Pedro Henrique Dalmolin de Vasconcelos Affonso

Sistema de controle de atitude de baixo custo usando rodas de reação

São Carlos
2014

Pedro Henrique Dalmolin de Vasconcelos Affonso

Sistema de controle de atitude de baixo custo usando rodas de reação

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

Curso de Engenharia de Computação

ORIENTADOR: Dr. Carlos Dias Maciel

São Carlos
2014

A257s

Affonso, Pedro Henrique Dalmolin de Vasconcelos
Sistema de controle de atitude de baixo custo usando rodas de reação / Pedro Henrique Dalmolin de Vasconcelos Affonso; orientador Carlos Dias Maciel. São Carlos, 2014.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Controle de Atitude. 2. Rodas de Reação. 3. Android. 4. Arduino. I. Título.

FOLHA DE APROVAÇÃO

Nome: Pedro Henrique Dalmolin de Vasconcelos Affonso

Título: "Sistema de controle de atitude de baixo custo usando rodas de reação"

Trabalho de Conclusão de Curso defendido em 11 / 06 / 2014.

Comissão Julgadora:	**Resultado:**
Prof. Associado Carlos Dias Maciel (Orientador) - SEL/EESC/USP	APROVADO
Prof. Assistente Carlos Goldenberg SEL/EESC/USP	APROVADO
Prof. Dr. Marcelo Basílio Joaquim SEL/EESC/USP	APROVADO

Coordenador do Curso Interunidades - Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Resumo

Esse trabalho teve como objetivo desenvolver um sistema de baixo custo capaz de controlar uma roda de reação, dispositivo muito usado para controle de orientação de sistemas por meio da troca de momento angular. Esse sistema foi implementado usando um dispositivo Android equipado com um acelerômetro e um magnetômetro, ambos com 3 eixos, uma placa de desenvolvimento Arduino Uno e um transmissor/receptor Bluetooth. O sistema desenvolvido permite ajustar manualmente a velocidade do motor ou ainda fazer um controle automático de posição usando parâmetros do tipo PID ajustáveis. Esse sistema também possui todo o hardware, documentação e base de software para que ele possa ser usado em projetos futuros.

Palavras-chave: Controle de atitude, Rodas de reação, Android, Arduino

Abstract

This work's objective was to develop a low-cost control system for reaction wheels, which are devices often used for attitude control using angular momentum exchange. This system was implemented using an Android device equipped with an accelerometer and a magnetometer, both with 3-axis sensing capabilities, an Arduino Uno development board and a Bluetooth transmitter/receiver. The developed system allows manual adjustment of the motor speed, and also the automatic control of the position using adjustable PID parameters. This system contains all the hardware, documentation and software base for use in future projects.

Keywords: Attitude control, Reaction Wheels, Android, Arduino

Sumário

Lista de Abreviaturas

AOSP - *Android Open Source Project* ou Projeto de Código Aberto Android

API - *Application Programming Interface* ou Interface de Programação de Aplicações

CC - Corrente Contínua

CRR - Conjunto de Rodas de Reação (Equivalente ao termo em inglês *Reaction Wheel Assembly*)

E/S - Entrada e Saída

IEEE - Instituto de Engenheiros Eletricistas e Eletrônicos

IDE - *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado

JVM - *Java Virtual Machine* ou Máquina Virtual Java

PID - Proporcional, Integrativo e Derivativo

PWM - *Pulse Width Modulation* ou Modulação em Largura de Pulso

RR - Roda de Reação

SPP - *Serial Port Protocol* ou Protocolo de Porta Serial

UART - *Universal Asynchronous Receiver/Transmitter*

USB - *Universal Serial Bus* ou Barramento Serial Universal

VE - Veículo Espacial

1. Introdução

Rodas de reação (RRs) são atuadores muito estudados em sistemas de controle há várias décadas (Chobotov, 1991). Sua principal aplicação é o controle de atitude de satélites e outros veículos espaciais, porém também são muito usadas para estudos e verificação de novas técnicas de controle de sistemas (Muehlebach, Mohanarajah e D'Andrea, 2013).

A atitude de um veículo espacial consiste na orientação do mesmo em relação a um planeta ou outro referencial, como a Terra por exemplo. O controle de atitude é fundamental para alinhar diversos dispositivos, como câmeras, painéis solares e antenas de comunicação, em veículos espaciais com diferentes missões. Um satélite para monitoramento terrestre, por exemplo, deve ter suas câmeras constantemente apontadas em direção à terra, e está sujeito a várias perturbações e torques externos que podem interferir em sua atitude e devem ser compensadas (Chobotov, 1991).

As rodas de reação são frequentemente utilizadas em satélites para estabelecer a orientação desejada, embora existam muitos outros métodos que podem ser usados conjuntamente ou separadamente. Alguns que podemos citar são: estabilização passiva por gradiente de gravidade, estabilização por spin, bobinas magnéticas torqueadoras, giroscópios de momento de controle e propulsores (Sidi, 1995).

Muitas universidades desenvolvem, para fins acadêmicos e de pesquisa, os chamados CubeSats, satélites de proporções muito pequenas e padronizadas (The CubeSat Program, 2009) que podem ser lançados no espaço por um custo relativamente baixo (Heidt, 2000). Isso se dá principalmente porque o custo do lançamento é menor para satélites pequenos e leves, e porque vários CubeSats podem ser colocados em órbita por um único lançamento (Puig-Suari, Turner e Twiggs, 2001). No entanto, o custo de componentes para montar tais satélites podem chegar a dezenas ou centenas de milhares de reais (ISIS, 2006), o que é uma grande barreira para que se possa desenvolver estudos na área.

A proposta desse projeto é desenvolver o protótipo de uma plataforma inercial para estudar o controle de atitude usando rodas de reação em um único eixo, uma grande simplificação em relação ao controle de atitude de satélites, que na maioria dos casos possuem rodas de reação em três eixos.

Os objetivos específicos desse projeto são:

1. Estudar a arquitetura dos sistemas embarcados;

2. Desenvolver interfaces externas de hardware dedicado com os sistemas Android;

3. Desenvolver um sistema inercial de roda de reação em um eixo.

2. Teoria

Nesse capítulo serão apresentados, com a finalidade de permitir uma maior compreensão do detalhamento técnico das atividades realizadas e resultados apresentados nos capítulos seguintes, alguns tópicos utilizados no desenvolvimento deste trabalho. Os itens 2.1 a 2.4 apresentam fundamentos matemáticos e conceitos usados no desenvolvimentos e necessários para o entendimento do projeto. Os itens 2.5, 2.6 e 2.7 apresentam as principais tecnologias empregadas - comunicação Bluetooth e as plataformas Android e Arduino.

2.1 Representações da atitude

Uma das maneiras mais usadas de representar uma rotação, a orientação de um corpo rígido no espaço tridimensional ou para descrever a orientação de um sistema de coordenadas em relação a outro é através dos ângulos de Euler (Sidi, 1995). No espaço tridimensional são necessários três parâmetros para especificar uma orientação. Os ângulos de Euler são três valores ($\alpha, \beta \, e \, \gamma$) que representam uma sequência de rotações em torno dos eixos do sistema de coordenadas do corpo, sendo a primeira e a última rotações em torno do mesmo eixo. Um exemplo seria uma sequência de rotações em torno dos eixos z, y, e z. Qualquer orientação pode ser alcançada por meio dessas três rotações, porém a representação não é única para alguns casos. Se o segundo ângulo for igual a zero, ocorre uma singularidade em que o primeiro e o terceiro não podem ser determinados individualmente, podendo ser determinada apenas a soma dos dois, pois ambos passam a representar rotações sobre o mesmo eixo no sistema de coordenadas de referência.

A representação pelos ângulos de *roll*, *pitch* e *yaw* também é muito usada (Craig, 2005). Ela é muito semelhante aos ângulos de Euler, porém a sequência dos eixos das rotações é x-y-z. O primeiro ângulo é denominado *roll*, o segundo *pitch*, e o terceiro *yaw*, sendo representados por θ, ϕ e ψ respectivamente. Essa representação também apresenta o mesmo problema de singularidade quando $\phi = 90°$ devido ao alinhamento dos eixos de rotação.

Para resolver o problema da unicidade das representações para esses casos particulares, podem ser usados alguns outros tipos de representação. Um deles é a representação eixo-ângulo. É possível demonstrar que qualquer orientação no espaço tridimensional pode ser descrita por uma única rotação em torno de algum eixo. Essa representação consiste em quatro parâmetros: três especificando um vetor (eixo), e um quarto especificando um ângulo (Sidi, 1995).

Outra representação é com o uso de quaterniões, que são um conjunto de números compostos de quatro números reais e que se assemelha ao conjunto dos números complexos, sendo que se definem operações aritméticas que podem ser usadas para representar rotações (Hamilton, 1844).

Uma última representação ainda possível são matrizes de rotação que, da mesma maneira que os ângulos de Euler, representam uma única rotação em torno de algum eixo. Essas matrizes têm a

desvantagem de usar 9 elementos para a representação, porém são frequentemente usadas dado que uma rotação, ou ainda a associação de uma sequência de rotações podem ser representadas por uma multiplicação matricial (Craig, 2005).

2.2 Rodas de Reação

Uma roda de reação consiste, basicamente, em um motor ligado a um disco ou anel e a um controlador, com a finalidade de trocar momento angular com o veículo e armazená-lo. Chama-se o conjunto formado pela roda, o motor e os componentes responsáveis pelo controle de Conjunto de Roda de Reação (CRR) (Sidi, 1995). O seu princípio de funcionamento é análogo à terceira lei de Newton: um torque sobre um corpo gera um torque igual, porém no sentido oposto, em outro corpo. Portanto um torque gerado sobre a roda, que é ligada ao rotor do motor, gera um torque no sentido oposto no estator, ligado ao corpo do satélite. A roda em si pode ser vista como um componente que armazena momento angular (Gajamohan, Merz, Thommen e D'Andrea, 2012).

Uma vantagem interessante desse princípio é que ele permite que um VE realize movimentos por meio de um atuador sem perda de massa, como ocorre por exemplo com os jatos propulsores, muito usados em VEs (Sidi, 1995). Dessa maneira seria possível, em teoria, operá-la por um tempo indefinido desde que haja energia disponível.

Pode-se citar a bobina torqueadora (ou magnetorquer), que usa o campo magnético da terra para gerar torque, como um atuador que possui essa mesma característica. A roda de reação, no entanto, apresenta algumas vantagens em relação às bobinas. Ela permite movimentos mais rápidos e precisos, e não depende do campo magnético da terra - bobinas são mais adequadas para órbitas terrestres baixas, e são ainda mais lentas e menos eficientes para órbitas de maior altitude como as geoestacionárias (Sidi, 1995).

Torques externos em um VE podem fazer com que seja necessário aumentar gradualmente a velocidade angular das rodas para se manter uma orientação fixa. O maior problema das RRs é que elas possuem uma velocidade máxima à qual elas podem chegar; quando chegam nessa velocidade, dizemos que elas estão saturadas, e elas não podem armazenar mais momento angular e se perde a capacidade de controle da orientação. É necessário, portanto, que se realize um descarregamento do momento antes que se chegue à saturação (Sidi, 1995). Por esse motivo, as rodas nunca podem ser o único atuador do sistema para o controle de atitude, sendo frequentemente usado com bobinas torqueadoras e/ou jatos propulsores (Chobotov, 1991).

VEs muito frequentemente são equipados com CRRs capazes de atuar em três eixos, e geralmente contendo três ou quatro rodas. Três das rodas são montadas perpendiculares umas às outras para permitir o controle em três eixos (Chobotov, 1991). A quarta roda ocorre como fator de

redundância para tolerância a falhas em alguma das outras três, sendo montada em um eixo que não é paralelo ao eixo de nenhuma das outras. Dessa maneira a quarta roda pode reparar uma falha em qualquer uma delas.

Figura 1 - Sistema de controle com rodas de reação em três eixos disponível comercialmente no site CubeSatShop.com (ISIS, 2006)

2.3 Descrição da dinâmica e equações gerais do motor

O estudo da dinâmica de VEs geralmente é feito usando cálculos vetoriais. No entanto, como nosso sistema atua em um único eixo, pode-se usar uma matemática bastante simples para descrever o problema, como é feito a seguir.

O momento angular h de uma partícula com massa m em um movimento rotacional em torno de um eixo com velocidade angular ω pode ser definido como

$$h = \omega \cdot m \cdot r^2$$

sendo r a distância entre a partícula e o eixo (Halliday, Resnick, & Walker, 2009). Consideremos agora um caso semelhante, porém levando em conta um corpo qualquer em vez de uma simples partícula. O momento angular desse corpo é

$$h = \omega \cdot I \tag{1}$$

onde I é uma grandeza chamada momento de inércia, que representa o grau de dificuldade em se alterar a velocidade angular de um corpo. O momento de inércia é dependente da geometria e massa do corpo, e também do eixo no qual ocorre o movimento rotacional. Ela é calculada pela integral em coordenadas cilíndricas

$$I = \int_C r^2 dm \tag{2}$$

assumindo-se que r é o raio entre um ponto arbitrário e o eixo de rotação, e C é o volume total do corpo em questão.

A variação temporal do momento de inércia é chamada torque e usaremos a notação

$$\tau = \dot{h} = \dot{\omega} \cdot I \tag{3}$$

A propriedade mais importante do momento de inércia é que ele se conserva, isto é, o momento de inércia de um sistema só muda se ocorrer uma transferência de momento de inércia com algum corpo externo a ele, ou seja, se um torque externo for aplicado. O princípio de funcionamento das rodas de reação é a troca de momento angular entre o corpo do satélite e as rodas. Na ausência de perturbação externa, a soma dos momentos angulares dos dois corpos se conserva (Sidi, 1995):

$$\dot{h}_s + \dot{h_r} = 0. \tag{4}$$

Isto significa também que para aplicar um torque no corpo em algum eixo, um torque no sentido oposto deve ser produzido. Isso pode ser feito, por exemplo, por meio de um motor elétrico, como é o caso dos Conjuntos de Roda de Reação (CRR) sob estudo.

Um motor de corrente contínua pode ser modelado simplificadamente e de maneira linear usando um pequeno conjunto de equações, que são dadas a seguir:

$$V = Ri + K_e\omega + L\frac{di}{dt} \tag{5}$$

$$I\dot{\omega} = K_t i + K_b\omega + \tau_{ext} \tag{6}$$

$$\frac{d\theta}{dx} = \omega \tag{7}$$

sendo que as variáveis usadas são:

V: tensão de armadura, ou a tensão aplicada nos terminais do motor

R: a resistência interna da bobina do motor

L: indutância da bobina

K_t: constante de torque

K_e: constante eletromotriz

K_b: constante de amortecimento (ou atrito)

ω: velocidade angular entre o rotor e o estator

θ: ângulo sobre o eixo de rotação entre um sistema de coordenadas fixo no rotor e outro no estator

τ_{ext}: o torque externo aplicado ao sistema

I: momento de inércia do rotor

Para o modelo (Sidi, 1995), usa-se uma simplificação adicional e é possível desprezar a indutância da bobina e consequentemente o terceiro termo da primeira equação do conjunto, ficando apenas com

$$V = Ri + K_e\omega. \tag{8}$$

Esse termo é desprezado nessa modelagem porque sabemos que o polo dominante do sistema será um associado à parte mecânica do sistema, embora a análise completa não seja apresentada aqui, devido ao fato de termos um valor relativamente alto para o momento de inércia I - pois quanto maior for esse termo, maior é a capacidade da roda de armazenar momento angular.

Um motor de corrente contínua pode ser modelado também pelo diagrama de blocos equivalente (Ogata, 1970), mostrado na Figura 2. Já em um satélite, devemos considerar a conservação do momento angular e o torque gerado pelo motor no sentido oposto, como pode ser visto na Figura 3 (Sidi, 1995).

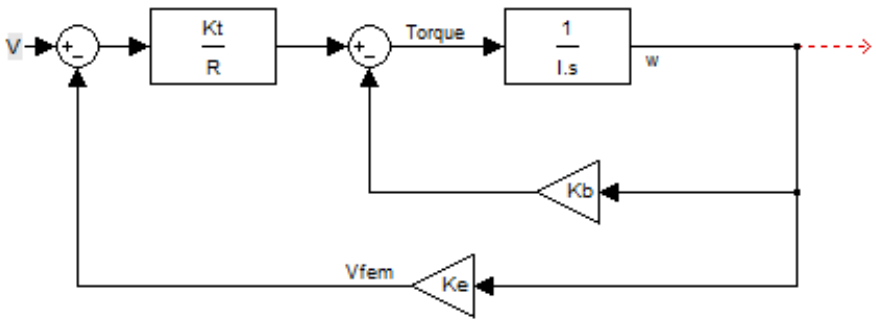

Figura 2 - Diagrama de blocos de um motor de corrente contínua. Esse modelo é equivalente ao conjunto das equações (2) e (4) (omitindo-se o torque externo). A equação (2) é representada pela malha fechada mais à direita, onde ocorre a realimentação de ω através de um bloco de ganho K_b, e a equação (4) é representada pelos demais blocos.

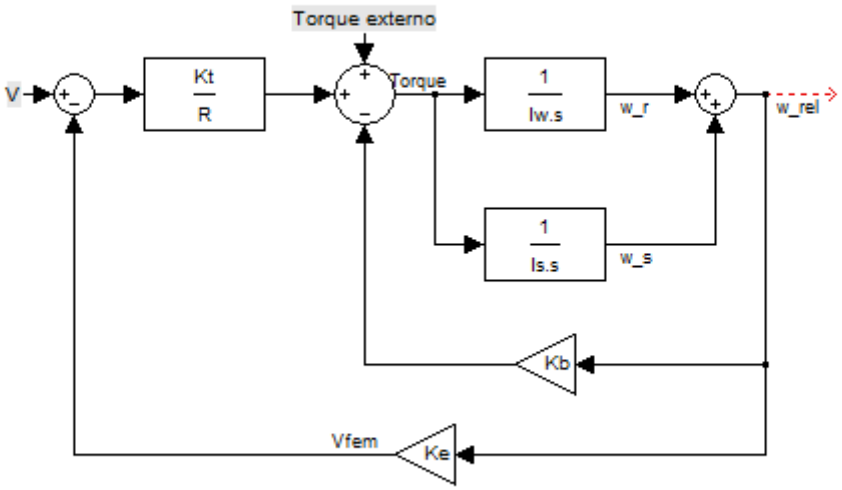

Figura 3 - Diagrama de blocos de um veículo espacial com roda de reação e sujeito a perturbações externas. Em relação ao diagrama da figura 2, a maior diferença é que agora leva-se em conta o momento de inércia I_s do VE além do momento de inércia I_w da roda de reação, pois o torque gerado pelo motor agora terá uma ação sobre um deles e uma reação de mesma intensidade e sentido oposto sobre o outro. Isso ocorre porque no caso anterior foi considerado que o estator do motor CC era fixo (Sidi, 1995).

Uma das maneiras de controlar esse motor é usando o esquema da Figura 4, embora (Sidi, 1995) apresente um modelo mais complexo, usando controle de torque e realimentação de corrente.

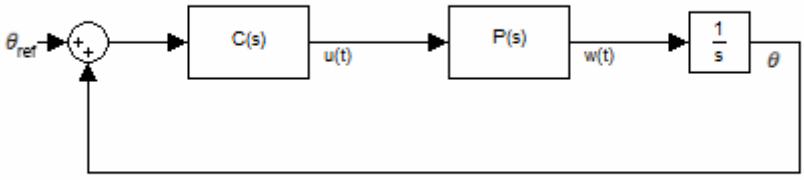

Figura 4 - Sistema de controle em malha fechada para um sistema com roda de reação. C(s) representa um controlador genérico e P(s) representa a planta. Na verdade, essa é uma configuração básica de malha fechada, sendo amplamente usada nas mais variadas aplicações.

2.4 Controle PID

Em um sistema de controle, um dos elementos de estudo principais é o controlador. Ele é geralmente tem como entrada o sinal de erro - obtido pela subtração do sinal de entrada e da realimentação dos sensores - e sua saída ligada aos atuadores da planta. A Figura 4 ilustra esse conceito.

Os controladores proporcional, integrativo e derivativo, ou simplesmente PID, são um tipo de controlador amplamente utilizado em aplicações industriais e são caracterizados por uma função de transferência da forma

$$C(s) = k_p + \frac{k_i}{s} + k_d s \tag{9}$$

onde k_p, k_i e k_s geralmente são parâmetros ajustáveis . O que essa função de transferência diz é que o controlador PID possui um integrador e um derivador, e a sua saída consiste em uma combinação linear do sinal de entrada, da sua derivada e da sua integral. Existem diversas técnicas e métodos computacionais para o ajuste dos parâmetros PID de acordo com o comportamento e desempenho desejado para o sistema (Ogata, 1970).

2.5 Bluetooth

O **Bluetooth** é um padrão que define um conjunto bastante extenso de protocolos para comunicações sem fio para comunicação em curtas distâncias. Ele usa para transmissão a faixa de frequência dos 2.4 a 2.485GHz, dividindo-a em 79 canais de 1MHz de largura. Diferente de outros padrões como os IEEE 802.11, uma rede **Bluetooth** não transfere dados em um único canal pré-determinado. É usado uma tecnologia chamada *Frequency Hopping Spread Spectrum*, em que a portadora é deslocada periodicamente em intervalos de 625μs, para um canal diferente escolhido pseudoaleatoriamente. Isso é importante porque a faixa de frequências do **Bluetooth** é compartilhada com outros dispositivos. Dessa forma as possíveis inferferências são minimizadas de forma evasiva (Huang e Rudolph, 2007).

O padrão **Bluetooth** define uma pilha de protocolos em várias camadas, e também diversos Perfis (ou *Profiles*) que serão explicados adiante, sendo que existem livros inteiros sobre o assunto. Portanto serão citadas e descritas aqui apenas as partes fundamentais para o entendimento do projeto, e sem muitos detalhes.

Os **Perfis Bluetooth** são especificações que vão além de protocolos de transporte e métodos de comunicação e definem maneiras de realizar tarefas de mais alto nível. Em outras palavras, cada perfil é uma solução de comunicação para uma aplicação padronizada e completa no sentido de especificar toda a pilha de protocolos, em todas as camadas. Existe uma quantidade grande de perfis Bluetooth associados a diferentes aplicações. Uma vantagem da existência desses perfis é permitir compatibilidade entre dispositivos de hardware com funções semelhantes (Huang e Rudolph, 2007).

Exemplos de perfis são o *Human Interface Device*, para dispositivos de entrada como teclados e mouses; o *Object Exchange* (OBEX) para transferência de arquivos; e o *Headset Profile* para

transferência de áudio em chamadas de voz. O nosso perfil de interesse é o que é suportado pelo nosso módulo **Bluetooth**, o *Serial Port Profile* (SPP) (Bluetooth SIG, 2014), que foi desenvolvido originalmente para que dois dispositivos possam atuar como substitutos de um cabo serial . Esses dois dispositivos são chamados de Dispositivo A, o que inicia a conexão; e dispositivo B, o que aguarda a conexão.

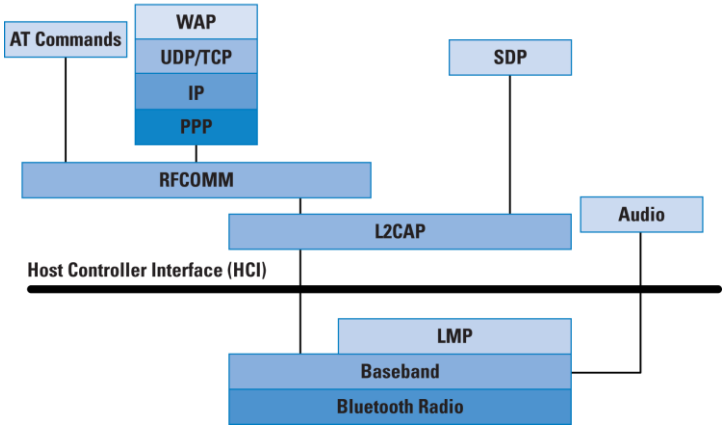

Figura 5 - Diagrama ilustrativo da pilha de protocolos Bluetooth. (Sridhar, 2008)

O SPP usa o protocolo RFCOMM (Bluetooth SIG, 2012), um protocolo de transporte de propósito geral, confiável e baseado em *streams*. As conexões RFCOMM são encapsuladas em conexões L2CAP (*Logial Link Control and Adaptation Protocol*), que é outro protocolo de transporte Bluetooth, que por sua vez também são encapsuladas em conexões ACL (*Assynchronous Connection-Less*), protocolo usado para casos em que a perda de pacotes não é desejável e não há necessidade de garantir uma baixa latência e largura de banda mínima, e que realiza retransmissão de pacotes corrompidos. A Figura 5 ilustra alguns dos protocolos **Bluetooth**.

Um ponto em que o **Bluetooth** difere de outros tipos de comunicação é que iniciar uma conexão é um processo de várias etapas e relativamente complicado para o desenvolvedor, e isso deve ser levado em conta ao escrever código utilizando essa tecnologia.

Considerando uma conexão entre dois dispositivos, pode-se chamar um de servidor e cliente - o primeiro é o que espera a conexão, e o outro o que inicia a conexão (essa terminologia será usada apenas para essa situação, não estando relacionada de maneira alguma a à arquitetura de aplicações cliente-servidor). O primeiro passo a ser tomado é a descoberta do dispositivo: o servidor deve estar configurado como descobrível, e periodicamente escuta por mensagens de pesquisa de dispositivos em um canal aleatório. O cliente por sua vez deve enviar mensagens de pesquisa de dispositivo em todos os canais. A descoberta ocorre quando o servidor está escutando mensagens no mesmo canal em que o cliente enviou. (Huang e Rudolph, 2007). Esse processo não é instantâneo para o usuário, podendo levar alguns segundos.

Depois disso, pode ser necessário o pareamento. O **Bluetooth** suporta autenticação e criptografia, e na primeira vez que dois dispositivos se conectam eles fazem o que é chamado de pareamento. Em dispositivos mais antigos é necessário digitar um segredo compartilhado chamado PIN em um ou ambos os dispositivos (em dispositivos sem teclado normalmente o PIN é um número estabelecido), porém o **Bluetooth** 2.1 especifica o *Simple Pairing*, que gera os PINS automaticamente. O PIN então é usado para gerar uma *link key*, que é armazenada em ambos os dispositivos para conexões futuras e é usada para criptografar os dados.

Depois disso, o servidor tem que aguardar conexões e o cliente tem que realizar uma descoberta de serviços (*Service Discovery*) usando o SDP (*Service Discovery Protocol*). A Figura 6 ilustra como ocorre a comunicação Bluetooth durante a descoberta de serviço. O protocolo SDP funciona, simplificadamente, da seguinte maneira: O servidor registra um Registro de Serviço em seu servidor SDP, um servidor que está sempre ativo em uma "porta" (fazendo uma analogia a redes TCP/IP) determinada, para que o cliente possa localizá-lo. O cliente pergunta ao servidor SDP se ele possui um Serviço ou Classe de Serviços específico, e se ele possuir a conexão é estabelecida usando uma "porta" que é então atribuída (Huang e Rudolph, 2007). Os identificadores de serviço devem ser números únicos, isto é, duas aplicações não deve nunca ter o mesmo identificador. Já identificadores de classe de serviço são compartilhados por dispositivos que fornecem a mesma funcionalidade, como por exemplo áudio sobre Bluetooth.

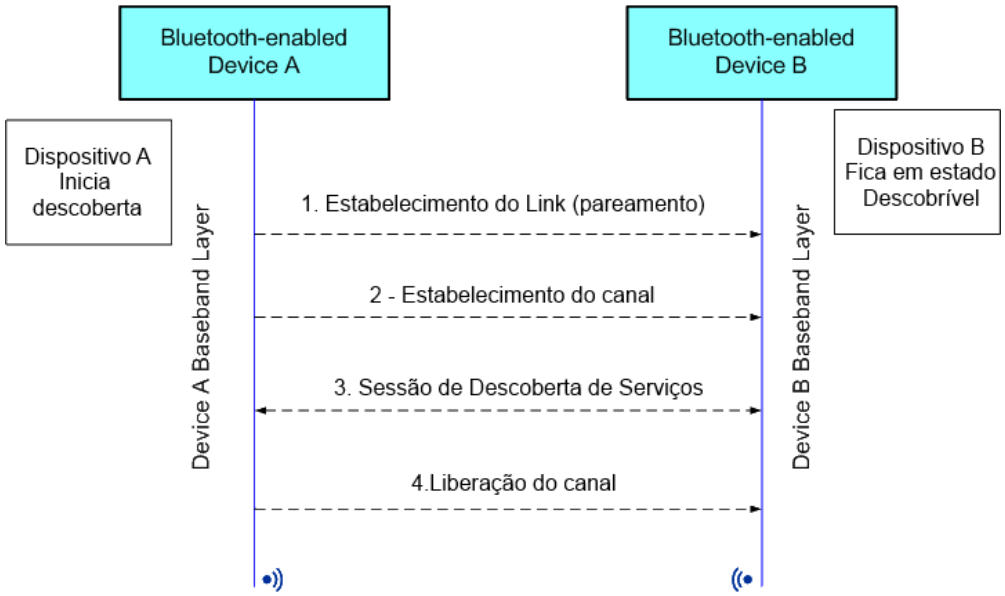

Figura 6 - Representação da comunicação Bluetooth durante a descoberta de serviços. (Microsoft, 2008)

2.6 Android

O Android é um sistema operacional baseado do núcleo do Linux e desenvolvido para ser usado principalmente em dispositivos móveis como *tablets* e *smartphones*.

Uma característica interessante do Android é que seu código fonte é distribuído sob a Licença Apache, ou seja, ele é um projeto software livre, e qualquer pessoa pode ter acesso ao código, distribuí-lo ou modifica-lo e distribuir a versão modificada (The Apache Software Foundation, 2004). Esse código é chamado muitas vezes de AOSP (*Android Open Source Project*) para evitar confundi-lo com as suas versões (*branches*) modificadas. A maioria das empresas que produzem dispositivos móveis com sistema Android os distribuem com suas próprias versões modificadas proprietárias, isto é, de código fechado. Também há uma grande comunidade de desenvolvedores independentes que produzem e distribuem suas próprias versões modificadas. Uma parte da comunidade de software livre critica o projeto Android por não adotar o modelo de desenvolvimento usado na maioria dos outros projetos de código aberto, em que o código é desenvolvido por uma comunidade, possuindo uma lista de discussão pública e um repositório com o código mais recente. Em vez disso, o Android é desenvolvido por funcionários do Google e o código e é mantido sob sigilo até que haja o lançamento de uma nova versão, que ocorre a cada 6 meses aproximadamente (Yaghmour, 2013).

Uma das grandes vantagens obtidas por escolher usar esse sistema operacional foi a de permitir o desenvolvimento de aplicativos que utilizem funcionalidades de hardware e comunicação - no caso desse trabalho, comunicação Bluetooth e obtenção de dados de sensores - de maneira transparente para o programador e independente da configuração de hardware do dispositivo. Dessa maneira, não ficamos presos a um modelo específico de smartphone ou tablet: seria possível utilizar o aplicativo desenvolvido com qualquer outro modelo, desde que ele tenha um acelerômetro e magnetômetro, e uma versão relativamente recente do sistema operacional.

O desenvolvimento de aplicativos para Android é normalmente feito na linguagem Java, apesar de já ser possível a utilização de outras linguagens. Apesar do uso dessa linguagem, o Android não executa as aplicações em um ambiente Java comum. O Android possui a sua própria API ou biblioteca padrão e apesar de muitas das funcionalidades da biblioteca padrão Java estarem implementadas nela, alguns pontos como a biblioteca de interface gráfica são diferentes. Além disso, não é usada a JVM padrão para executar o bytecode, e sim uma máquina virtual chamada Dalvik que é uma implementação otimizada para dispositivos móveis.

Por questões de segurança, cada aplicativo Android roda em sua própria caixa de areia (sandbox) e em uma máquina virtual à parte, ficando totalmente isolado de outros aplicativos, embora a API forneça meios para que aplicativos comuniquem entre si e integrem funcionalidades. Os aplicativos também não têm acesso, por padrão, a todos os recursos do sistema. Para poder ler ou escrever

arquivos, ou utilizar comunicação Bluetooth, por exemplo, o aplicativo deve pedir acesso aos recursos. Todas as permissões devem ser concedidas pelo usuário no momento da instalação.

Outra diferença que é relevante para o desenvolvedor é que os aplicativos Android não têm um único ponto de entrada como comumente ocorre na programação na maioria das linguagens. Em outras palavras, os aplicativos não são um programa com uma função "main()" que será executada. Em vez disso, eles são construídos baseados em quatro componentes básicos - Atividade, Serviço, Provedor de conteúdo e Receptor de broadcast, sendo que cada um deles possui um ciclo de vida distinto. O mais importante deles é a Atividade, que representa uma tela com uma interface de usuário. Um Serviço é uma tarefa que é realiada em segundo plano e não possui uma interface visual. Provedores de conteúdo são usados para o compartilhamento de dados entre aplicações e Receptores de broadcast permitem realizar ações quando certos eventos ocorrem ou quando são recebidas mensagens de outras aplicações.

2.7 Arduino

O Arduino é uma plataforma livre de prototipação e desenvolvimento de software embarcado baseada em microcontroladores, normalmente de 8 bits (Arduino, 2014). Os kits Arduino atingiram uma alta popularidade, principalmente para fins educacionais entre hobbistas, pelo seu baixo custo e facilidade de uso (Arduino Projects, 2014). O projeto Arduino disponibiliza uma IDE extremamente simples que permite que se programe em uma linguagem semelhante a C, porém sem lidar com detalhes de nível de hardware típicos da programação de microcontroladores. Por ser um projeto livre, as placas Arduino são produzidas não só pelos fabricantes oficiais, havendo placas compatíveis praticamente idênticas produzidas em diversos países, inclusive no Brasil (RoboCore, 2014).

Existem alguns modelos diferentes de placa Arduino, sendo que a placa mais comumente encontrada no mercado e usada é o Arduino Uno R3 (vide Figura 7), é baseada no processador Atmel ATmega328. Ela possui 14 pinos de E/S digital, sendo que seis deles também podem ser usados como PWM, 6 entradas analógicas, um oscilador de cristal de 16MHz, um conector USB e uma entrada para alimentação.

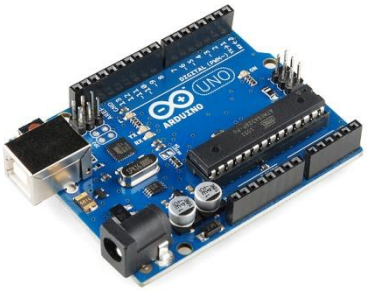

Figura 7 - Placa Arduino Uno (Arduino, 2014)

3. Materiais e Métodos

Nesse capítulo, serão descritos os materiais usados na montagem do protótipo, assim como os métodos usados na modelagem e análise das características do sistema.

Os materiais usados na montagem do sistema são listados a seguir, e detalhados logo em seguida:

Uma Placa Arduino Uno R3
Um módulo transmissor/receptor Bluetooth
Um Circuito Integrado de ponte H L239D
Um motor de corrente contínua RF-300FA-12350
Uma bateria de 9V para alimentação
Um *protoboard* e fios para conexão
Um dispositivo Android

3.1 Arduino Uno R3

Foi escolhida uma placa Arduino Uno R3 por ela permitir o uso de um ambiente de programação descomplicado (Arduino, 2014), possuir baixo custo e ser facilmente encontrada em lojas especializadas.

3.2 Módulo transmissor/receptor bluetooth

Utilizou-se um módulo receptor/transmissor **Bluetooth** que implementa o perfil SPP para atuar como *Device B* (Bluetooth SIG, 2014), ou seja, é capaz de aguardar que um outro dispositivo, chamado *Device A*, o descubra, inicie o pareamento e depois inicie a conexão. Uma vez conectado, ele converte os dados recebidos em Bluetooth para comunicação serial semelhante ao padrão RS232, porém com níveis lógicos diferentes, podendo ter seus pinos Rx/Tx ligados diretamente aos pinos correspondentes de um microcontrolador que possua um UART. O módulo escolhido não é facilmente encontrado por se tratar de um modelo um pouco antigo, porém há muitos produtos semelhantes em lojas e sites especializados, geralmente identificados como Módulos Bluetooth Serial.

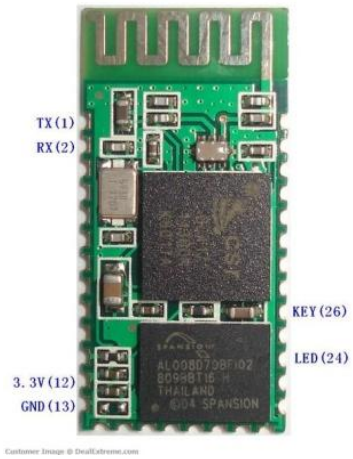

Figura 8 - Módulo Bluetooth semelhante ao utilizado, com *pinout* dos pinos mais importantes. (DX)

3.3 Circuito de Ponte H L293D

O L293D é um circuito integrado que possui quatro circuitos de meia-ponte independentes, que podem ser usados como duas pontes H (Texas Instruments, 2004). Uma ponte H é um circuito projetado de maneira a poder acionar uma carga de quatro maneiras diferentes, sendo possível, para um motor por exemplo, acioná-lo com polaridade direta ou reversa, ou ainda desligá-lo. O esquema de uma ponte H ideal é mostrado na Figura 9. A parte do circuito composta pelas duas chaves do lado esquerdo é chamada meia-ponte (o mesmo vale para o lado esquerdo). As duas chaves de uma meia-ponte nunca são acionadas ao mesmo tempo, pois isso causaria um curto-circuito. Pontes H são comumente usadas para acionar motores ou outras cargas com característica indutiva, e por isso quase sempre são incluídos os chamados diodos de flyback, necessários para evitar picos de tensão reversa que podem queimar componentes do circuito. Esses picos de tensão ocorrem quando há chaveamento, devido ao fato de que a tensão entre os terminais de um indutor são proporcionais à taxa de variação da corrente que o atravessa. Um chaveamento sem diodos de flyback pode fazer com que a corrente caia bruscamente para zero, gerando tensões altíssimas. Os diodos são incluídos para que o mesmo valor de corrente continue atravessando a carga indutiva logo após o chaveamento, impedindo que esse fenômeno ocorra. Um circuito típico é mostrado na Figura 10.

O L293D é projetado para operar com baixas correntes (< 600ma, 1200 de pico). A queda de tensão típica da saída quando usada como ponte H, isto é, a queda de tensão somada das duas meias pontes, é de 2,6V. Uma vantagem interessante desse CI é que ele possui diodos de flyback integrados.

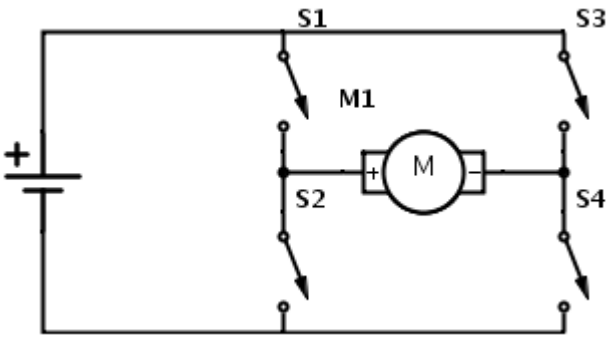

Figura 9 - Uma ponte H ideal com chaves

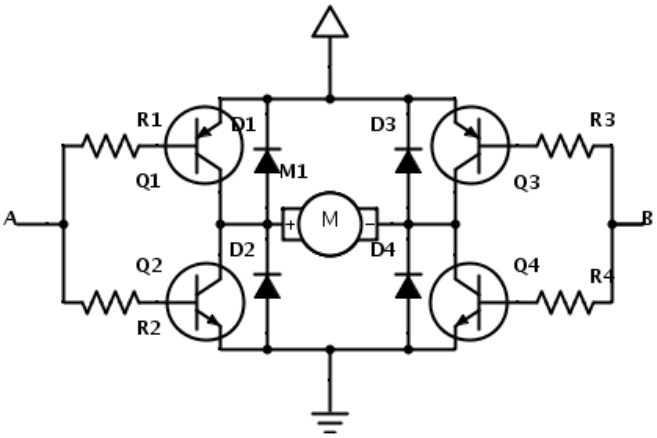

Figura 10 - Uma implementação simples de ponte H usando transistores bipolares e diodos de flyback (Texas Instruments, 2004)

3.4 Motor RF-300FA-12350

Esse motor foi escolhido para constituir a base da roda de reação por ter tamanho e peso reduzidos, corrente de acionamento relativamente baixa e um torque relativamente alto - fator mais importante no nosso caso, pois terá impacto direto no tempo de resposta do sistema. A sua aplicação principal é em leitores de CD e DVD e por isso pode ser adquirido facilmente em lojas especializadas.

3.5 Bateria 9V

Utilizamos uma bateria de 9V comum, não recarregável. No entanto que essa não é uma solução ideal - o motor exige dessas baterias uma corrente próxima do limite que elas podem fornecer, e isso compromete muito o tempo de vida delas. Uma opção melhor seria usar baterias especiais capazes de fornecer correntes altas (até 1A).

3.6 Dispositivo Android

Para embarcar o nosso software de sensoriamento e controle, escolhemos um *smartphone* Android do modelo Samsung GT-S5360 - Galaxy Y (Figura 11). Ele possui o sistema operacional

Android 2.3 e apresenta, além de comunicação Bluetooth, um acelerômetro um magnetômetro integrados. Esse modelo específico foi escolhido apenas pela disponibilidade, tamanho reduzido e baixo custo - na data da escrita desse trabalho, menos de R$300. No entanto, poderia ter sido usado qualquer outro dispositivo Android com os recursos mínimos citados acima.

Figura 11 - Samsung GT-S5360 - Galaxy Y (Samsung)

3.7 Montagem do sistema e circuitos

Usando todos os componentes já descritos e fios apropriados, foi feita a montagem do sistema em um *protoboard*. O esquemático mostrando os *pinouts* dos circuitos e as conexões feitas é mostrado na Figura 12.

O motor, acoplado a uma roda, foi colado na parte inferior do protoboard, e foram usados fios para suspender o experimento montado, de maneira que ele possa ser rotacionado livremente em torno do eixo Z, estando sujeito a uma quantidade mínima de torques e forças externas. Dessa maneira a roda pode ser usada para troca de momento angular com o restante do sistema e seria possível, por meio de um controle preciso da sua velocidade angular, controlar a orientação do sistema. O conjunto das partes mecânica e elétrica do dispositivo pode ser visualizado na Figura 13.

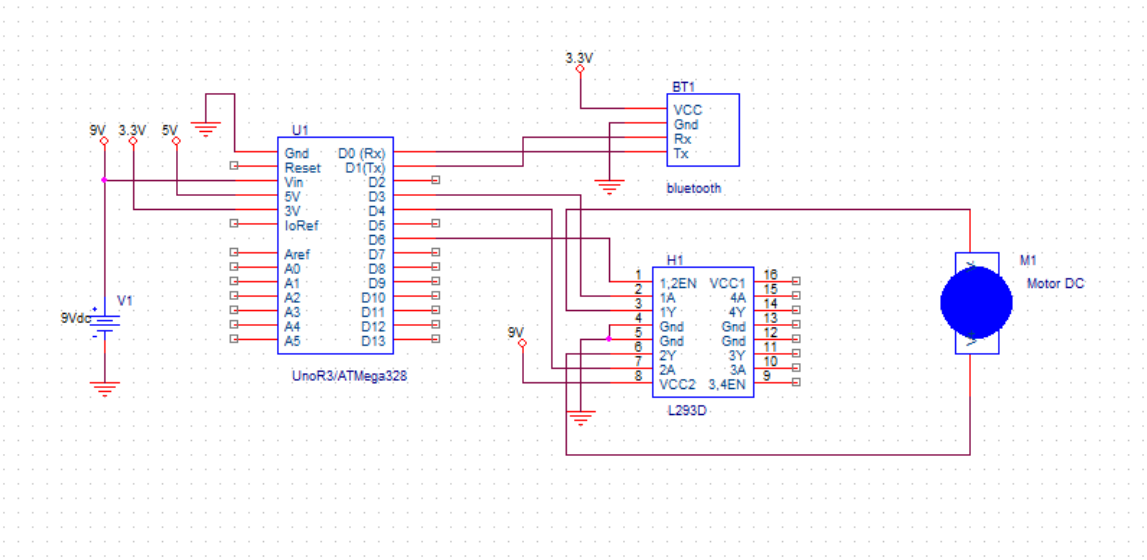

Figura 12 - Esquemático com as conexões entre o microcontrolador, motor e demais circuitos do sistema de rodas de reação

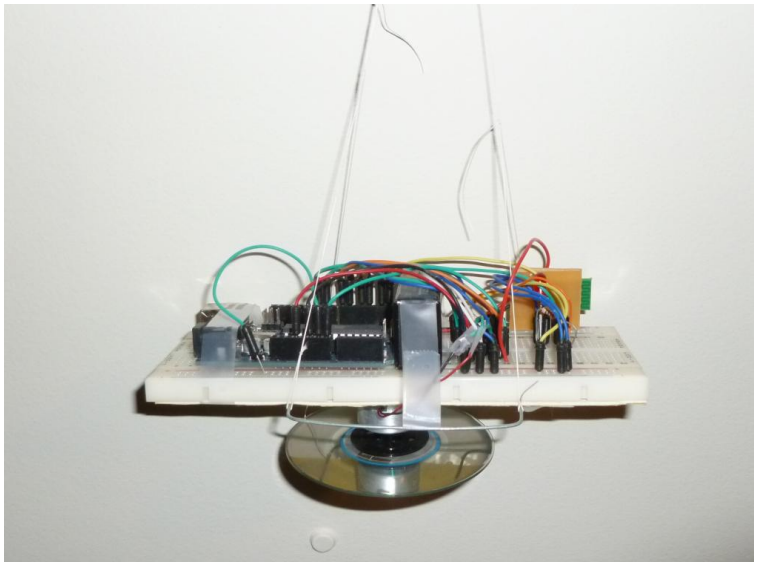

Figura 13 - Fotografia do sistema de roda de reação montado, com todos os componentes exceto o dispositivo Android. Podem ser observados os fios que o suspendem, permitindo um movimento rotacional no eixo z (vertical) praticamente livre da atuação de qualquer força externa. Sob a placa é possível ver o motor e a roda, responsáveis pela troca de momento angular com o restante do sistema. Se o sistema estiver realizando um movimento de rotação indesejado no sentido horário, por exemplo, é possível acionar o motor para que ele gire nesse mesmo sentido, transferindo à roda todo o momento angular até que o restante do sistema fique imóvel.

4. Resultados

Esse capítulo apresenta os resultados obtidos ao longo do projeto. Os itens 4.1 e 4.2 detalham o software resultante do desenvolvimento do trabalho. Os itens 4.3 e 4.4 descrevem um estudo feito sobre as características do sistema desenvolvido, em termos de ruído e modelagem, respectivamente.

4.1 Software desenvolvido - Arduino

Foi escrito um código relativamente simples que foi embarcado na placa Arduino. Ela basicamente realiza uma única tarefa, que é aguardar a chegada de dados pela porta serial. Quando eles são recebidos, eles são interpretados como a direção de acionamento do circuito de ponte H e ao ciclo de trabalho. Então os pinos de I/O correspondentes ligados ponte H recebem os valores correspondentes à direção especificada, e o pino que controla o PWM tem o seu ciclo de trabalho alterado. O microcontrolador então volta novamente ao estado em que aguarda por novos dados de controle.

O protocolo de comunicação estabelecido é bastante simples, com a finalidade de facilitar a implementação e evitar falhas e problemas. Ele consiste em sequências de três bytes. Desses, o primeiro indica a direção de aplicação da tensão; o segundo representa o valor do ciclo de trabalho do PWM, numa escala de 0 a 255, sendo que 255 corresponde a um ciclo de 100%. Dessa maneira, quanto mais alto for esse valor, maior será o módulo da tensão média aplicada no motor. O terceiro byte deve possuir um valor fixo estabelecido e é redundante. Ele foi acrescentado apenas para evitar que pacotes recebidos indevidamente - por erros de comunicação ou por erros no código do aplicativo Android - fossem interpretados como comandos válidos e dessa maneira evitar danos ao protótipo por acionamento indevido do motor.

O código fonte do programa desenvolvido para executar sobre a plataforma Arduino pode ser visto integralmente no anexo A.

4.2 Software desenvolvido - Android

Foi desenvolvido um aplicativo Android capaz de realizar uma série de tarefas relativas ao sensoriamento, controle do sistema e registro de dados.

O aferimento da orientação foi feito usando o sensor de rotação, que é uma abstração de software disponibilizada pela API Android e consiste na combinação de vários sensores de orientação disponíveis, sendo identificado pela constante Sensor.TYPE_ROTATION_VECTOR. Registrando esse sensor, o aplicativo passa a receber eventos referentes a mudanças na orientação, que é calculada a partir dos sensores disponíveis que normalmente são apenas o acelerômetro e magnetômetro nos modelos de baixo custo, sendo que em alguns modelos também pode haver um giroscópio.

Os valores de orientação que são recebidos nesses eventos usam a representação na forma de quaterniões, que não são muito práticos para o nosso problema em particular, e são convertidos em ângulos de *roll*, *pitch* e *yaw* usando funções específicas da API específicas para isso.

As funcionalidades implementadas foram:

- Conectar-se ao módulo SPP/Bluetooth, permitindo a busca de dispositivos e realizando a operação de pareamento, se necessário.
- Calcular e exibir os ângulos de *roll*, *pitch* e *yaw* em campos de texto correspondentes.
- Enviar comandos de controle manualmente, através de uma caixa de texto onde pode ser inserido um valor numérico. Ao pressionar o botão correspondente, esse valor numérico é enviado por Bluetooth e refletirá na velocidade do motor.
- Registrar dados de orientação calculados a partir da informação dos sensores (sequência de valores de *yaw* ao longo do tempo, para análise de ruído por exemplo). Esses dados são gravados em um arquivo dentro do cartão SD (armazenamento externo) do dispositivo, em formato que pode ser facilmente lido por um script em MATLAB, por exemplo.
- Controle automático do ângulo de *yaw*, feito através de parâmetros PID ajustáveis. Isso se deu por meio da implementação de um controlador PID digital em software no dispositivo Android. O valor do ângulo de *yaw*, calculador a partir dos dados dos sensores, é integrado e diferenciado. Os três sinais - o original, a integral e a derivada - são multiplicados pelos coeficientes correspondentes - k_p, k_i e k_d, respectivamente - e somados, formando um sinal de controle. Esse sinal de controle é enviado por Bluetooth para a placa Arduino e consequentemente refletirá na tensão aplicada no motor. A interface gráfica do aplicativo permite que o usuário ajuste esses três coeficientes para obter o comportamento desejado do controlador.

A parte principal do código fonte do aplicativo Android é apresentada no Apêndice B.

A tela correspondente à atividade principal do aplicativo é mostrada na Figura 14.

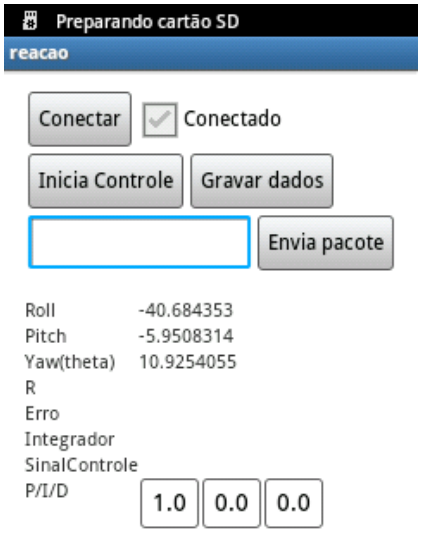

Figura 14 - Interface visual do aplicativo Android usado para controlar o sistema. O usuário pode usar o botão no topo da tela para iniciar a conexão Bluetooth com o módulo. Depois disso, é possível enviar comandos manualmente usando um campo numérico editável, que aceita valores de -255 a 255, representando a faixa de valores de tensão média que podem ser aplicados ao motor usando PWM. Um botão permite que se inicie o modo de gravação de dados, no qual os valores da variável *yaw* são registrados conforme atualizações são recebidas do sensor, e posteriormente gravados em um arquivo. Os ângulos de *roll*, *pitch* e *yaw* também são calculados e exibidos. Por último, o usuário pode realizar o controle automático na variável ψ (*yaw*) por meio de um controlador tipo PID, podendo inclusive escolher os parâmetros.

4.3 Identificação e Análise de Ruído no sensor

Com a finalidade de descobrir as características do ruído presente nos sensores de orientação, foi feita uma análise usando uma funcionalidade de registro de dados do aplicativo Android desenvolvido (que será detalhado mais adiante). Conhecer as propriedades do ruído permite conhecer e entender o seu impacto no desempenho do sistema e pode permitir, em alguns casos, a sua filtragem e consequentemente a obtenção de uma melhora no desempenho.

Como apenas o eixo z é tratado no nosso problema, foi feita uma análise do ruído presente apenas na variável ψ (*yaw*). O experimento foi realizado da seguinte maneira: um programa foi escrito para registrar os valores recebidos do sensor, e o dispositivo Android foi deixado em repouso enquanto os dados foram adquiridos. Modelamos o ruído como sendo uma variável independente de ϕ, isto é, se considerarmos que ϕ_s é o valor real de ϕ, ϕ_r é o valor de ϕ medido pelo sensor e $w(t)$ é o ruído, temos que

$$\psi_s(t) = \psi_r(t) + w(t) \tag{10}$$

Com o dispositivo em repouso, o valor de ϕ_r é constante. Logo, toda a variação temporal observada em ϕ_s se deve ao ruído. Além disso, para um período suficientemente longo de observação, a média temporal de ϕ_s se aproxima de ϕ_r se $w(t)$ for uma variável aleatória de média zero.

Levando isso em conta, foi feito um experimento em que o dispositivo foi deixado em repouso e o valor de ψ foi registrado ao longo do tempo. Foi feita uma análise na frequência do ruído usando uma Transformada Discreta de Fourier (TDF) (Oppenheim e Schafer, 1974) sobre uma curva de interpolação do tipo *spline* da série de valores de ψ. O cálculo dessa curva de interpolação foi necessário porque a série original não tinha uma taxa de amostragem constante, um requisito para que o resultado da TDF seja válido. Foi calculada uma aproximação do valor RMS do ruído a partir da sequência de valores de ψ, do seu valor médio ϕ_{med} e do número total de amostras N conforme a fórmula a seguir:

$$w_{rms} = \sqrt{\sum_{i=0}^{N} \frac{(\psi_i - \psi_{med})^2}{\mathrm{N}}} \tag{11}$$

O mesmo experimento foi repetido usando um outro dispositivo Android, o GT-i9190 (Galaxy S4 mini) por este possuir, além dos sensores que o primeiro dispositivo testado possui, um sensor giroscópico.

Com os dados do experimento feito para analisar as características de ruído dos sensores do dispositivo Android que usamos no protótipo (GT-S5360), o valor RMS estimado do ruído foi

$$w_{rms} = 1{,}36°$$

No caso do experimento usando o dispositivo Android com acelerômetro (GT-i9190), o valor foi um pouco menor: obteve-se 0,77º usando a mesma fórmula.

A curva de $\psi_r(t)$ ao longo do tempo obtida usando o primeiro dispositivo e que mostra as oscilações causadas pelo ruído pode ser vista na Figura 15. A Figura 16 mostra a TDF dessa curva, onde é possível ver onde se concentram as suas componentes espectrais.

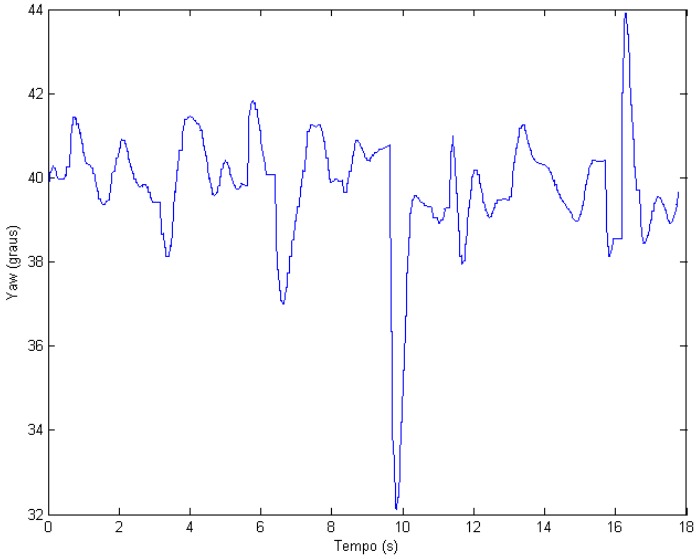

Figura 15 - Gráfico dos dados do experimento usado para analisar as características do ruído dos sensores, em que o dispositivo foi mantido em repouso. Na ausência de ruído, o ângulo ψ deveria se manter constante, mas a variação observada chegou a cerca de 12º. O valor RMS do ruído é de aproximadamente 1,36º.

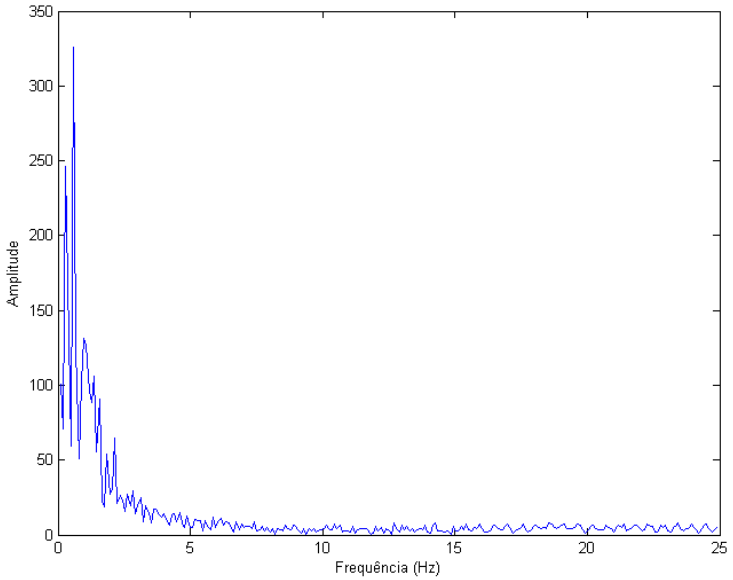

Figura 16 - Gráfico do módulo da Transformada Discreta de Fourier do experimento, desenhado para analisar as componentes espectrais do ruído. É fácil observar que o ruído se concentra nas frequências mais baixas, inferiores a 2Hz.

4.4 Modelo e levantamento de parâmetros do motor DC e roda

Baseado no modelo simplificado para motores de corrente contínua apresentado anteriormente, e na folha de dados do motor usado e medições, foi realizado um levantamento de parâmetros, que será detalhado nesta seção.

Alguns dos dados relevantes presentes na folha de dados foram organizados na Tabela 1.

	Torque (mNm)	Corrente (mA)	Velocidade (rpm)
Stall (3V)	2.51	0.39	-
Sem carga (3V)	-	0.22	3500

Tabela 1 -Alguns dados relevantes extraídos da folha de dados do motor RF-300FA-12350

Além disso, a resistência do enrolamento foi medida com multímetro e obtivemos o valor de 10,2 Ohms.

O torque é relacionado à corrente no enrolamento pela expressão

$$\tau = K_t i - K_b \omega$$

Observando novamente os dados da Tabela 1, podemos substituir valores nessa equação. Para o caso de *stall* do motor (isto é, quando o rotor está parado) o termo relacionado à velocidade angular se anula, e como I e τ são determinados, substituindo-os na equação chegamos a

$$K_t = 6{,}4 \cdot 10^{-3} \frac{Nm}{A}$$

Para o caso do funcionamento sem carga, sabe-se que o termo de torque é zero, pois o momento angular é constante e não há forças externar atuando sobre o rotor. Temos então que

$$K_t i = K_b \omega$$

e então é fácil reorganizar a equação para a forma

$$K_b = \frac{K_t i}{\omega}.$$

Susbtituindo os valores apropriados para I, ω e K_t, chegamos ao valor

$$K_b = 3{,}86 \cdot 10^{-7} Nm \cdot s$$

Podemos também estimar o valor de K_e. Para o caso do motor operando sem carga, e lembrando da equação $V = Ri + K_e \omega$, podemos escrevê-la da forma

$$K_e = \frac{V - Ri}{\omega}$$

Como todos os outros parâmetros dessa já são conhecidos, chegamos a

$$K_e = 7{,}3 \cdot 10^{-3}\, Vs\,.$$

Outro parâmetro importante que precisou ser estimado foi o momento de inércia da roda. Como utilizamos um disco com 8cm de diâmetro e aproximadamente 8g de massa acoplado ao rotor, e sabendo que a fórmula do momento de inércia de um disco em torno do eixo perpendicular ao seu plano é

$$I = \frac{1}{2}MR^2$$

sendo M a sua massa e R o seu raio, chegamos ao valor

$$I_r = 6{,}4 \cdot 10^{-6} kg \cdot m^2.$$

Para estimar o valor do momento de inércia do restante do sistema, fizemos uma aproximação grosseira e o consideramos como uma barra delgada de comprimento L. Sabe-se que para um objeto com essa geometria, o momento de inércia é dado por

$$I = \frac{1}{12}ML^2$$

e usando esta expressão, substituindo M pelo valor da massa, chegamos ao valor

$$I_s = 4{,}05 \cdot 10^{-4} kg \cdot m^2.$$

Os parâmetros estimados foram todos organizados na tabela 2.

Parâmetro	Símbolo	Valor
Resistência do enrolamento	R	10,2Ω
Constante de torque	K_t	$6{,}4 \cdot 10^{-3}\ Nm/A$
Constante de amortecimento	K_b	$3{,}86 \cdot 10^{-7} Nm \cdot s$
Constante Eletromotriz	K_e	$7{,}3 \cdot 10^{-3}\ Vs$
Momento de inércia da roda	I_w	$6{,}4 \cdot 10^{-6} kg \cdot m^2$
Momento de inércia da placa	I_s	$4{,}05 \cdot 10^{-4} kg \cdot m^2$

Tabela 2 - Valores calculados e estimados para os parâmetros do motor DC e da roda de reação.

Em seguida, foi feito um experimento com o auxílio de um tacômetro e usando PWM para aplicar uma tensão média variável no motor. O objetivo era medir a velocidade angular atingida pelo motor para diferentes valores de tensão e comparar os valores obtidos com o modelo desenvolvido. A tensão aplicada foi de 2,4V, correspondente a 5V menos a queda de tensão na ponte H; o ciclo de

trabalho foi variado de 0 até 100%. O experimento foi feito aplicando a tensão nos dois sentidos possíveis. Os dados obtidos foram organizados no gráfico da Figura 17.

Pelo gráfico da Figura 17, é fácil observar que o modelo desenvolvido não modela perfeitamente o sistema. Isso ocorre porque o mesmo foi considerado linear, quando na verdade não é: para tensões médias menores que 0,4V o motor sequer se move devido ao atrito. O atrito, na verdade, tem uma característica fortemente não linear. No entanto, a modelagem de um sistema não linear exigiria técnicas de controle muito mais complexas e por isso descartamos essa possibilidade, da mesma maneira que (Sidi, 1995) faz em sua análise do CRR. Observamos que para tensões acima de 2V, o nosso modelo começa a se aproximar mais da curva real e portanto continuamos usando esse modelo pelo resto do projeto.

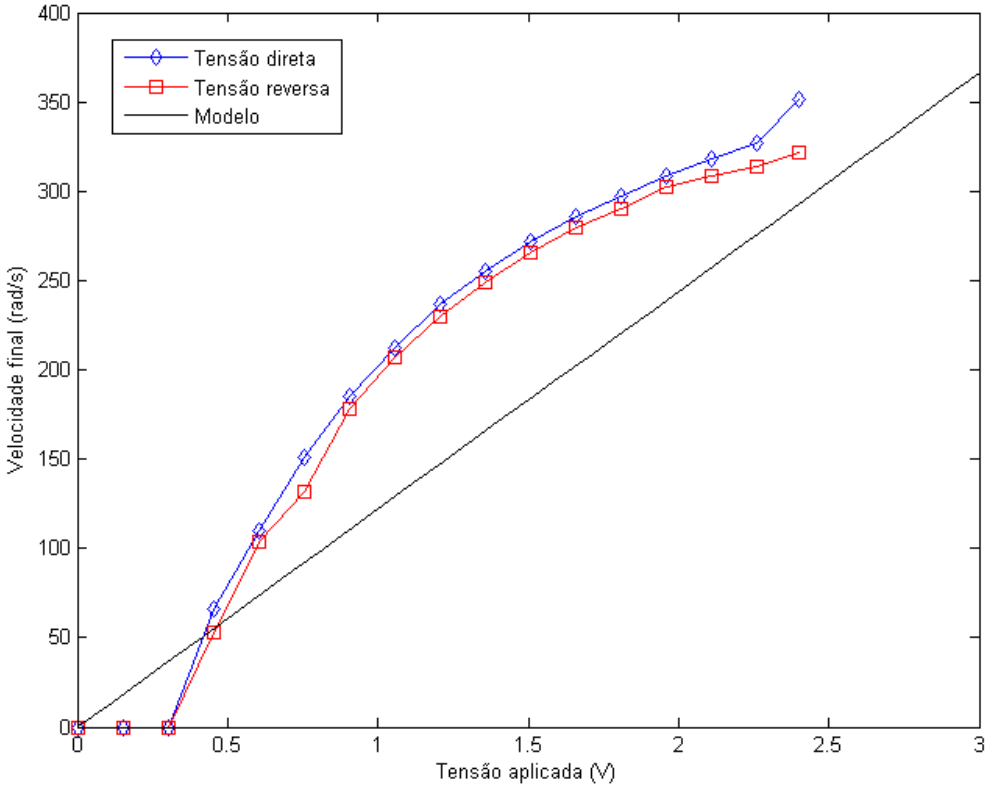

Figura 17 - Gráfico do experimento medindo a velocidade final atingida com diferentes valores de tensão média (em módulo, nos dois sentidos de rotação) traçado contra a reta obtida pelo modelo e parâmetros levantados.

Para finalizar a verificação da validade do modelo, calculamos a função de transferência em malha aberta do sistema, de acordo com o diagrama de blocos da Figura 3. Traçamos um gráfico da resposta do sistema (em malha aberta) a uma entrada na forma de degrau. O resultado é apresentado na Figura 18. O gráfico mostra que essa função de transferência teórica se aproxima do comportamento do sistema real: a partir do momento de aplicação da tensão, a velocidade sobe

lentamente até o seu valor máximo. Foi verificado também que os tempos de subida e acomodação são condizentes com a realidade (apesar de não ter sido levantada uma curva de resposta ao degrau do sistema real).

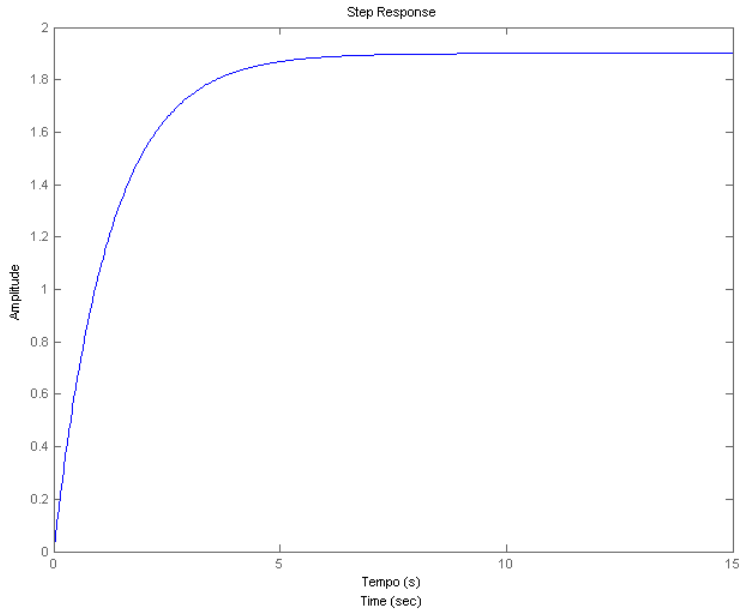

Figura 18 -Gráfico da resposta ao degrau do sistema modelado, em malha aberta. O eixo vertical corresponde à velocidade angular em rad/s e o eixo horizontal é correspondente ao tempo, em segundos.

5. Discussão e Conclusões

O protótipo montado teve sucesso em testes realizados, sendo que tanto o hardware quando o software se mostraram funcionando corretamente. O sistema foi capaz de ativar o motor usando a interface do aplicativo Android e através da conexão Bluetooth. Ele será disponibilizado para a Escola de Engenharia de São Carlos (EESC) para uso em trabalhos e estudos futuros. As demais funcionalidades incluídas no aplicativo Android também foram testadas com sucesso. O modo de controle PID funcionou corretamente e foi testado com vários parâmetros diferentes, calculados usando ferramentas do software MATLAB. No entanto, foi verificado que devido às características do sistema que não foram consideradas no modelo - o ruído e a não-linearidade - pode ser difícil determinar parâmetros PID para esse sistema de maneira que ele se torne estável e tenha um tempo de resposta rápido.

Analisando os dados obtidos no experimento usado para analisar o ruído nos sensores, chegamos a duas conclusões. A primeira é que os níveis de ruído são razoavelmente altos, e que por ter componentes predominantemente de baixa frequência, ele não pode ser filtrado de maneira a diminuí-lo significantemente sem que o desempenho do sistema seja comprometido. Isso significa que o controle do sistema estará sujeito a essas oscilações e o comportamento do sistema em malha fechada pode ser comprometido, dependendo da técnica de controle empregada (assumindo-se que possam ser usadas técnicas mais sofisticadas do que o controlador PID desenvolvido). A segunda é que a disponibilidade de um giroscópio não reduziu de maneira drástica o ruído na variável analisada, como esperávamos a princípio.

5.1 Continuação do projeto

Durante a execução e após a realização de diversos testes com o sistema, foram identificados vários pontos que poderiam ser melhorados no mesmo. Muitas dessas melhorias, no entanto, não foram executadas devido a restrições de tempo, materiais ou por serem demasiadamente complicadas e fugirem ao escopo do projeto. Essas melhorias serão detalhadas a seguir, para permitir aperfeiçoamentos nesse sistema ou em qualquer outro projeto que venha a usa-lo como base no futuro.

A bateria usada - do tipo alcalina comum, não recarregável - apresentou uma duração muito baixa, suficiente apenas para menos de 30 minutos de operação. Isso ocorre porque esse tipo de bateria tem seu período de vida diminuído drasticamente quando se exige uma alta corrente dela. Uma solução possível seria usar baterias de níquel-hidreto metálico (NiMh). Essas baterias, além de serem recarregáveis e terem uma alta capacidade de carga, são ideais para aplicações com alto dreno de corrente (Energizer, 2010). Células de NiMh possuem uma tensão nominal de 1,25V, e portanto seria necessária a utilização de pelo menos seis células em série para atingir a tensão mínima de alimentação da placa Arduino (7V).

O desempenho obtido através da técnica de controle que empregamos - um controlador PID e uma única malha fechada - não seria aceitável para muitas aplicações. Uma alternativa seria usar uma malha de controle mais sofisticada, usando realimentação não só de posição, mas também da corrente que atravessa o motor DC. A ideia por trás disso é que, como o torque do motor é proporcional à corrente que o atravessa, seria possível fazer uma malha de controle fechada para controlar o torque no motor, e uma segunda malha para controlar a posição do sistema. Dessa maneira seria possível controlar a posição angular com resultados melhores (Sidi, 1995).

Essa implementação foi cogitada, porém não foi executada pois adicionar a realimentação da corrente exigiria mudanças no hardware do sistema. Analisando as possibilidades de implementação, concluímos que uma das mais fáceis seria substituir o CI L293D por outro da família L298 (ST Microelectronics, 2000), de circuitos integrados de ponte H com saídas para medição de corrente. Essas saídas devem ser ligadas por meio de um resistor de resistência baixa (1 Ohm ou menos para a maioria das aplicações) ao terra. Dessa maneira, e o valor da tensão nessa saída será proporcional à corrente que atravessa a carga da ponte H de acordo com a lei de Ohm. O pino dessa saída pode então ser ligado a uma das entradas analógicas do microcontrolador para possibilitar a leitura do seu valor.

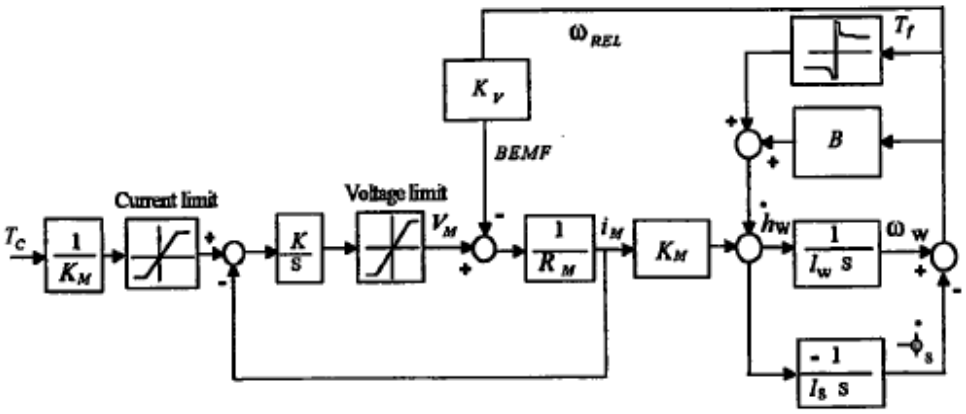

Figura 19 - Malha de controle com realimentação de corrente, conforme apresentada em (Sidi, 1995).

Bibliografia

Arduino. (2014). *Arduino HomePage*. Fonte: http://www.arduino.cc/

Arduino. (2014). *Language Reference*. Fonte: Arduino Website: http://arduino.cc/en/Reference/HomePage

Arduino Projects. (2014). Fonte: Instructables: http://www.instructables.com/id/Arduino-Projects/

Bluetooth SIG. (2012). *RFCOMM with TS 07.10*. Fonte: Bluetooth Developer Portal: https://developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx

Bluetooth SIG. (2014). *Serial Port Profile*. Fonte: Bluetooth Developer Portal: https://developer.bluetooth.org/TechnologyOverview/Pages/SPP.aspx

Chobotov, V. A. (1991). *Spacecraft Attitude Dynamics and Control*. Krieger Publication Company.

Craig, J. J. (2005). *Introduction to Robotics - Mechanics and Control* (3ª ed.). Pearson Prentice Hall.

DX. (s.d.). *Loja Online DealExtreme*. Fonte: www.dx.com

Energizer. (2010). *Nickel Metal Hydride (NiMh) Handbook and Application Manual*. Fonte: http://data.energizer.com/PDFs/nickelmetalhydride_appman.pdf

Gajamohan, M., Merz, M., Thommen, I., & D'Andrea, R. (2012). The Cubli: A Cube that can Jump Up and Balance. *International Conference on Intelligent Robots and Systems*.

Gajamohan, M., Muehlebach, M., Widmer, T., & D'Andrea, R. (2013). The Cubli: A Reaction Wheel Based 3D Inverted Pendulum. *European Control Conference (ECC)*.

Halliday, D., Resnick, R., & Walker, J. (2009). *Fundamentos de Física 8a. Edição*. Rio de Janeiro: LTC.

Hamilton, W. R. (1844). *ON QUATERNIONS, OR ON A NEW SYSTEM OF IMAGINARIES IN ALGEBRA*. Fonte: http://www.emis.ams.org/classics/Hamilton/OnQuat.pdf

Heidt, H. (2000). CubeSat: A new Generation of Picosatellite for Education and Industry Low-Cost Space. *14TH Annual/USU Conference on Small Satellites*.

Huang, A. S., & Rudolph, L. (2007). *Bluetooth Essentials for Programmers*. Cambridge University Press.

ISIS. (2006). *CubeSatShop*. Acesso em 2014, disponível em CubeSatShop: http://www.cubesatshop.com/

Microsoft. (2008). *Overview of Bluetooth Pairing.* Fonte: Microsoft Developer Network: http://msdn.microsoft.com/pt-br/library/cc510479.aspx

Muehlebach, M., Mohanarajah, G., & D'Andrea, R. (2013). Nonlinear Analysis and Control of a Reaction Wheel-based 3D Inverted Pendulum. *Conference on Decision and Control, CDC.*

Ogata, K. (1970). *Engenharia de Controle Moderno* (2ª ed.). Rio de Janeiro: Editora Prentice Hall do Brasil.

Oppenheim, A. P., & Schafer, R. W. (1974). *Digital Signal Processing.* Prentice Hall.

Puig-Suari, J., Turner, C., & Twiggs, R. J. (2001). CubeSat: The Development and Launch Support Infrastructure for Eighteen Different Satellite. *15TH Annual/USU Conference on Small Satellites.*

RoboCore. (2014). *Loja RoboCore.* Acesso em 17 de 05 de 2014, disponível em https://www.robocore.net/

Samsung. (s.d.). *Samsung Homepage*. Fonte: Samsung Website: www.samsung.com

Sidi, M. J. (1995). *Spacecraft Dynamics and Control - a Practical Engineering Approach.* New York: Cambridge Univesity Press.

Sridhar, T. (2008). *Wi-Fi, Bluetooth and WiMAX.* Fonte: www.cisco.com.

ST Microelectronics. (2000). L298 Dual Full-Bridge Driver Datasheet.

Texas Instruments. (2004). *L293, L293D Datasheet.* Fonte: Texas Instruments: http://www.ti.com/lit/ds/symlink/l293d.pdf

The Apache Software Foundation. (2004). *Apache License, Version 2.0.* Fonte: http://www.apache.org/licenses/LICENSE-2.0

The CubeSat Program. (2009). *CubeSat Design Specification Rev. 12.* Fonte: http://browncubesat.org/wp-content/uploads/2013/01/Cubesat-Reqs.pdf

Yaghmour, K. (2013). *Embedded Android.* O'Reilly.

Apêndice A - Código Arduino (arquivo completo)

```
//controle_rodas.ino

const int e12 = 9, e34 = 10, i1 = 3, i2 = 4, i3 = 6, i4 = 7;
    //definição das ligações dos pinos do microcontrolador com os
    //pinos do CI
    //e12 é um dos pinos de Enable (usados para PWM)
    //i1 e i2 são as duas entradas da ponte H
    //(usados para escolher a direção de aplicação da tensão)

    //Função de inicialização
    void setup() {
     //Inicialização dos pinos e da interface serial:
      Serial.begin(9600);
      while (!Serial) {
        ;
      }

      pinMode(e12, OUTPUT);
      pinMode(i1, OUTPUT);
      pinMode(i2, OUTPUT);

      digitalWrite(i1, LOW);
      digitalWrite(i2, HIGH);
      analogWrite(e12, LOW);

      //TCCR1B = (TCCR1B & 0b11111000) | 0x05;
    }

    //Função que é chamada repetidamente enquanto o microcontrolador
    //estiver ligado
    void loop() {
      //aguarda a chegada de um conjunto de 3 bytes
      if(Serial.available() >= 3){
        char b0 = Serial.read();
        byte b1 = Serial.read();
        char b2 = Serial.read();
        //confere se o terceiro byte está correto
        if(b2 == '\n'){
          Serial.write("recvd\n");
          Serial.write(b0);
          Serial.write(b1);
          Serial.write(b2);
          //Usa o primeiro byte para controlar a direção
          if(b0 == '1'){
            Serial.write(positive)
            digitalWrite(i1, LOW);
            digitalWrite(i2, HIGH);
          }
          else{
            digitalWrite(i1, HIGH);
            digitalWrite(i2, LOW);
          }
          //Usa o segundo byte (velocidade) para controlar o PWM
          analogWrite(e12, b1);
        }else{
        }
      }
    }
```

Apêndice B - Código do Aplicativo Android

Abaixo inclupido o código completo do principal arquivo de código fonte do aplicativo Android desenvolvido, chamado MainActivity.java

```
package com.example.reacao;

import android.os.*;
import android.os.MessageQueue.IdleHandler;
import android.app.Activity;
import android.view.Menu;

import java.awt.font.NumericShaper;
import java.io.*;
import java.nio.charset.Charset;
import java.util.Timer;
import java.util.UUID;

import android.app.Activity;
import android.bluetooth.*;
import android.content.*;
import android.gesture.GestureOverlayView;
import android.hardware.*;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnTouchListener;
import android.widget.*;

//Atividade principal da nossa aplicação
public class MainActivity extends Activity implements OnClickListener, SensorEventListener{

  //Thread que estabelece a conexão Bluetooth
  private class ConnectThread extends Thread {
      private BluetoothSocket mmSocket;

       private final BluetoothDevice mmDevice;
      private UUID uuid = //Esse é o UUID referente ao perfil SPP
            UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");

      public ConnectThread(BluetoothDevice device) {
       mmSocket = null;
          mmDevice = device;

          // Cria um socket para se conectar ao dispositivo bluetooth
          try {
              // O uuid especifica o serviço desejado (SPP)
              mmSocket = device.createRfcommSocketToServiceRecord(uuid);
              Log.e("PEDRO", "RFCOMM socket created");
          } catch (IOException e) {
            Log.e("PEDRO", "Error creating RFCOMM socket");
          }

      }

      public void run() {
       try{
               // Cancela a descoberta porque ela torna a conexão lenta
               bta.cancelDiscovery();

               try {
                   // Tenta conectar o socket. Bloqueia até que ocorra sucesso ou falha.
                   mmSocket.connect();
                   Log.e("PEDRO", "RFCOMM socket connected");
               } catch (IOException connectException) {
                   // Falha na conexão
                   try {
                       mmSocket.close();
                   } catch (IOException closeException) { }
```

```
                return;
            }

            // Gerencia a conexão em uma thread separada
            manageConnectedSocket(mmSocket);
     }catch(Exception e){

     }
    }

    /* Cancela uma conexão e fecha o socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}

//Classe interna usada para gerenciar a conexão bluetooth
private class Conexao{
    private final BluetoothSocket mmSocket;
    private final OutputStream mmOutStream;
    private final InputStream mmInputStream;

    public Conexao(BluetoothSocket socket) {
        mmSocket = socket;
        OutputStream tmpOut = null;
        InputStream tmpIn = null;

        try {
            tmpOut = socket.getOutputStream();
            tmpIn = socket.getInputStream();
        } catch (IOException e) {
          Log.e("PEDRO", "Error getting io streams");
        }

        mmOutStream = tmpOut;
        mmInputStream = tmpIn;
    }

    /* Envia dados ao dispositivo */
    public void write(byte[] bytes) {
        try {
            mmOutStream.write(bytes);
            mmOutStream.flush();
        } catch (IOException e) { }
    }

     public void writeln(String s){
     write((s+"\n").getBytes());
    }

    public int read(byte[] buffer) throws IOException{
     return mmInputStream.read(buffer);
    }

    /* Desativa a conexão */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }

    public int available() throws IOException{
     return mmInputStream.available();
    }
}
//Inicio do codigo da classe MainActivity em si
//-------------------------------------------------------------------------------------------//

//Gerenciamento da conexao bluetooth
  private static BluetoothAdapter bta;
  private static Conexao conexao;
  private static boolean connected;//, accelOn;
```

```
  //Views da UI
  private Button btnConn, btnEnvia, btnControlador;
  private CheckBox checkBox;
  private static ConnectThread thread;
  private static EditText editVel;
  private Button btnGrava;

  //Codigos de requisicao para os Intents
  private static final int COD_REQ = 12;
private static final int REQUEST_CONNECT_DEVICE = 4;

//Gerenciamento do controlador
private boolean controllerOn = false;
private float ref_theta = 0;
private float theta = 0;
private static float MAX_PWM = 128; //Valor maximo de duty cycle PWM que pode ser enviado aos motores
(para //nao danifica-los)
private long ultimoT = 0;
private float ultimoErro = 0;
private static final int DELAY_ENVIO = 5; //Intervalo minimo entre envio de 2 comandos para os motores, em
ms

//Gerenciamento dos sensores
private boolean mInitialized, gravando = false;
private SensorManager mSensorManager;
private Sensor mRot;
private static int TAXA = SensorManager.SENSOR_DELAY_GAME;

//Gerenciamento dos testes/registro de dados
private long t0 = 0;
private float[] dados;
private int[] tempo;
private int iDados;

private float p = 1.0f, i =0.0f, d=0.0f;//parametros de ganho do controlador PI
private float integrador = 0.0f;

  private static BroadcastReceiver receiver;

  @Override
  public void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      try{
        setContentView(R.layout.activity_main);

        connected = false;

        bta = BluetoothAdapter.getDefaultAdapter();

        if(bta != null && !bta.isEnabled()){
          Intent in = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
          startActivityForResult(in, COD_REQ);
        }

        btnConn = (Button)this.findViewById(R.id.btnConn);
        btnEnvia = (Button)this.findViewById(R.id.btnEnvia);
        btnControlador = (Button)this.findViewById(R.id.btnControlador);
        checkBox = (CheckBox)(this.findViewById(R.id.checkBox));
        editVel = (EditText)this.findViewById(R.id.editVel);
        btnGrava = (Button)this.findViewById(R.id.btnGrava);

        btnConn.setOnClickListener(this);
        btnEnvia.setOnClickListener(this);
        btnGrava.setOnClickListener(this);
        btnControlador.setOnClickListener(this);

        mInitialized = false;
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mRot = mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);

        mSensorManager.registerListener(this, mRot, TAXA);
```

```
    }catch(Exception e){
        Log.e("ERROR", "erro no código: "+e.getMessage());
        e.printStackTrace();
    }

}

//callback assíncrono, chamado pela ConnectThread quando a conexão é estabelecida.
public void manageConnectedSocket(BluetoothSocket bs){
   connected = true;
   conexao = new Conexao(bs);

   this.setConnectedIcon(true);
   final BufferedReader br = new BufferedReader( new InputStreamReader(conexao.mmInputStream));
   new Thread(){
        @Override
        public void run(){
             while(true){

                  try {
                       if(br.ready()){
                            Log.e("PEDRO", "Mensagem redebida: "+br.readLine());
                       }
                  } catch (IOException e) {
                       e.printStackTrace();
                  }

             }
        }
   }.start();
}

public void setConnectedIcon(boolean status){
   final boolean s = status;
   this.runOnUiThread(new Runnable(){
       public void run(){
        checkBox.setChecked(s);
       }
   });
}

public void evento(View v){
   onClick(v);
}

public void controlaVel(int vel){
   //int correcao = 20;
   byte val = (byte)Math.min(MAX_PWM, Math.abs(vel));
   if(val == 20)
        val = 0;
   byte sign;
   if(vel < 0)
        sign = (byte)'0';
   else
        sign = (byte)'1';

   conexao.write(new byte[]{sign, val, (byte)'\n'});
}

@Override
public void onClick(View view){
   try{

        if(view.equals(btnConn)){
        if(bta == null){
             Log.e("ERROR", "Conexao cancelada, Não ha bluetooth adapter");
             return;
        }

        Intent it = new Intent(this, DeviceListActivity.class);
          startActivityForResult(it, REQUEST_CONNECT_DEVICE);
```

```
}else
if(view.equals(btnEnvia)){
     if(connected){
          int num = Integer.parseInt(editVel.getText().toString());

          controlaVel(num);
          msg("pacote enviado "+num);

     }
}else
if(view.equals(btnControlador)){
     //if(connected){
          if(!controllerOn){
               btnControlador.setEnabled(false);
               msg("Posicione o dispositivo");
               this.p = Float.parseFloat(((EditText)findViewById(R.id.P)).getText().toString());
               this.i = Float.parseFloat(((EditText)findViewById(R.id.I)).getText().toString());
               this.d = Float.parseFloat(((EditText)findViewById(R.id.D)).getText().toString());
               new Thread(){
                    public void run(){
                         try {
                                   Thread.sleep(6000);
                              } catch (InterruptedException e) {}
                         msg("Controle iniciado");
                    ref_theta = theta;
                    integrador = 0.0f;
                    msg(ref_theta+"");
                    controllerOn = true;

                    runOnUiThread(new Runnable(){

                         public void run(){
                              TextView uiR = (TextView)findViewById(R.id.refTheta);
                              uiR.setText(ref_theta+"");

                              btnControlador.setEnabled(true);
                         }
                    });
                    }
               }.start();
               btnControlador.setText("Parar controlador");
          }else{
               controllerOn = false;
               btnControlador.setText("Inicia controlador");
               controlaVel(0);
               integrador = 0;
          }

     //}
}else if(view.equals(btnGrava)){
          if(!gravando){
                    if(isExternalStorageWritable()){
                    gravando = true;
                    btnGrava.setText("Parar gravação");
                    dados = new float[10000];
                    tempo = new int[10000];
                    iDados = 0;
                    t0 = System.currentTimeMillis();
               }else
                    msg("Armazenamento externo nao disponivel para gravacao");
          }else{
               gravando = false;
               btnGrava.setText("Gravar");

               try{
                    File arq = new File(getExternalFilesDir(null), "datalog.txt");
                    OutputStream os = new FileOutputStream(arq);
                    PrintWriter pw = new PrintWriter(os);
                    pw.write("M = [");
                    for(int i = 0; i < iDados; i++){
                         //Log.e("SENS", tempo[i]+": "+dados[i]);
                         if(i == iDados -1)
                              pw.write(tempo[i]+" "+dados[i]+"];");
                         else
```

```
                                        pw.write(tempo[i]+" "+dados[i]+"\n");
                                   }
                                   pw.close();
                              }catch(IOException e){
                                   msg("gravacao falhou");
                              }
                         }
                    }

          }
          catch(Exception e){
               e.printStackTrace();
               Log.e("PEDRO", e.getMessage());
          }

     }

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@Override
protected void onActivityResult (int requestCode, int resultCode, Intent data){
     switch(requestCode){
        case REQUEST_CONNECT_DEVICE:

          if (resultCode == Activity.RESULT_OK) {
                String address = data.getExtras()
                                        .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
                BluetoothDevice device = bta.getRemoteDevice(address);
                thread = new ConnectThread(device);
                thread.start();

                //checkBox.setChecked(true);
            }
            break;
        case COD_REQ:
                if(resultCode == RESULT_OK){

                }else
                if(resultCode == RESULT_CANCELED){
                     msg("Não será possível conectar porque o Bluetooth deve estar ligado");
                }
                break;
     }

}

@Override
//Realiza todas as ações que devem acontecer quando um novo dado do sensor é recebido
public void onSensorChanged(SensorEvent event) {
     try{
          long t = System.currentTimeMillis();

          float x = event.values[0], y = event.values[1], z = event.values[2];
          float[] r = new float[9];
          SensorManager.getRotationMatrixFromVector(r, event.values);
          float[] orientation = new float[3];
          SensorManager.getOrientation(r, orientation);
          z = (float) (orientation[0]/Math.PI*180);
          x = (float) (orientation[1]/Math.PI*180);
          y = (float) (orientation[2]/Math.PI*180);
          theta = z;

          TextView barX = (TextView)findViewById(R.id.rotX);
          TextView barY = (TextView)findViewById(R.id.rotY);
          TextView barZ = (TextView)findViewById(R.id.rotZ);

          barX.setText(x+"");
          barY.setText(y+"");
          barZ.setText(z+"");

          if(controllerOn && (t - ultimoT > DELAY_ENVIO)){
```

```
                float deltaT = (t-ultimoT)/1000.0f;
                if(deltaT > 2) deltaT = 0;
                ultimoT = t;

                float delta_theta = ref_theta - theta;
                while(delta_theta > 180)
                     delta_theta -= 360;
                while(delta_theta < -180)
                     delta_theta += 360;

                //float vel = Math.min(MAX_PWM, delta_theta * p);
                float erro = delta_theta/180*(float)Math.PI;
                float derivada = 0;
                if(deltaT > 0)
                     derivada = (erro - ultimoErro) / deltaT;
                if(Math.abs(derivada) > 1)
                     derivada = 1*Math.signum(derivada);
                ultimoErro = erro;
                integrador += deltaT*erro;
                float sinalControle = -(p*erro + integrador*i + d*derivada)* 255.0f / 7.5f;
                TextView uiDelta = (TextView)findViewById(R.id.deltaTheta);
                uiDelta.setText(erro+"");
                TextView uiInt = (TextView)findViewById(R.id.integrador);
                uiInt.setText(integrador+"");
                TextView uiC = (TextView)findViewById(R.id.sinalControle);
                uiC.setText(sinalControle+"");

                if(connected)
                     controlaVel((int)sinalControle);
           }

           if(gravando && iDados < dados.length){
                tempo[iDados] = (int)(t-t0);
                dados[iDados++] = z;
           }
      }catch(Exception e){
           Log.e("ERROR", e.getMessage());
      }
}

public boolean isExternalStorageWritable() {
       String state = Environment.getExternalStorageState();
       if (Environment.MEDIA_MOUNTED.equals(state)) {
           return true;
       }
       return false;
  }

//Encerra a conexao bluetooth quando o aplicativo termina
protected void onStop(){
     super.onStop();
     try {

          unregisterReceiver(receiver);
          if(conexao != null)
               conexao.cancel();
          if(thread != null && thread.isAlive())
               thread.cancel();
          this.setConnectedIcon(false);
     } catch (Exception e) {
          e.printStackTrace();
          Log.e("PEDRO", e.getMessage());
     }
}

protected void onPause(){
     super.onPause();

     mSensorManager.unregisterListener(this);
}

  protected void onResume() {
     super.onResume();
```

```
        mSensorManager.registerListener(this, mRot, TAXA);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Adiciona itens à barra de ação, se houver
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    //Mostra uma caixa de texto por um periodo breve na tela
    private void msg(final String st){
        runOnUiThread(new Runnable(){
            public void run(){
              Toast.makeText(MainActivity.this, st, Toast.LENGTH_SHORT).show();
            }
        });
    }

}
```