD(value1, BCD_i, BCD_f); // Conversão para BCD

    USCI_A0_UART_puthex(BCD_i[0]); // Sinal
    USCI_A0_UART_puthex(BCD_i[3] + 0x30); USCI_A0_UART_puthex(BCD_i[2] +
0x30); USCI_A0_UART_puthex(BCD_i[1] + 0x30);
    USCI_A0_UART_puthex(46); // Ponto
    USCI_A0_UART_puthex(BCD_f[2] + 0x30); USCI_A0_UART_puthex(BCD_f[1] +
0x30); USCI_A0_UART_puthex(BCD_f[0] + 0x30);
    USCI_A0_UART_puthex(0x09); // Tab

    float2BCD(value2, BCD_i, BCD_f); // Conversão para BCD

    USCI_A0_UART_puthex(BCD_i[0]); // Sinal
    USCI_A0_UART_puthex(BCD_i[3] + 0x30); USCI_A0_UART_puthex(BCD_i[2] +
0x30); USCI_A0_UART_puthex(BCD_i[1] + 0x30);
    USCI_A0_UART_puthex(46); // Ponto
    USCI_A0_UART_puthex(BCD_f[2] + 0x30); USCI_A0_UART_puthex(BCD_f[1] +
0x30); USCI_A0_UART_puthex(BCD_f[0] + 0x30);
    USCI_A0_UART_puthex(0x09); // Tab

    float2BCD(value3, BCD_i, BCD_f); // Conversão para BCD

    USCI_A0_UART_puthex(BCD_i[0]); // Sinal
    USCI_A0_UART_puthex(BCD_i[3] + 0x30); USCI_A0_UART_puthex(BCD_i[2] +
0x30); USCI_A0_UART_puthex(BCD_i[1] + 0x30);
    USCI_A0_UART_puthex(46); // Ponto
    USCI_A0_UART_puthex(BCD_f[2] + 0x30); USCI_A0_UART_puthex(BCD_f[1] +
0x30); USCI_A0_UART_puthex(BCD_f[0] + 0x30);
    USCI_A0_UART_puthex(0x09); // Tab
    if (linefeed == 1) USCI_A0_UART_puthex(0x0A); // Nova linha
}

// Rotina de interrupção de transmissão UART
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
    if(uart_f == 1) // Transmissão pendente?
    {
        UCA0TXBUF = tx_char; // Copia tx_char para o buffer TX
        tx_flag = 0;          // Transmissão efetuada
    }
}

```

```
        uart_f = 0;
        IE2 &= ~UCA0TXIE;
transmissão UART
    }
}
```

```
// Sem transmissão pendente
// Desliga interrupção de
```

- **UART.h**

```
/* Definição dos pinos */
#define RXD BIT1 // UCA0RDX UART
#define TXD BIT2 // UCA0TXD UART

/* Protótipos das funções */
void USCI_A0_UART_Init(void);

void USCI_A0_UART_puthex(unsigned int hex);

void SendValuesUART(float value1, float value2, float value3, char
linefeed);
```

- LIS302DL.c

```
#include "LIS302DL.h"
#include "SPI.h"

/* READ_ACC_OUTPUT */
// Lê os três eixos do acelerômetro (3 palavras de 8 bits)
// Esta função modifica diretamente os valores dos vetores fornecidos
// Entrada: *ACC_OUTPUT: Ponteiro do vetor que conterá os valores lidos
// Retorna: Nada
void READ_ACC_OUTPUT(float* ACC_OUTPUT)
{
    *ACC_OUTPUT = USCI_B0_SPI_Read16(CS_ACC, ACC_OUT_X, high);
    ACC_OUTPUT++;
    *ACC_OUTPUT = USCI_B0_SPI_Read16(CS_ACC, ACC_OUT_Y, high);
    ACC_OUTPUT++;
    *ACC_OUTPUT = USCI_B0_SPI_Read16(CS_ACC, ACC_OUT_Z, high);
}
```

- LIS302DL.h

```

/* REGISTER ADDRESS MAPPING */

#define ACC_WHO_AM_I                0x0F
#define ACC_CTRL_REG1               0x20
#define ACC_CTRL_REG2               0x21
#define ACC_CTRL_REG3               0x22
#define ACC_HP_FILTER_RESET         0x23
#define ACC_STATUS_REG              0x27
#define ACC_OUT_X                   0x29
#define ACC_OUT_Y                   0x2B
#define ACC_OUT_Z                   0x2D
#define ACC_FF_WU_CFG_1             0x30
#define ACC_FF_WU_SRC_1             0x31
#define ACC_FF_WU_THS_1             0x32
#define ACC_FF_WU_DURATION_1 0x33
#define ACC_FF_WU_CFG_2             0x34
#define ACC_FF_WU_SRC_2             0x35
#define ACC_FF_WU_THS_2             0x36
#define ACC_FF_WU_DURATION_2 0x37
#define ACC_CLICK_CFG               0x38
#define ACC_CLICK_SRC               0x39
#define ACC_CLICK_THSY_X            0x3B
#define ACC_CLICK_THSZ              0x3C
#define ACC_CLICK_TimeLimit         0x3D
#define ACC_CLICK_Latency           0x3E
#define ACC_CLICK_Window            0x3F

/* CTRL_REG_1 BIT DEFINITIONS */

#define ACC_DATA_RATE                0x80
#define ACC_POWER_DOWN               0x40
#define ACC_FULL_SCALE               0x20
#define ACC_STP                      0x10
#define ACC_STM                      0x08

/* CTRL_REG_2 BIT DEFINITIONS */

#define ACC_BOOT                     0x40
#define ACC_FDS                      0x10
#define ACC_HP_Coeff_2               0x02
#define ACC_HP_Coeff_1               0x01

/* STATUS_REG BIT DEFINITIONS */

#define ACC_ZYXOR                    0x80
#define ACC_ZOR                      0x40
#define ACC_YOR                      0x20
#define ACC_XOR                      0x10
#define ACC_ZYXDA                    0x08
#define ACC_ZDA                      0x04
#define ACC_YDA                      0x02
#define ACC_XDA                      0x01

/* FUNCTION PROTOTYPES */

void READ_ACC_OUTPUT(float* ACC_OUTPUT);

```

- **L3G4200D.c**

```
#include "L3G4200D.h"
#include "SPI.h"

/* READ_ACC_OUTPUT */
// Lê os três eixos do giroscópio (3 palavras de 16 bits)
// Esta função modifica diretamente os valores dos vetores fornecidos
// Entrada: *GYRO_OUTPUT: Ponteiro do vetor que conterà os valores lidos
// Retorna: Nada
void READ_GYRO_OUTPUT(float* GYRO_OUTPUT)
{
    *GYRO_OUTPUT = USCI_B0_SPI_Read16(CS_GYRO, GYRO_OUT_Y_H, high);
    *GYRO_OUTPUT = (*GYRO_OUTPUT)*256;
    *GYRO_OUTPUT = *GYRO_OUTPUT +
    USCI_B0_SPI_Read16(CS_GYRO, GYRO_OUT_Y_L, low);
    GYRO_OUTPUT++;

    *GYRO_OUTPUT = USCI_B0_SPI_Read16(CS_GYRO, GYRO_OUT_X_H, high);
    *GYRO_OUTPUT = (*GYRO_OUTPUT)*256;
    *GYRO_OUTPUT = *GYRO_OUTPUT +
    USCI_B0_SPI_Read16(CS_GYRO, GYRO_OUT_X_L, low);
    GYRO_OUTPUT++;

    *GYRO_OUTPUT = USCI_B0_SPI_Read16(CS_GYRO, GYRO_OUT_Z_H, high);
    *GYRO_OUTPUT = (*GYRO_OUTPUT)*256;
    *GYRO_OUTPUT = *GYRO_OUTPUT +
    USCI_B0_SPI_Read16(CS_GYRO, GYRO_OUT_Z_L, low);
}
```

- L3G4200D.h

```

/* REGISTER ADDRESS MAPPING */

#define GYRO_WHO_AM_I          0x0F
#define GYRO_CTRL_REG1        0x20
#define GYRO_CTRL_REG2        0x21
#define GYRO_CTRL_REG3        0x22
#define GYRO_CTRL_REG4        0x23
#define GYRO_CTRL_REG5        0x24
#define GYRO_REFERENCE         0x25
#define GYRO_OUT_TEMP          0x26
#define GYRO_STATUS_REG        0x27
#define GYRO_OUT_X_L           0x28
#define GYRO_OUT_X_H           0x29
#define GYRO_OUT_Y_L           0x2A
#define GYRO_OUT_Y_H           0x2B
#define GYRO_OUT_Z_L           0x2C
#define GYRO_OUT_Z_H           0x2D
#define GYRO_FIFO_CTRL_REG     0x2E
#define GYRO_FIFO_SRC_REG      0x2F
#define GYRO_INT1_CFG          0x30
#define GYRO_INT1_SRC          0x31
#define GYRO_INT1_TSH_XH       0x32
#define GYRO_INT1_TSH_XL       0x33
#define GYRO_INT1_TSH_YH       0x34
#define GYRO_INT1_TSH_YL       0x35
#define GYRO_INT1_TSH_ZH       0x36
#define GYRO_INT1_TSH_ZL       0x37
#define GYRO_INT1_DURATION     0x38

/* FUNCTION PROTOTYPES */

void READ_GYRO_OUTPUT(float* GYRO_OUTPUT);

```

- **CP2120.c**

```
#include "msp430g2553.h"
```

```
/* ***** */
// As funções abaixo são para a comunicação com o magnetômetro através do
// CP2120, e não foram utilizadas
/* ***** */
```

```
void USCI_B0_SPI_CP_SPI_MSBF()
{
    P1OUT &= ~BIT0;

    UCB0TXBUF = 0x18;
    _delay_cycles(8);
    UCB0TXBUF = 0x81;
    while(UCB0STAT & UCBUSY);

    P1OUT |= BIT0;

    return;
}
```

```
void USCI_B0_SPI_CP_WRITE_INTREG(unsigned int address, unsigned int data)
{
    P1OUT &= ~BIT0;

    UCB0TXBUF = 0x20;
    _delay_cycles(8);
    UCB0TXBUF = address;
    _delay_cycles(8);
    UCB0TXBUF = data;
    while(UCB0STAT & UCBUSY);

    P1OUT |= BIT0;

    return;
}
```

```
void USCI_B0_SPI_CP_READ_I2C(unsigned int address)
{
    P1OUT &= ~BIT0;

    UCB0TXBUF = 0x02; // Comando
    _delay_cycles(10);
    UCB0TXBUF = 0x01; // Escreve 1 byte
    _delay_cycles(10);
    UCB0TXBUF = 0x01; // Lê 1 byte
    _delay_cycles(10);

    UCB0TXBUF = 0x3C; // Endereço do magnetômetro (W)
    _delay_cycles(10);
    UCB0TXBUF = address; // Endereço do registrador
    _delay_cycles(10);
    UCB0TXBUF = 0x3D; // Endereço do magnetômetro (R)
    _delay_cycles(10);

    while(UCB0STAT & UCBUSY);

    P1OUT |= BIT0;
```

```

        return;
    }

    signed int USCI_B0_SPI_CP_READ_BUFFER(unsigned int highB)
    {
        signed int receive_SPI;

        P1OUT &= ~BIT0;

        UCB0TXBUF = 0x06;
        UCB0TXBUF = 0x0A;           // Dummy byte
        _delay_cycles(10);
        UCB0TXBUF = 0x0A;           // Dummy byte
        while(UCB0STAT & UCBUSY);

        receive_SPI = UCB0RXBUF;

        if (highB) // Se o byte é o high byte, verifica o bit de sinal
        {
            if (receive_SPI & 0x0080)
            {
                receive_SPI ^= 0x00FF;
                receive_SPI++;
                receive_SPI ^= 0xFFFF;
            }
        }

        P1OUT |= BIT0;

        return receive_SPI;
    }

```

- **CP2120.h**

```
/* REGISTER ADDRESS MAPPING */
```

```
#define CP_IOCONFIG          0x00
#define CP_IOSTATE           0x01
#define CP_I2CCLOCK          0x02
#define CP_I2CTO             0x03
#define CP_I2CSTAT           0x04
#define CP_I2CADR            0x05
#define CP_RXBUFF            0x06
#define CP_IOCONFIG2         0x07
#define CP_EDGEINT           0x08
#define CP_I2CTO2            0x09
```

```
/* CP_IOCONFIG BIT DEFINITIONS */
```

```
#define CP_PCIO3.1          0x80
#define CP_PCIO3.0          0x40
#define CP_PCIO2.1          0x20
#define CP_PCIO2.0          0x10
#define CP_PCIO1.1          0x08
#define CP_PCIO1.0          0x04
#define CP_PCIO0.1          0x02
#define CP_PCIO0.0          0x01
```

```
/* CP_IOSTATE BIT DEFINITIONS */
```

```
#define CP_GPIO7            0x80
#define CP_GPIO6            0x40
#define CP_GPIO5            0x20
#define CP_GPIO4            0x10
#define CP_GPIO3            0x08
#define CP_GPIO2            0x04
#define CP_GPIO1            0x02
#define CP_GPIO0            0x01
```

```
/* CP_I2CCLOCK BIT DEFINITIONS */
```

```
#define CP_I2CCK7           0x80
#define CP_I2CCK6           0x40
#define CP_I2CCK5           0x20
#define CP_I2CCK4           0x10
#define CP_I2CCK3           0x08
#define CP_I2CCK2           0x04
#define CP_I2CCK1           0x02
#define CP_I2CCK0           0x01
```

```
/* CP_I2CTO BIT DEFINITIONS */
```

```
#define CP_TO6              0x80
#define CP_TO5              0x40
#define CP_TO4              0x20
#define CP_TO3              0x10
#define CP_TO2              0x08
#define CP_TO1              0x04
```

```

#define CP_TO0                0x02
#define CP_TEN                0x01

```

```

/* CP_I2CSTAT BIT DEFINITIONS */

```

```

#define CP_I2ST7              0x80
#define CP_I2ST6              0x40
#define CP_I2ST5              0x20
#define CP_I2ST4              0x10
#define CP_I2ST3              0x08
#define CP_I2ST2              0x04
#define CP_I2ST1              0x02
#define CP_I2ST0              0x01

```

```

/* CP_I2CADR BIT DEFINITIONS */

```

```

#define CP_I2CAD7             0x80
#define CP_I2CAD6             0x40
#define CP_I2CAD5             0x20
#define CP_I2CAD4             0x10
#define CP_I2CAD3             0x08
#define CP_I2CAD2             0x04
#define CP_I2CAD1             0x02
#define CP_I2CAD0             0x01

```

```

/* CP_RXBUFF BIT DEFINITIONS */

```

```

#define CP_RXB7               0x80
#define CP_RXB6               0x40
#define CP_RXB5               0x20
#define CP_RXB4               0x10
#define CP_RXB3               0x08
#define CP_RXB2               0x04
#define CP_RXB1               0x02
#define CP_RXB0               0x01

```

```

/* CP_IOCONFIG2 BIT DEFINITIONS */

```

```

#define CP_PCIO7.1            0x80
#define CP_PCIO7.0            0x40
#define CP_PCIO6.1            0x20
#define CP_PCIO6.0            0x10
#define CP_PCIO5.1            0x08
#define CP_PCIO5.0            0x04
#define CP_PCIO4.1            0x02
#define CP_PCIO4.0            0x01

```

```

/* CP_EDGEINT BIT DEFINITIONS */

```

```

#define CP_EIF                0x80
#define CP_EIE                0x40
#define CP_EIT                0x20

```

```

/* CP_I2CTO2 BIT DEFINITIONS */

```

```
#define CP_FREN                                0x02
#define CP_LWEN                                0x01

/* FUNCTION PROTOTYPES */

void USCI_B0_SPI_CP_SPI_MSBF();

void USCI_B0_SPI_CP_WRITE_INTREG(unsigned int address, unsigned int data);

void USCI_B0_SPI_CP_READ_I2C(unsigned int address);

signed int USCI_B0_SPI_CP_READ_BUFFER(unsigned int highB);
```

- LSM303DLHC.h

```

/* ***** */
/* ***** MAGNETÔMETRO ***** */
/* ***** */

/* REGISTER ADDRESS MAPPING */

#define MAG_CRA_REG_M          0x00
#define MAG_CRB_REG_M          0x01
#define MAG_MR_REG_M           0x02
#define MAG_OUT_X_H_M           0x03

#define MAG_OUT_X_L_M           0x04
#define MAG_OUT_Z_H_M           0x05
#define MAG_OUT_Z_L_M           0x06
#define MAG_OUT_Y_H_M           0x07

#define MAG_OUT_Y_L_M           0x08
#define MAG_SR_REG_Mg           0x09
#define MAG_IRA_REG_M           0x0A
#define MAG_IRB_REG_M           0x0B

#define MAG_IRC_REG_M           0x0C
#define MAG_TEMP_OUT_H_M        0x31
#define MAG_TEMP_OUT_L_M        0x32

/* CRA_REG_M BIT DEFINITIONS */

#define MAG_TEMP_EN             0x80
#define MAG_DO2                  0x10
#define MAG_DO1                  0x08
#define MAG_DO0                  0x04

/* CRB_REG_M BIT DEFINITIONS */

#define MAG_GN2                  0x80
#define MAG_GN1                  0x40
#define MAG_GN0                  0x20

/* MR_REG_M BIT DEFINITIONS */

#define MAG_MD1                  0x02
#define MAG_MD0                  0x01

/* SR_REG_M BIT DEFINITIONS */

#define MAG_LOCK                 0x02
#define MAG_DRDY                 0x01

/* TEMP_OUT_H_M BIT DEFINITIONS */

#define MAG_TEMP11               0x80
#define MAG_TEMP10               0x40
#define MAG_TEMP9                0x20

```

```

#define MAG_TEMP8                0x10
#define MAG_TEMP7                0x08
#define MAG_TEMP6                0x04
#define MAG_TEMP5                0x02
#define MAG_TEMP4                0x01

/* TEMP_OUT_H_L BIT DEFINITIONS */

#define MAG_TEMP3                0x80
#define MAG_TEMP2                0x40
#define MAG_TEMP1                0x20
#define MAG_TEMP0                0x10

/* ***** */
/* ***** ACCELERÔMETRO ***** */
/* ***** */

/* REGISTER ADDRESS MAPPING */

#define M_ACC_CTRL_REG1_A        0x20
#define M_ACC_CTRL_REG2_A        0x21
#define M_ACC_CTRL_REG3_A        0x22
#define M_ACC_CTRL_REG4_A        0x23

#define M_ACC_CTRL_REG5_A        0x24
#define M_ACC_CTRL_REG6_A        0x25
#define M_ACC_REFERENCE_A        0x26
#define M_ACC_STATUS_REG_A       0x27

#define M_ACC_OUT_X_L_A          0x28
#define M_ACC_OUT_X_H_A          0x29
#define M_ACC_OUT_Y_L_A          0x2A
#define M_ACC_OUT_Y_H_A          0x2B

#define M_ACC_OUT_Z_L_A          0x2C
#define M_ACC_OUT_Z_H_A          0x2D
#define M_ACC_FIFO_CTRL_REG_A    0x2E
#define M_ACC_FIFO_SRC_REG_A     0x2F

#define M_ACC_INT1_CFG_A          0x30
#define M_ACC_INT1_SOURCE_A       0x31
#define M_ACC_INT1_THS_A          0x32
#define M_ACC_INT1_DURATION_A     0x33

#define M_ACC_INT2_CFG_A          0x34
#define M_ACC_INT2_SOURCE_A       0x35
#define M_ACC_INT2_THS_A          0x36
#define M_ACC_INT2_DURATION_A     0x37

#define M_ACC_CLICK_CFG_A         0x38
#define M_ACC_CLICK_SRC_A         0x39
#define M_ACC_CLICK_THS_A         0x3A
#define M_ACC_TIME_LIMIT_A        0x3B

#define M_ACC_TIME_LATENCY_A      0x3C
#define M_ACC_TIME_WINDOW_A       0x3D

/* CTRL_REG1_A BIT DEFINITIONS */

#define M_ACC_ODR3                0x80

```

```

#define M_ACC_ODR2          0x40
#define M_ACC_ODR1          0x20
#define M_ACC_ODR0          0x10
#define M_ACC_LPen          0x08
#define M_ACC_Zen           0x04
#define M_ACC_Yen           0x02
#define M_ACC_Xen           0x01

```

```
/* CTRL_REG2_A BIT DEFINITIONS */
```

```

#define M_ACC_HPM1          0x80
#define M_ACC_HPM0          0x40
#define M_ACC_HPCF2         0x20
#define M_ACC_HPCF1         0x10
#define M_ACC_FDS           0x08
#define M_ACC_HPCLICK       0x04
#define M_ACC_HPIS2         0x02
#define M_ACC_HPIS1         0x01

```

```
/* CTRL_REG3_A BIT DEFINITIONS */
```

```

#define M_ACC_I1_CLICK      0x80
#define M_ACC_I1_AOI1       0x40
#define M_ACC_I1_AOI2       0x20
#define M_ACC_I1_DRDY1      0x10
#define M_ACC_I1_DRDY2      0x08
#define M_ACC_I1_WTM        0x04
#define M_ACC_I1_OVERRUN    0x02

```

```
/* CTRL_REG4_A BIT DEFINITIONS */
```

```

#define M_ACC_BDU           0x80
#define M_ACC_BLE           0x40
#define M_ACC_FS1           0x20
#define M_ACC_FS0           0x10
#define M_ACC_HR            0x08
#define M_ACC_SIM           0x01

```

```
/* CTRL_REG5_A BIT DEFINITIONS */
```

```

#define M_ACC_BOOT          0x80
#define M_ACC_FIFO_EN       0x40
#define M_ACC_LIR_INT1      0x08
#define M_ACC_D4D_INT1      0x04
#define M_ACC_LIR_INT2      0x02
#define M_ACC_D4D_INT2      0x01

```

```
/* CTRL_REG6_A BIT DEFINITIONS */
```

```

#define M_ACC_I2_CLICKen    0x80
#define M_ACC_I2_INT1       0x40
#define M_ACC_I2_INT2       0x20
#define M_ACC_BOOT_I1       0x10
#define M_ACC_P2_ACT        0x08
#define M_ACC_H_LACTIVE     0x02

```

```
/* REFERENCE_A BIT DEFINITIONS */
```

```
#define M_ACC_Ref7          0x80
#define M_ACC_Ref6          0x40
#define M_ACC_Ref5          0x20
#define M_ACC_Ref4          0x10
#define M_ACC_Ref3          0x08
#define M_ACC_Ref2          0x04
#define M_ACC_Ref1          0x02
#define M_ACC_Ref0          0x01
```

```
/* STATUS_REG_A BIT DEFINITIONS*/
```

```
#define M_ACC_ZYXOR          0x80
#define M_ACC_ZOR            0x40
#define M_ACC_YOR            0x20
#define M_ACC_XOR            0x10
#define M_ACC_ZYXDA          0x08
#define M_ACC_ZDA            0x04
#define M_ACC_YDA            0x02
#define M_ACC_XDA            0x01
```

```
/* FIFO_CTRL_REG_A BIT DEFINITIONS */
```

```
#define M_ACC_FM1            0x80
#define M_ACC_FM0            0x40
#define M_ACC_TR             0x20
#define M_ACC_FTH4           0x10
#define M_ACC_FTH3           0x08
#define M_ACC_FTH2           0x04
#define M_ACC_FTH1           0x02
#define M_ACC_FTH0           0x01
```

```
/* FIFO_SRC_REG_A BIT DEFINITIONS */
```

```
#define M_ACC_WTM            0x80
#define M_ACC_OVRN_FIFO      0x40
#define M_ACC_EMPTY          0x20
#define M_ACC_FSS4           0x10
#define M_ACC_FSS3           0x08
#define M_ACC_FSS2           0x04
#define M_ACC_FSS1           0x02
#define M_ACC_FSS0           0x01
```

```
/* INT1_CFG_A BIT DEFINITIONS */
```

```
#define M_ACC_AOI            0x80
#define M_ACC_6D             0x40
#define M_ACC_ZHIE/ZUPE      0x20
#define M_ACC_ZLIE/ZDOWNE    0x10
#define M_ACC_YHIE/YUPE      0x08
#define M_ACC_YLIE/YDOWNE    0x04
#define M_ACC_XHIE/XUPE      0x02
#define M_ACC_XLIE/XDOWNE    0x01
```

```
/* INT1_SRC_A BIT DEFINITIONS */
```

```

#define M_ACC_IA          0x40
#define M_ACC_ZH          0x20
#define M_ACC_ZL          0x10
#define M_ACC_YH          0x08
#define M_ACC_YL          0x04
#define M_ACC_XH          0x02
#define M_ACC_XL          0x01

```

```

/* INT1_THS_A BIT DEFINITIONS */

```

```

#define M_ACC_THS6        0x40
#define M_ACC_THS5        0x20
#define M_ACC_THS4        0x10
#define M_ACC_THS3        0x08
#define M_ACC_THS2        0x04
#define M_ACC_THS1        0x02
#define M_ACC_THS0        0x01

```

```

/* INT1_DURATION_A BIT DEFINITIONS */

```

```

#define M_ACC_D6          0x40
#define M_ACC_D5          0x20
#define M_ACC_D4          0x10
#define M_ACC_D3          0x08
#define M_ACC_D2          0x04
#define M_ACC_D1          0x02
#define M_ACC_D0          0x01

```

```

/* INT2_CFG_A BIT DEFINITIONS */

```

```

#define M_ACC_AOI         0x80
#define M_ACC_6D          0x40
#define M_ACC_ZHIE        0x20
#define M_ACC_ZLIE        0x10
#define M_ACC_YHIE        0x08
#define M_ACC_YLIE        0x04
#define M_ACC_XHIE        0x02
#define M_ACC_XLIE        0x01

```

```

/* INT2_SRC_A BIT DEFINITIONS */

```

```

#define M_ACC_IA          0x40
#define M_ACC_ZH          0x20
#define M_ACC_ZL          0x10
#define M_ACC_YH          0x08
#define M_ACC_YL          0x04
#define M_ACC_XH          0x02
#define M_ACC_XL          0x01

```

```

/* INT2_THS_A BIT DEFINITIONS */

```

```

#define M_ACC_THS6        0x40
#define M_ACC_THS5        0x20
#define M_ACC_THS4        0x10
#define M_ACC_THS3        0x08

```

```

#define M_ACC_THS2          0x04
#define M_ACC_THS1          0x02
#define M_ACC_THS0          0x01

```

```

/* INT2_DURATION_A BIT DEFINITIONS */

```

```

#define M_ACC_D6            0x40
#define M_ACC_D5            0x20
#define M_ACC_D4            0x10
#define M_ACC_D3            0x08
#define M_ACC_D2            0x04
#define M_ACC_D1            0x02
#define M_ACC_D0            0x01

```

```

/* CLICK_CFG_A BIT DEFINITIONS */

```

```

#define M_ACC_ZD            0x20
#define M_ACC_ZS            0x10
#define M_ACC_YD            0x08
#define M_ACC_YS            0x04
#define M_ACC_XD            0x02
#define M_ACC_XS            0x01

```

```

/* CLICK_SRC_A BIT DEFINITIONS */

```

```

#define M_ACC_IA            0x40
#define M_ACC_DCLICK        0x20
#define M_ACC_SCLICK        0x10
#define M_ACC_Sign          0x08
#define M_ACC_Z              0x04
#define M_ACC_Y              0x02
#define M_ACC_X              0x01

```

```

/* CLICK_THS_A BIT DEFINITIONS */

```

```

#define M_ACC_Ths6          0x40
#define M_ACC_Ths5          0x20
#define M_ACC_Ths4          0x10
#define M_ACC_Ths3          0x08
#define M_ACC_Ths2          0x04
#define M_ACC_Ths1          0x02
#define M_ACC_Ths0          0x01

```

```

/* TIME_LIMIT_A BIT DEFINITIONS */

```

```

#define M_ACC_TLI6          0x40
#define M_ACC_TLI5          0x20
#define M_ACC_TLI4          0x10
#define M_ACC_TLI3          0x08
#define M_ACC_TLI2          0x04
#define M_ACC_TLI1          0x02
#define M_ACC_TLI0          0x01

```

```

/* TIME_LATENCY_A BIT DEFINITIONS */

```

```

#define M_ACC_TLA7          0x80

```

```
#define M_ACC_TLA6      0x40
#define M_ACC_TLA5      0x20
#define M_ACC_TLA4      0x10
#define M_ACC_TLA3      0x08
#define M_ACC_TLA2      0x04
#define M_ACC_TLA1      0x02
#define M_ACC_TLA0      0x01
```

```
/* TIME_WINDOW_A BIT DEFINITIONS */
```

```
#define M_ACC_TW7      0x80
#define M_ACC_TW6      0x40
#define M_ACC_TW5      0x20
#define M_ACC_TW4      0x10
#define M_ACC_TW3      0x08
#define M_ACC_TW2      0x04
#define M_ACC_TW1      0x02
#define M_ACC_TW0      0x01
```



## **APÊNDICE C – Código fonte em MATLAB**



```

% Plot de vetores %

clc
close all

set(0,'DefaultFigureWindowStyle','docked');

[az, el] = view;

F = fopen('C:\Users\Rafael\Dropbox\TCC\capture.txt');
lim = 1;

while (1)
    fseek(F,-82,'eof');
    A = fread(F, 80, 'uint8=>char');
    A = A';
    A = str2num(A);
    if (length(A) == 9)

        vetor1 = A(1:3);
        vetor2 = A(4:6);
        vetor3 = A(7:9);

        quiver3(0,0,0,1,0,0,'LineWidth',2,'Color',[.7,.7,.7])
        view (az, el);
        hold on;

        [x y z] = sphere(40);
        h = surfl(x, y, z);
        set(h, 'FaceAlpha', .2)
        shading interp

        quiver3(0,0,0,0,1,0,'LineWidth',2,'Color',[.7,.7,.7])
        quiver3(0,0,0,0,0,1,'LineWidth',2,'Color',[.7,.7,.7])

        quiver3(0,0,0,vetor1(1),vetor1(2),vetor1(3),'LineWidth',3,'Color','b')
        quiver3(0,0,0,vetor2(1),vetor2(2),vetor2(3),'LineWidth',3,'Color','g')
        quiver3(0,0,0,vetor3(1),vetor3(2),vetor3(3),'LineWidth',3,'Color','r')
        hold off;
        axis([-lim lim -lim lim -lim lim]);
        axis square
        drawnow;
        [az, el] = view;
    end
end

```



## **APÊNDICE D – Desenhos da mesa de testes**



As figuras D.1 e D.2 mostram o projeto da estrutura, e a Figura D.3 mostra o projeto da peça de acoplamento entre o motor e a plataforma.

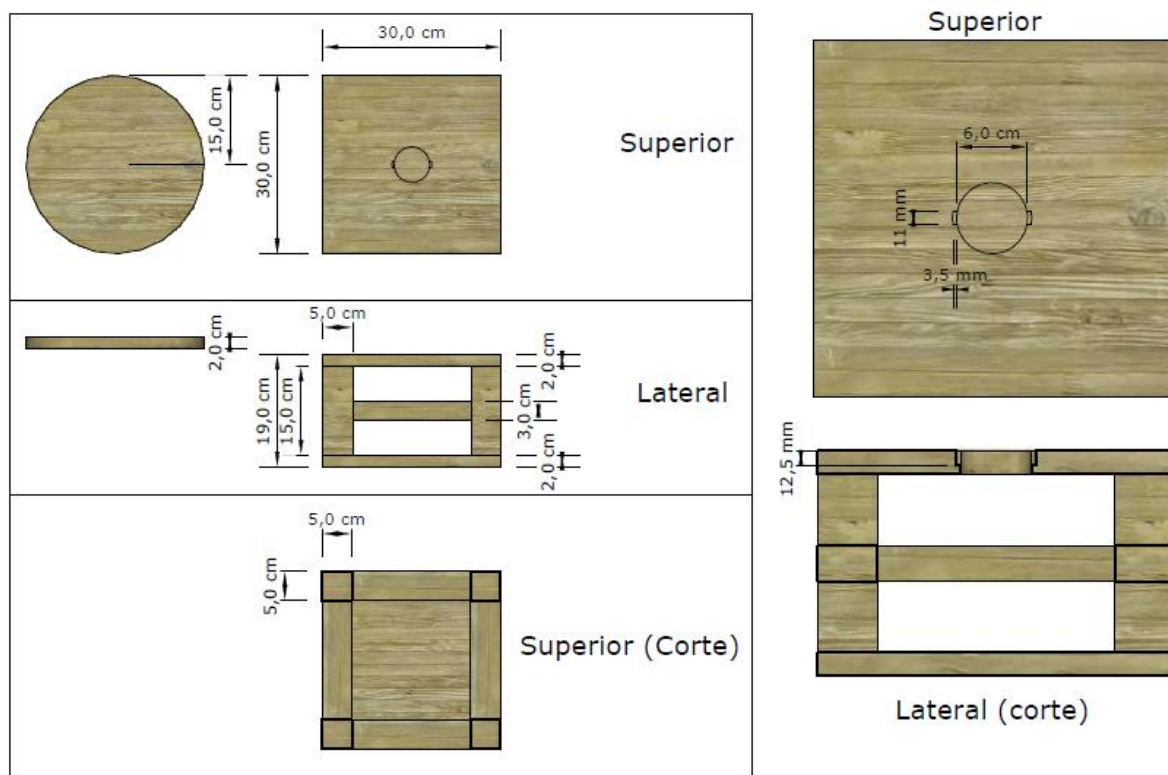


Figura D.1: Projeto da mesa giratória.

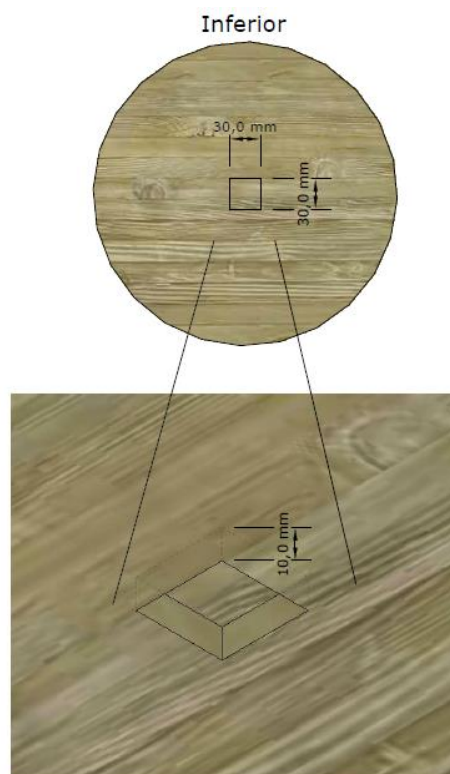


**Figura D.2: Detalhe do encaixe da peça de acoplamento na plataforma.**

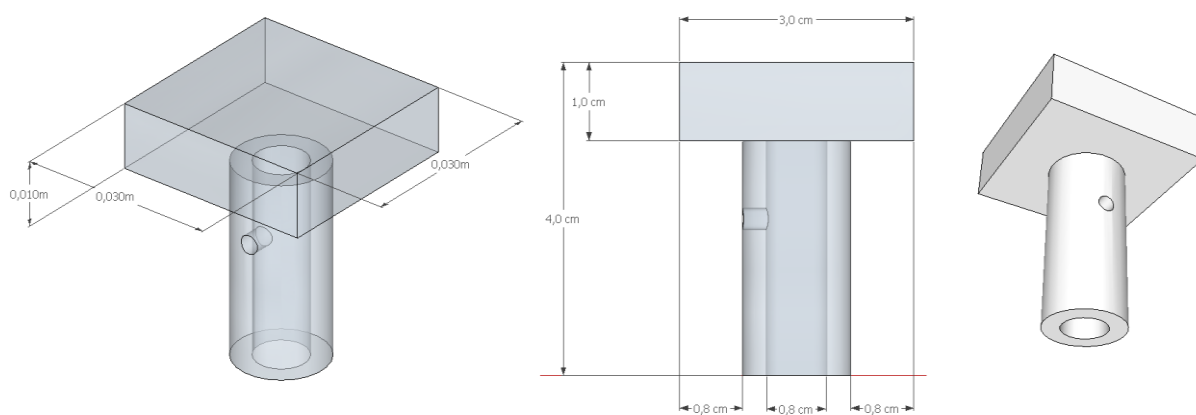


**Figura D.3: Projeto da peça de acoplamento.**

Fotos da mesa de testes e da peça de acoplamento são apresentadas nas figuras D.4, D.5 e D.6. Nas fotos pode-se observar que durante a construção da estrutura da mesa foram feitas adaptações para fixar o motor e a peça de acoplamento mais firmemente.



**Figura D.4:** Foto da mesa de testes.

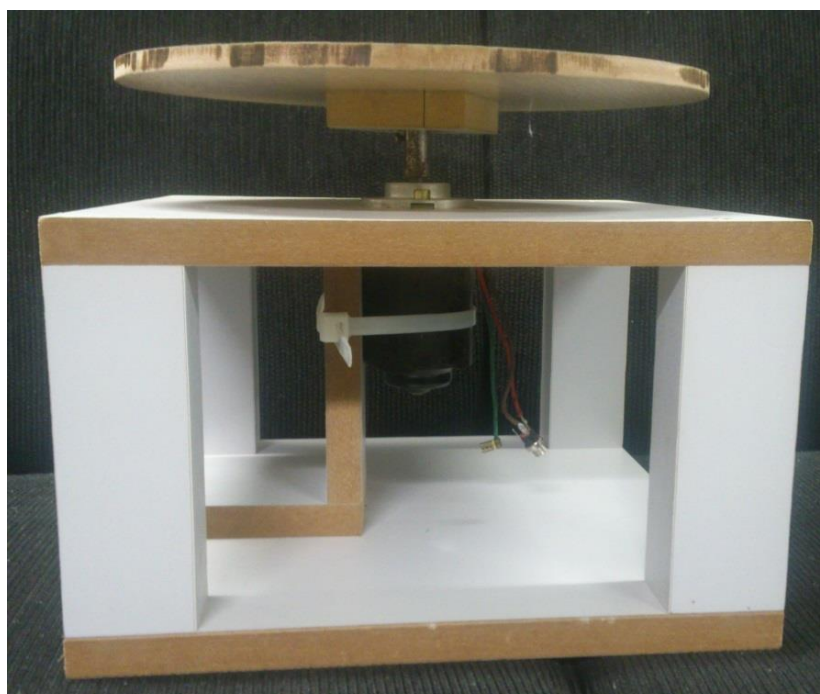


**Figura D.5:** Foto em perfil da mesa de testes.



**Figura D.6: Foto da peça de acoplamento.**