PROJETO DE GRADUAÇÃO

Sistema de Controle de um VANT

Bruno Castro da Silva
Priscila Ortolan Libardi

Brasília, 08 de Julho de 2016

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
ENE – Departamento de Engenharia Elétrica

PROJETO DE GRADUAÇÃO

Sistema de Controle de um VANT

Bruno Castro da Silva
Priscila Ortolan Libardi

RELATÓRIO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA

Aprovada por

Prof. D. Sc. Ricardo Zelenovsky, PUC-RJ, UnB/ENE

Orientador

Prof. Eduardo Peixoto Fernandes da Silva, UnB/ ENE

Examinador

Prof. Edson Mintsu Hung, UnB/ ENE

Examinador

Brasília, 08 de Julho de 2016

FICHA CATALOGRÁFICA

SILVA, BRUNO; LIBARDI, PRISCILA
Sistema de Controle de um VANT [Distrito Federal] 2016
X, XXp., 210 x 297 mm (ENE/FT/UnB, Engenharia Elétrica).
Monografia de Graduação – Universidade de Brasília, Faculdade de Tecnologia Departamento de Engenharia Elétrica
1. – Sistema de Controle de um VANT
2. – Veículo Aéreo Não Tripulado

I. ENE/FT/UNB
II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

SILVA, B.; LIBARDI, P. (2016). Sistema de Controle de um VANT, Relatório de Graduação em Engenharia Elétrica, publicação XXXXXXXXX, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, XXp.

CESSÃO DE DIREITOS

AUTORES: Bruno Castro da Silva, Priscila Ortolan Libardi

TÍTULO: Sistema de Controle de um VANT

GRAU: Engenheira Eletricista

ANO: 2016

_____
Bruno Castro da Silva

_____
Priscila Ortolan Libardi

UnB – Universidade de Brasília

Campus Universitário Darcy Ribeiro

FT – Faculdade de Tecnologia

ENE – Departamento de Engenharia Elétrica

Brasília – DF – 70919-970

Brasil

Dedicatória

Dedico este trabalho a Deus, aos meus familiares e também a todos aqueles que contribuíram para que, de algum modo, estes anos de faculdade fossem o mais proveitosos possíveis. Este trabalho não seria possível sem vocês.

Bruno Castro da Silva

Dedicatória

Dedico este trabalho primeiramente a Deus, pois é quem dá sentido a todas as coisas, à minha família, que sempre me apoiou e caminhou comigo, e a todos que fizeram parte destes longos anos de graduação.

Priscila Ortolan Libardi

Agradecimentos

Agradeço à minha família, que sempre esteve ao meu lado com seu constante suporte à conquista dos meus diversos sonhos, em especial, aos meus pais por todos os sacrifícios e votos de confiança dados durante a minha vida para que eu pudesse me tornar uma pessoa de bem e capaz de superar desafios. Agradeço aos meus amigos mais próximos, que por diversas vezes me deram o incentivo necessário para que eu tivesse a paciência e calma necessárias para não desviar o foco dos meus objetivos acadêmicos e pessoais, em especial a meus amigos de curso, por todas as noites viradas de estudo, provas, trabalhos e relatórios feitos, dificuldades e facilidades. Agradeço aos professores e facilitadores do departamento de engenharia elétrica da UnB, pelas lições de vida, suporte e conhecimentos passados nestes árduos anos de graduação, em especial ao professor Ricardo Zelenovsky pela oportunidade que me foi dada de participar deste projeto, e todas as ajudas recebidas no decorrer do mesmo. Agradeço à Enetec – Consultoria Júnior em Engenharia Elétrica, que foi um pilar fundamental para a minha formação como engenheiro, que me proporcionou, em inúmeras ocasiões, experiências profissionais e pessoais que me prepararam melhor para o mercado de trabalho e ratificaram os conhecimentos previamente adquiridos em sala de aula.

Bruno Castro da Silva

Agradecimentos

Agradeço a Deus em primeiro lugar, o maior responsável por eu estar aqui hoje. À minha mãe e meu pai, que desde sempre se esforçaram para dar a mim e à minha irmã a melhor educação que podiam, além de todo o apoio, paciência e compreensão prestados ao longo dos anos. Agradeço à Luisa, minha irmã, que sempre ouviu com amor às dificuldades que eu enfrentava no curso e que quis se solidarizar escolhendo também a engenharia. Aos meus queridos amigos, que sabem quem são, que tanto me ajudaram com este trabalho em diversos sentidos, fossem técnicos, de logística ou mesmo emocional. Agradeço à Enetec, que me formou não só como engenheira, mas como profissional e cidadã. Aos amigos de curso, com quem dividi ansiedade pré-prova, decepção pós-prova e incontáveis risadas. Agradeço aos professores, que de fato me formaram engenheira nestes anos de graduação e em especial ao professor Ricardo Zelenovsky que aceitou o desafio que compreendia este projeto, sendo sempre muito atencioso e disposto a nos ajudar.

Priscila Ortolan Libardi

RESUMO

Este projeto apresenta o desenvolvimento de um sistema de controle de um veículo aéreo não tripulado – VANT implementado para funcionar em um *smartphone* de sistema operacional Android. A proposta é viabilizar o controle por meio de dispositivos móveis, possibilitando ao usuário decidir por controlar o veículo pela *touchscreen* ou utilizando o giroscópio do dispositivo. Um aplicativo para a plataforma Android é a interface utilizada pelo usuário e foi desenvolvido pelos autores utilizando o *software* Android Studio. Uma placa Arduino Uno realiza todo o controle do sistema; nela são gerados quatro sinais PWM que, tratados com um filtro passa-baixas, substituem os potenciômetros dos *joysticks* do controle original. O aplicativo se comunica com a placa por um sistema de comunicação *Bluetooth,* que permite ao usuário pilotar o *drone*. Mais funções podem ser acrescidas ao aplicativo, de forma a utilizar o VANT em trabalhos futuros.

Palavras-chave: *Arduino, Android Studio, Bluetooth, VANT, controle, giroscópio, drone.*

ABSTRACT

This paper presents the development of a control system for an unmanned aerial vehicle – UAV implemented to work on an Android operating system smartphone. The proposal is to enable its control from mobile devices, making it possible to the user to choose to control the vehicle on the touchscreen or using the gyroscope sensor of the device. An application on Android platform is the interface used by the user and it was developed by the authors of this work on the software Android Studio. An Arduino Uno board executes the control of the system, where four PWM signals are generated and then treated in a low pass filter, replacing the potentiometers of the original control joysticks. The application communicates with the board through a Bluetooth communication system, which allows the user to pilote the drone. It is also possible to add more functions to the app in order to use the UAV in future projects.

Keywords: *Arduino, Android Studio, Bluetooth, UAV, control, gyroscope, drone.*

Sumário

LISTA DE FIGURAS

1. Introdução

1.1. Ambientação:

O ser humano sempre nutriu o desejo de voar, de dominar não só a terra como também os céus. Não podemos porém explicar esse fenômeno interior do ser sem entrar em uma profunda reflexão antropológica sobre o assunto, o que não compete a este trabalho. Fatos históricos porém comprovam que desde muito tempo se deram inúmeras tentativas de tirar os pés do homem do chão. Na Grécia antiga mesmo, em 400 A.C., um estudioso construiu o que se tem notícia de ser uma das primeiras tentativas que obtiveram melhor resultado: um pombo de madeira impulsionado por um jato de ar e que planava por um tempo. Logo depois surgiu a pipa, o mais rudimentar planador, invenção dos chineses, em 300 A.C.; mas os primeiros estudos e o primeiro projeto sério de uma máquina capaz de voar carregando uma pessoa surgiu com o italiano Leonardo da Vinci no século XIV. Estes projetos porém nunca chegaram a ser construídos e o primeiro voo bem sucedido foi de um balão de ar quente (A Passarola), de Bartolomeu de Gusmão no ano de 1709 em Lisboa.

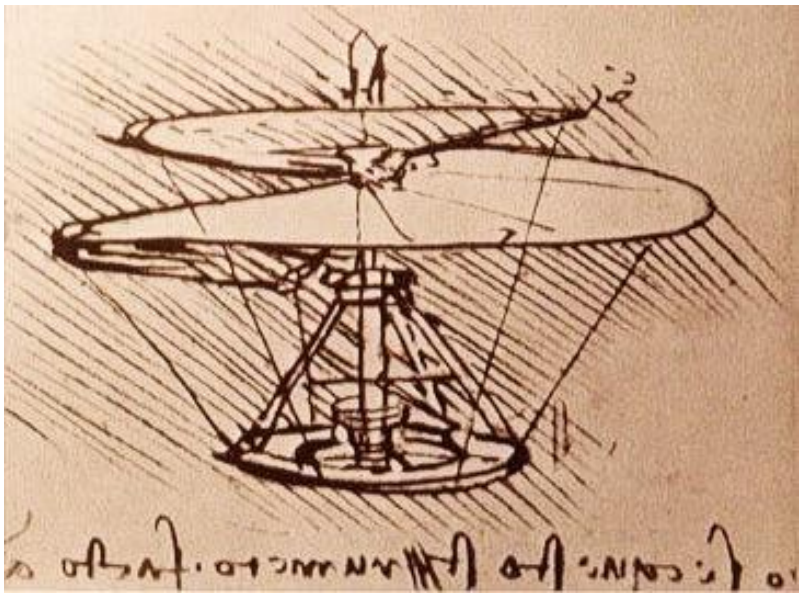

Figura 1 - Máquina voadora projetada por Leonardo da Vinci

O primeiro voo realizado com sucesso, de acordo com a mídia europeia e norte-americana, foi protagonizado pelo brasileiro Santos Dumont, em Paris na França no ano de 1906. O 14-Bis, famoso avião que marcou este momento histórico, foi projetado e construído por ele, sendo o primeiro avião na história da aviação a ter seu voo homologado por uma instituição pública aeronáutica, o Aeroclube da França. O destaque a este voo se deve principalmente por não ter dependido de nenhum tipo de exterioridade, como ventos e propulsões não-próprias. A tecnologia dos aviões avançou surpreendentemente durante a Primeira Guerra Mundial: criaram-se motores mais poderosos e melhorou-se a aerodinâmica, além das adaptações de fins bélicos.

Os anos que se seguiram, principalmente o período entre-guerras, foram então marcados por rápidos avanços neste ramo e linhas aéreas começaram a operar. Assim, gerou-se um crescente impacto sócio econômico mundial. Durante a era de ouro da aviação, década de 1930 em especial, melhoraram-se consideravelmente os equipamentos de controle e a tecnologia de rádio-telecomunicação, permitindo seu uso na aviação gerando técnicas de navegação aérea mais precisas, como o próprio piloto automático. Nas décadas seguintes com a corrida armamentista, marca da Guerra Fria, foram criadas tecnologias militares como o voo automático de mísseis de longa distância e aeronaves de espionagem. Neste ponto, a história da aviação e da robótica já se misturava: tais tecnologias foram as que mais contribuíram para o desenvolvimento da automatização do transporte aéreo comercial contemporâneo, influenciando diretamente o surgimento de aeronaves robóticas.

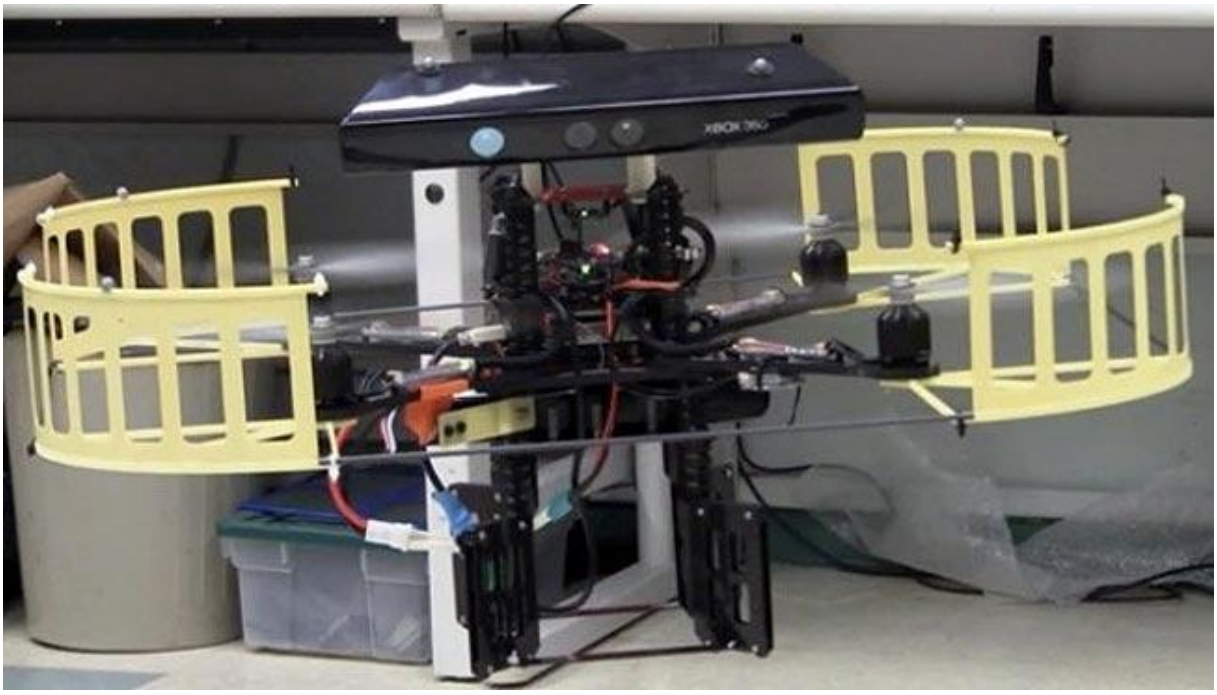

Figura 2 - Robô aéreo Pelican, da empresa alemã Ascending Technologies

Hoje, há uma imensa variedade de robôs aéreos: vão desde os VANTs (Veículos Aéreos Não Tripulados) até os VTOL (*Vertical Take-off and Landing*). Estes últimos são os helicópteros convencionais com rotor principal e de cauda e também os quadrotores, dotados de quatro motores; enquanto os primeiros são aviões tradicionais em tamanho reduzido e tema principal deste trabalho. Independente do nome que leva, um robô aéreo é assim definido por seus sistemas embarcados, ou seja, a tecnologia que permite o desempenho autônomo de suas tarefas. O avanço da microeletrônica, da tecnologia de sensores e da confecção de baterias contribuem para minimizar o volume e o peso desses sistemas, assim como a computação, que utiliza algoritmos para substituir boa parte dos circuitos elétricos e melhorar a estabilidade da máquina. As altas velocidades dos processadores modernos garantem que o desempenho e a precisão no ajuste da altitude sejam feitos praticamente em tempo real. Assim, esses robôs vem sendo utilizados nas mais diversas aplicações: patrulhas de fronteiras, aerofotogrametria, inspeção e resgate e várias outras, o que demonstra sua tendência de uso em um futuro que já começou [2].

1.2. Motivação:

Muitos centros de pesquisa ao redor do mundo vem desenvolvendo tecnologias para aeronaves robóticas. Os VANTs e os drones são os principais alvos desses projetos. Apesar do avanço tecnológico conhecido e da vasta aplicação em que esse tipo de equipamento já é utilizado, percebe-se que, no Brasil, ele ainda é pouco popular, o que se deve principalmente ao seu elevado custo. Além disso, são poucos ainda os centros de pesquisa e universidades que desenvolvem linhas de pesquisa voltadas à essa tecnologia de robôs aéreos e, por isso, se achou pertinente aprofundar os estudos nessa área.

Assim, na busca de um projeto que permitisse o estudo e desenvolvimento de um sistema de controle para o VANT, achou-se conveniente utilizar um desses modelos mais populares, disponíveis no mercado e de menor custo. Idealizou-se um sistema completo: controle humano por meio de gestos em uma tela, bem como o mesmo controle por sensores como giroscópio; GPS embarcado e automatização de rota; adaptação para transporte de pequenos objetos. Este trabalho se comprometeu com a primeira parte deste sistema, dando início a um projeto que pode ser a base para futuros sistemas de transporte, de segurança e muitos outros.

1.3. Objetivos:

Este projeto tem como objetivo desenvolver um sistema de controle de um VANT, de forma a conseguir enviar os comandos de um *smartphone*, tanto por gestos na tela como por medidas feitas pelo giroscópio do mesmo. Esses comandos são enviados via *Bluetooth* para um Arduino UNO, que realiza a interface entre o usuário e o controle do *drone*, que é reutilizado a fim de aproveitar a comunicação RF original. Assim, o desenvolvimento se deu tanto em hardware como em software, para Arduino e aplicativo Android.

1.4. Estrutura do Trabalho:

Os capítulos deste trabalho estão organizados da seguinte maneira:

- *Capítulo 1 – Introdução: Ambientação, motivação e apresentação dos objetivos do projeto.*
- *Capítulo 2 – VANT: Motivação para escolha do VANT, especificações e tomadas de decisão para desenvolvimento do projeto.*
- *Capítulo 3 – Hardware: Modificações da placa original, embasamento teórico e desenvolvimento de circuito adicional.*
- *Capítulo 4 – Software - Android: Definição de funcionalidades e desenvolvimento do aplicativo.*
- *Capítulo 5 – Software - Arduino: Definição de funcionalidades e desenvolvimento.*

- *Capítulo 6 – Ensaios e testes: Apresentação dos resultados obtidos dos ensaios e testes realizados após o desenvolvimento.*
- *Capítulo 7 – Conclusão: Resumo do projeto, conclusões e sugestões para trabalhos futuros.*

2. VANT

Um VANT, criteriosamente falando, é assim entendido principalmente pelo fim que tem: não-recreativo, ou seja, é um equipamento utilizado para pesquisa, experimentos ou fins comerciais. Além disso, ele precisa possuir uma carga útil embarcada que não seja necessária para voar. O nome *drone* é mais informal e engloba qualquer objeto aéreo e não tripulado, sendo mais utilizado quando se trata de um equipamento de lazer e esporte, assim como um aeromodelo. De forma ainda mais específica, existem os RPAs, "*Remoted-Piloted Aircraft*" (aeronave remotamente pilotada), que como o próprio nome sugere, são os VANTs que possuem um piloto remoto. Neste trabalho, os nomes VANT e *drone* serão utilizados como sinônimos, visto que o modelo escolhido para o projeto é um modelo de lazer sendo utilizado em pesquisa.

2.1. Syma X5SW

O projeto baseia-se na modificação do sistema de controle de um *drone*. Sendo assim, se fez necessário adquirir um modelo do mercado, pois a construção de um excederia toda a verba destinada ao trabalho e acabaria por demandar demasiado tempo para uma atividade não ligada ao tema do mesmo. O Syma X5SW mostrou-se o equipamento com melhor custo-benefício para atender aos planos estabelecidos.

Figura 3 - Syma X5SW

O Syma X5SW é um dos mais novos e avançados modelos entre os quadricópteros da série Syma X5. Está entre os mais dispendiosos dentre os de sua categoria, apesar da variação de preço ser pequena. Considerado um excelente *drone* para iniciantes em aeromodelismo, foi o primeiro modelo a apresentar vídeo FPV (*First Person View* ou Visão em Primeira Pessoa) por conexão WiFi. O Syma X5SW se destaca principalmente por seu preço de mercado:

aproximadamente U$60, contra preços exorbitantes de modelos semiprofissionais que chegam a U$1400, como o *DJI Phantom 4*.

Além do que mencionado acima, o modelo conta com uma grande popularidade virtual, o que contribui na busca por informações e até mesmo por acessórios e peças extras caso necessário. Concluiu-se então, que as qualidades deste modelo de *drone* tornavam suas deficiências relativamente insignificantes e poderiam ser contornadas durante o projeto. Citando-as rapidamente: duração da bateria (3,7V 500mAh) extremamente baixa resultando em um tempo de voo de 6 a 8 minutos e do atraso na transmissão FPV de cerca de 3 segundos, que torna inviável o controle feito diretamente por meio da mesma. Quatro baterias extras tornaram os testes possíveis, assim como o carregador USB que foi adquirido posteriormente.

2.2. Especificações

As especificações técnicas do *drone* utilizado neste projeto, o Syma X5SW, estão listadas abaixo [3]:

- Dimensões: 310x310x105mm
- Tamanho do motor: 8x20mm
- Peso: 120g (com câmera e bateria)
- Tempo de voo
  - De 6 a 8 minutos
  - Aproximadamente 4 minutos com a câmera ligada
- Bateria: 3,7V 500mAh LiPo
- Tempo de carga: 40-45 minutos
- Distância de controle: aproximadamente 50m
- Transmissão em 2.4GHz
- Bateria do transmissor: 4x1,5V AA
- Câmera de 2MP
- Transmissão WiFi em tempo real – FPV

2.3. Sistema de Controle de um VANT

Como já foi mencionado, o objetivo deste trabalho é o desenvolvimento do sistema de controle de um VANT, automatizando-o para oferecer flexibilidade em aplicações futuras. A ideia inicial de automatização seria possibilitar ao usuário pilotar o *drone* utilizando o próprio celular, por meio de um aplicativo intuitivo que tornasse esse controle simples. Portanto, decidiu-se por modificar o sistema de controle original do Syma X5SW, reaproveitando a transmissão de Rádio Frequência (RF).

A proposta para o projeto se traduziu em uma plataforma fixa onde o antigo controle remoto foi controlado por um Arduino UNO. Ele é responsável pela geração dos sinais analógicos, que antes eram gerados pelos potenciômetros que compõem o *joystick*, que comandarão o VANT. Tais sinais são gerados à medida em que o Arduino recebe, por *Bluetooth*, a mensagem contendo as informações enviadas pelo piloto no *smartphone*. O esquemático abaixo representa o sistema desenvolvido:

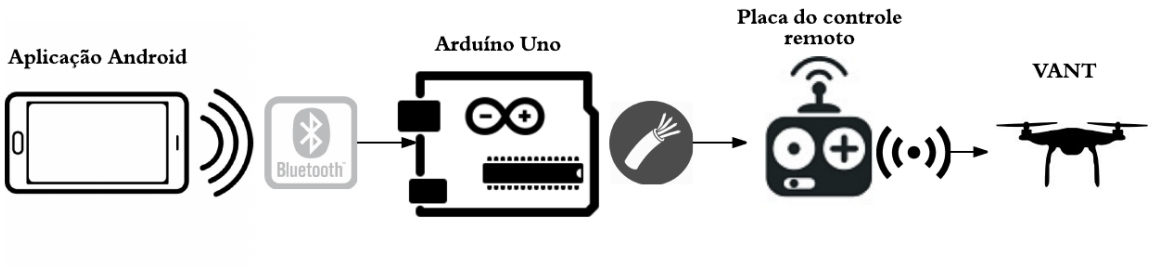

Figura 4 - Esquemático de funcionamento do sistema

A seguir, são detalhados os componentes utilizados no desenvolvimento do projeto.

2.3.1. Placa do Controle Remoto

A ideia de manter a placa utilizada no controle remoto é importante para que a comunicação RF entre controle e *drone* fosse mantida. Assim, com algumas modificações na placa do controle, foi possível substituir os *joysticks* e controlá-la por um microcontrolador como o Arduino, escolhido aqui para realizar esta função. O controle original é formado apenas por esta placa, além da carcaça.

Figura 5 - Controle Remoto do Syma

Os comandos convencionais de direção e altitude são executados pelos dois *joysticks* e seguem a lógica descrita em seguida. O *joystick* da esquerda controla a potência dos motores no sentido de determinar a altitude do quadricóptero (cima e baixo) além de executar a sua rotação em torno do próprio eixo. Enquanto isso, o da direita controla a movimentação no que se pode chamar de plano x-y: para frente e para trás, para a direita e para a esquerda. O controle também conta com alguns outros botões que realizam diversas outras funções, tais como *loop* de 360 graus, seleção de velocidades, troca de modo de voo, entre outros. Um esquemático dessas funcionalidades e seus respectivos botões pode ser verificado na Figura 6, bem como a antena, o botão liga/desliga, a tela LCD, a luz indicadora e os *joysticks*.

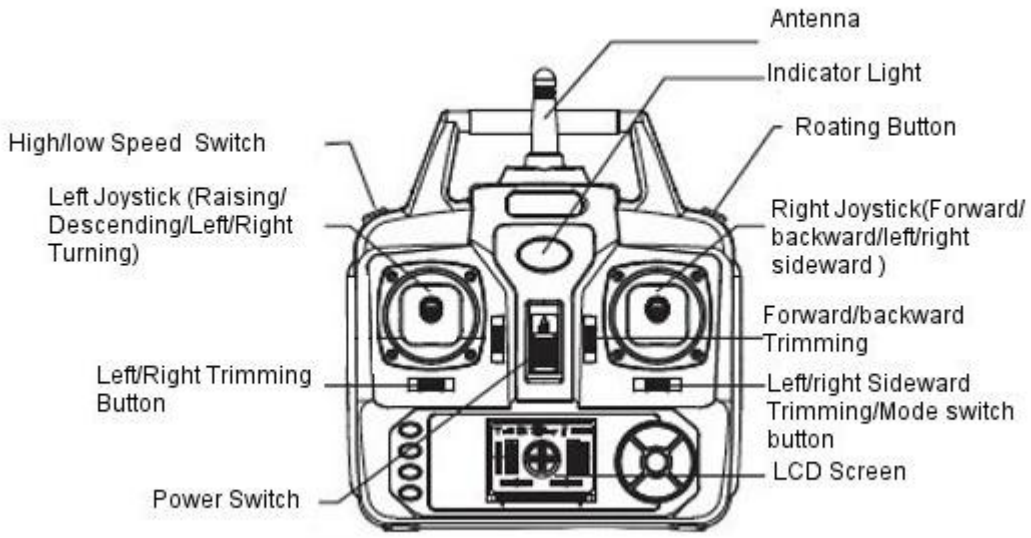

Figura 6 - Esquema indicador de funções do controle (Manual do drone)

Dentre todas essas funções, apenas o botão liga/desliga, os controles de movimentação e a tela LCD serão de fato utilizadas no presente trabalho. Isso porque são elas que realizam as atividades fundamentais de todo sistema de controle. Olhando a placa de forma mais atenta, verifica-se que o design é bem organizado e de lado único, o que é esperado, já que torna a produção mais barata. Ela conta principalmente com um microcontrolador *Winbond* 8 *bits* da série W79E804 e um chip de memória EEPROM.

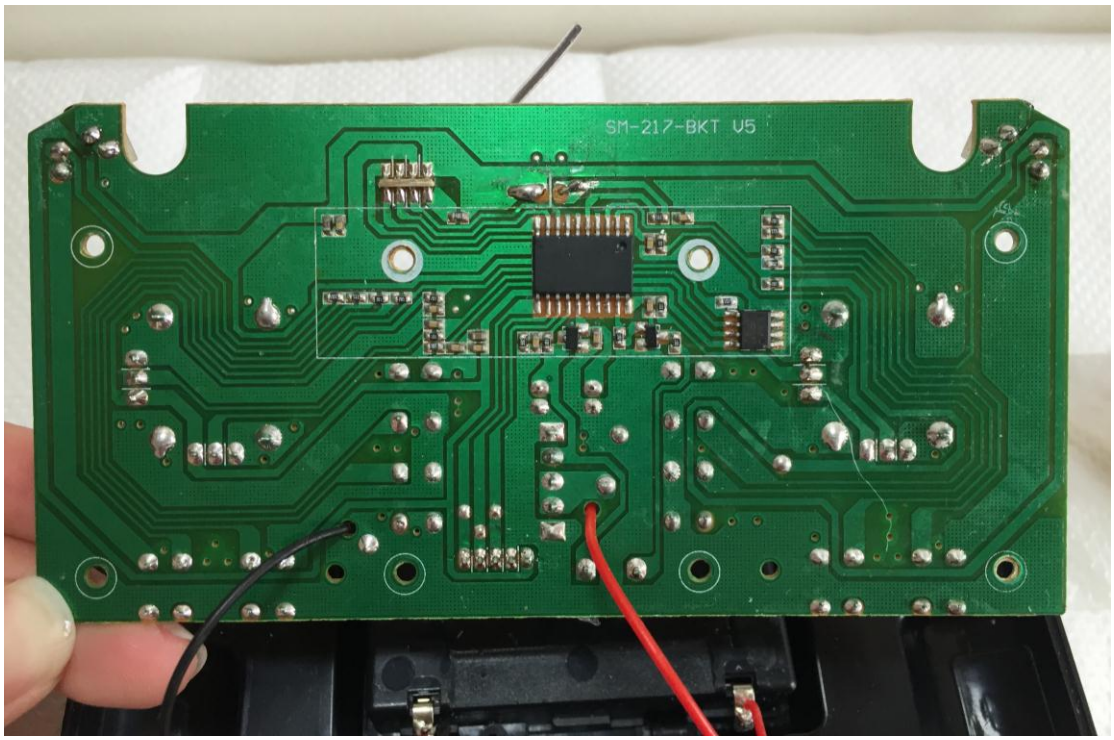

Figura 7 - Placa do controle original

2.3.2. Arduino UNO

Arduino é uma plataforma de prototipagem de código aberto em hardware e software. Geralmente projetada com microcontroladores Atmel AVR de 8, 16 ou 32 *bits* com suporte de entrada e saída embutido e linguagem padrão de programação semelhante à C/C++ essencialmente. Tem a finalidade de criar ferramentas acessíveis e de baixo custo, para que novos usuários tenham facilidade de se adaptar à ela e fazer uso de forma prática.

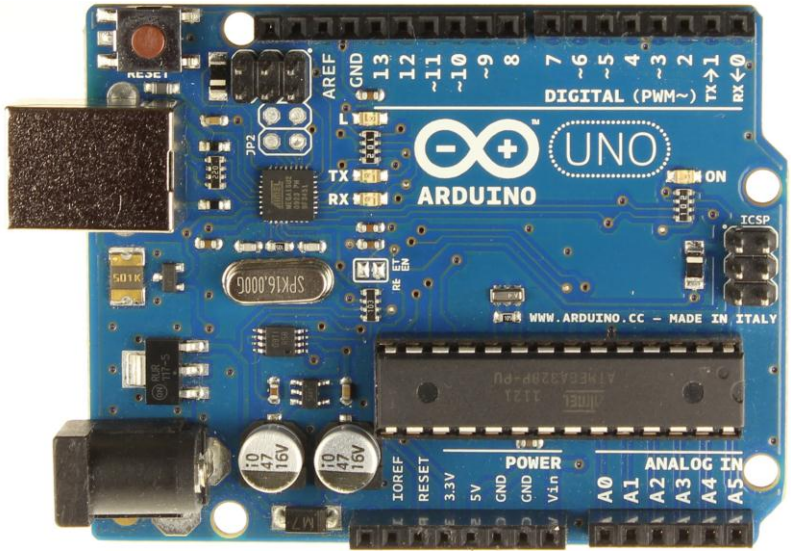

Figura 8 - Plataforma Arduino UNO

O Arduino Uno é o modelo utilizado neste projeto. Este microcontrolador é uma placa baseada no ATMega328P, e conta com 14 pinos de entrada/saída digital, dos quais 6 podem ser usadas como saídas de PWM (*pulse width modulation*), 6 entradas analógicas, um cristal de quartzo de 16 MHz, conexão USB, *jack* de energia e um botão de reset. A placa Uno é a primeira de uma série de placas Arduino USB e o modelo de referência da plataforma Arduino [4].

O ATMega328P possui 3 *timers*, sendo os *timers* "0" e "2" de 8 *bits* e o *timer* "1" de 16 *bits*. Todos podem ser configurados para a geração de um sinal de PWM em até 2 pinos da plataforma Arduino cada. Este processador também provê comunicação serial UART nos pinos digitais 0 (RX) e 1 (TX). Além disso, fazendo uso da biblioteca *SoftwareSerial*, é possível realizar comunicação serial em qualquer um dos pinos digitais do Uno.

O ambiente de desenvolvimento integrado (IDE) padrão para compilação do código é a Arduino Software e é disponibilizada no site da companhia: *arduino.cc/en/Main/Software*. Por ela também é feito o *upload* do código para a plataforma.

Muitas destas configurações já vem no padrão de fábrica e, portanto, são automaticamente ajustadas. Elas seriam, por exemplo, a definição do *clock* para 16MHz. A Arduino Software também é uma ferramenta imprescindível para a realização de testes: um monitor serial que, quando habilitado, permite que simples mensagens de texto sejam enviadas da placa e para ela, sendo muito útil para testes principalmente.

A figura a seguir é o diagrama de pinos do Arduino UNO:

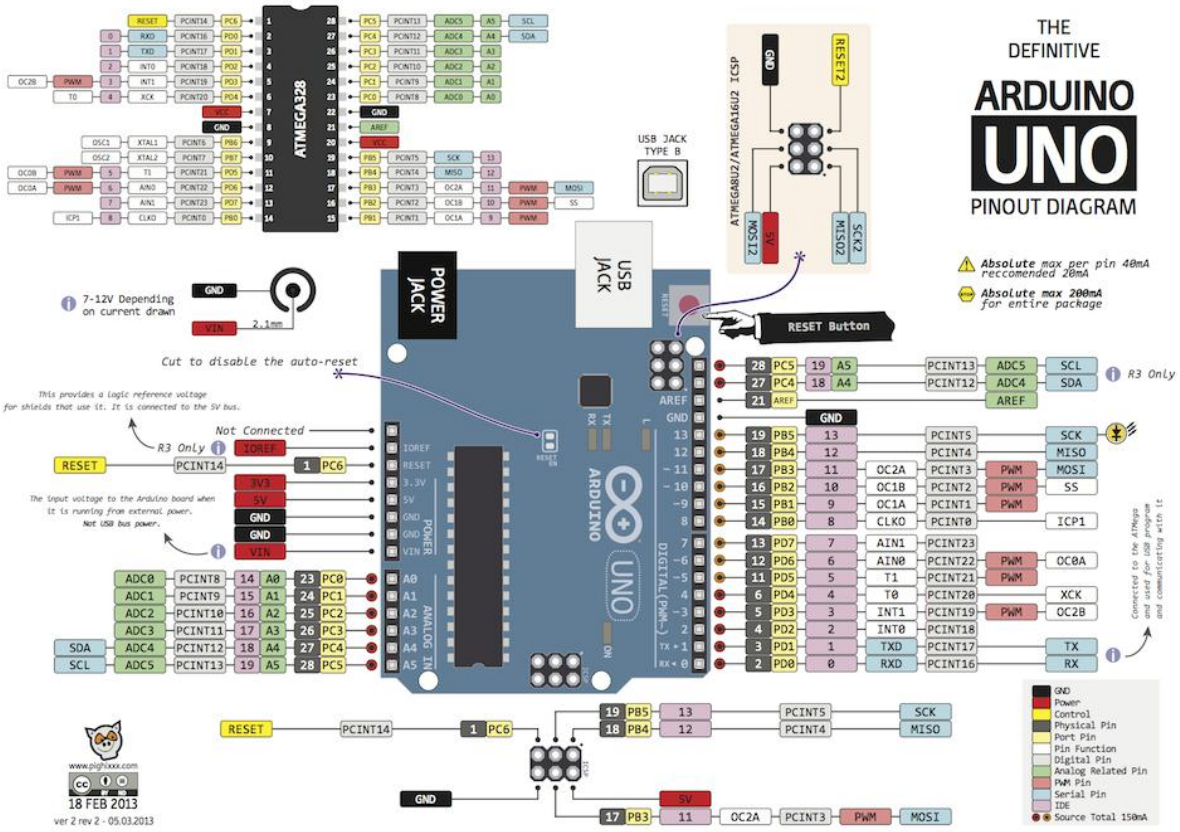

Figura 9 - Diagrama de pinos Arduino Uno (Fonte: http://www.pighixxx.com/test/)

2.3.3. Aplicativo Android

Android é um sistema operacional (SO) móvel desenvolvido pela Google. Hoje, essa é a plataforma mobile mais utilizada. De acordo com os dados do site Statista, até Julho de 2013, foram feitos mais de 50 bilhões de downloads de aplicativos pelo mundo e o SO está presente em 85% dos aparelhos em mais de 190 países [5].

O Android tem como base o núcleo Linux, que é um dos melhores exemplos de software livre, provendo o fundamento para outros softwares livres, como é o caso do Android: intencionalmente e explicitamente um software de código livre.

A Figura 10 apresenta a estrutura do código fonte da plataforma, que são descritas abaixo:

- Aplicativos: são aplicativos e jogos desenvolvidos em Java;
- Framework: conjunto de APIs (*Application Programming Interface*) que permitem o desenvolvimento fácil e rápido de aplicativos. Consiste em um conjunto de ferramentas para o design da interface com o usuário, tais como botões, campos de texto, campos para imagens e ferramentas para acionar outros aplicativos, telas ou mesmo para abrir arquivos;
- Bibliotecas e serviços nativos: recursos nativos do Android utilizados para desenvolvimento e interpretação/execução de aplicativos;
- Linux: inclui todos os drivers de hardware e redes, sistemas de arquivos e processamento;

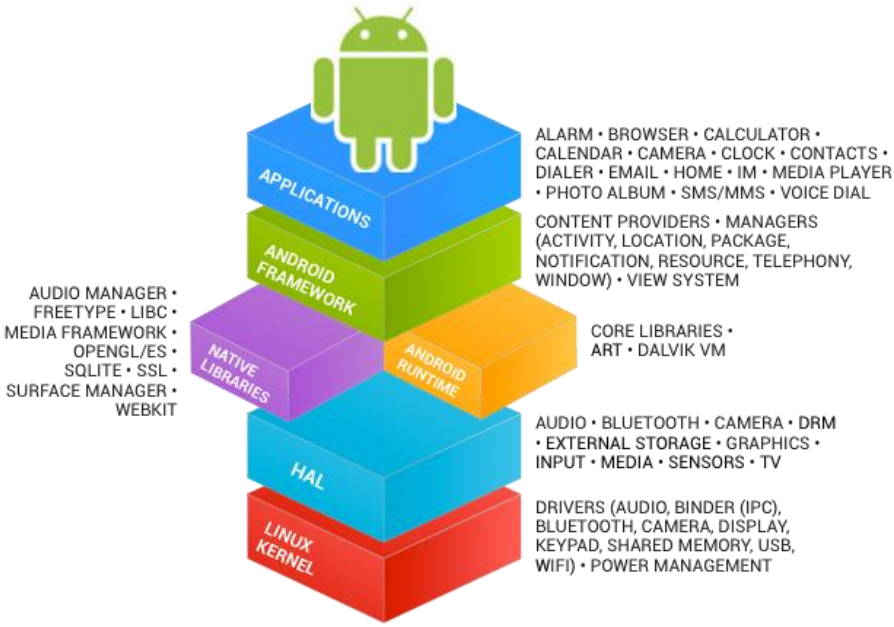

Figura 10 - Estrutura do código fonte do Android (Fonte: https://source.android.com)

Aplicativos estendem a funcionalidade do dispositivo e são desenvolvidos na linguagem JAVA, por meio do sistema de desenvolvimento do software Android (SDK), que providencia as ferramentas necessárias ao ambiente de desenvolvimento integrado. A IDE oficial utilizada para a criação de aplicativos Android é atualmente o Android Studio, disponibilizado gratuitamente sob a licença Apache 2.0. Sua primeira compilação estável foi lançada em Dezembro de 2014, na versão 1.0 [6]. Neste projeto em específico, foi utilizada a versão 1.5.1 desta IDE (disponível em *https://developer.android.com/studio/index.html*). Os testes podem ser rodados em um emulador disponível pelo Android Studio, mas também em um dispositivo real.

Desenvolvedores contam com um endereço web oficial de suporte ao desenvolvimento de aplicativos Android. Disponível em *developer.android.com*, o conteúdo vai da documentação da API e guias de design até maneiras de distribuir e monetizar o aplicativo criado.

3. Hardware

3.1. Embasamento teórico

O foco do projeto a ser desenvolvido é, com o auxílio do Arduino, alguns filtros e adaptações na placa, simular as funcionalidades básicas do controle para que se possa pilotar o *drone* ou, em outras palavras, ter domínio remoto de seus *joysticks*.

Após aberto e analisado o controle remoto, tomou-se conhecimento de que cada *joystick* era composto por um par de potenciômetros perpendiculares entre si: o *joystick* da esquerda com um não-retrátil na vertical (não retorna para a sua posição inicial), conforme o esperado, tendo em vista que este é o responsável pela ascensão perpendicular do veículo, e outro retrátil na horizontal (retorna para a posição inicial); e o da direita tendo ambos retráteis.

Potenciômetros são resistores cujo valor de resistência pode ser variado, girando-se um eixo que movimenta um contato móvel. Possuindo 3 terminais, a resistência entre suas 2 extremidades é fixa em seu valor nominal, enquanto que este valor entre uma extremidade e sua derivação central muda de acordo com a posição do segundo. A figura a seguir retrata o funcionamento acima descrito, onde A e B representam as extremidades fixas, e W seu cursor móvel [7].

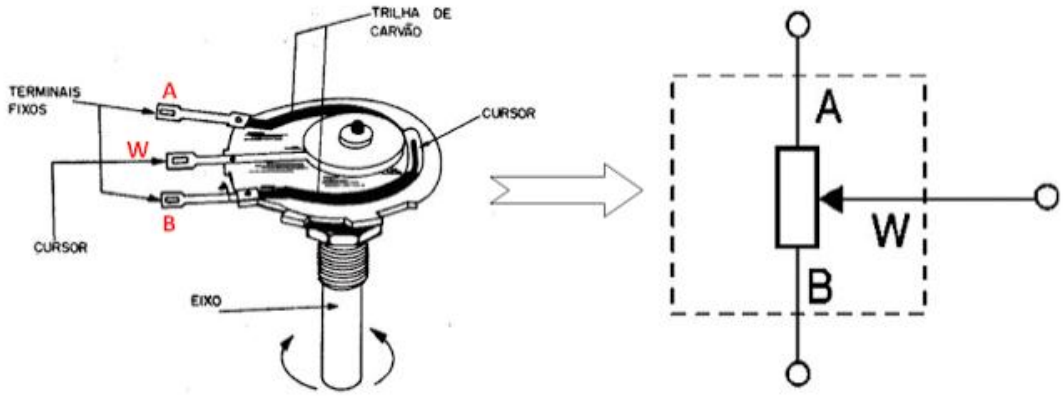

Figura 11 - Esquemático de um potenciômetro

Segundo a Lei de Ohm, tem-se que: "*A tensão entre os terminais de uma resistência é diretamente proporcional à corrente que flui através dela*", ou seja, percebe-se que, se mantida a corrente constante, a relação entre a resistência e a tensão nos terminais dos potenciômetros em questão é diretamente proporcional [8]. Visto que o arduíno não possui recursos para operar diretamente como um DAC (conversor digital-analógico), surgiu a ideia de utilizá-lo em modo *Fast* PWM aliado a um filtro RC para gerar diretamente as tensões desejadas, de modo que fosse possível fornecer de forma analógica a tensão que seria gerada pelo movimento dos *joysticks*. Dessa maneira poder-se-ia desconectar os potênciometros do restante do circuito da placa.

A modulação por largura de pulso, mais conhecida como onda PWM, é uma técnica para se obter um sinal analógico através de um digital. Uma onda quadrada é gerada digitalmente variando o seu valor entre mínimo (0 Volts) e máximo (5 Volts). Este padrão *on/off* nos permite simular tensões desejadas entre 0 e 5 Volts apenas alterando a proporção de tempo em que o sinal se mantém ligado, duração esta denominada como largura de pulso da onda. A programação do Arduino será comentada mais à frente, no capítulo 5.

Na figura a seguir, as linhas verdes representam o período da onda PWM, cuja duração é de aproximadamente 2ms, e *duty cycle* (ciclo de carga) é a mencionada proporção de tempo em que a onda está em seu valor máximo [9].

Para calcular nossa tensão média em cada valor de ciclo de carga, basta multiplicar este pela tensão máxima aplicada, de 5Volts para o nosso caso, ou seja:

$$V_{média} = V_{máxima} * Duty\ Cycle$$

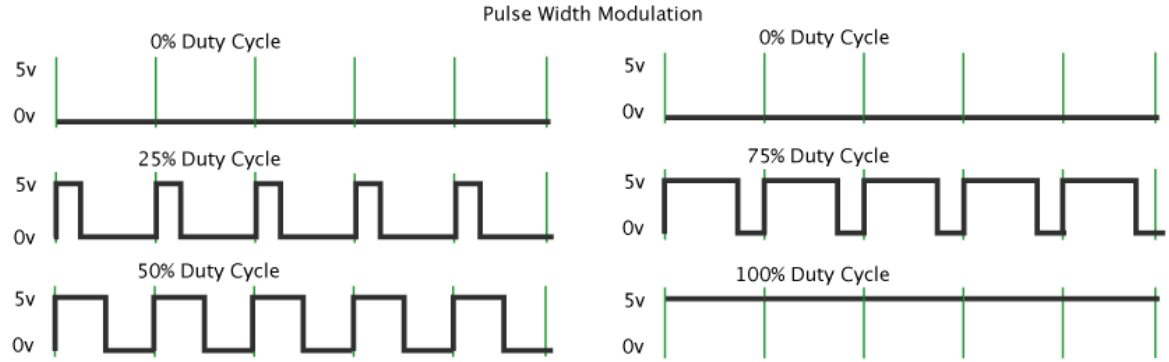

Figura 12 - Comportamento do pwm de acordo com a variação do duty cycle

Os sinais vistos acima podem ser decompostos, cada um, em duas componentes: uma DC e uma outra onda quadrada de mesmo *duty cycle*, mas com uma amplitude média valendo zero. A ideia por trás da conversão DA (digital-analógica) visada em nosso circuito é passar a onda gerada em um filtro passa-baixas para remover a maior parte dos componentes de alta frequência.

Analisando a situação na época do projeto, notou-se que a utilização de um filtro simples de primeira ordem já satisfaria as necessidades do projeto. Veremos mais à frente que não foi preciso a obtenção de valores analógicos extremamente precisos para a construção do controle. Outra vantagem do uso desse é a eliminação de possíveis harmônicos de alta frequência presentes em nossa onda PWM.

Idealmente, um filtro passa-baixas possuiria um ganho constante e igual a 1 durante toda a banda passante F_{bw} que imediatamente cairia para zero assim que a atingisse. Tais filtros de alta precisão são extremamente caros e difíceis de se construir, então, por motivos financeiros e práticos, montou-se um cujo comportamento é demonstrado e comparado com o ideal na figura a seguir.

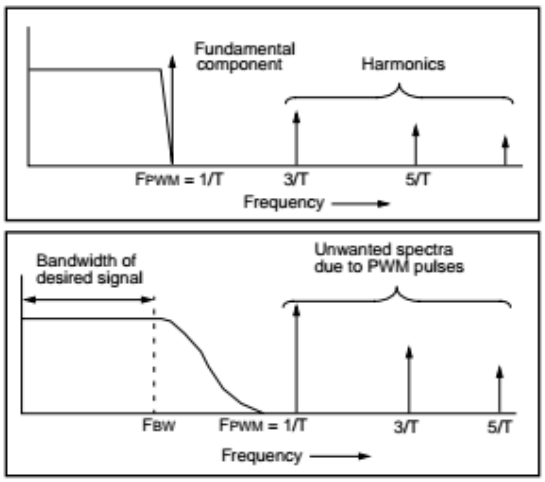

Figura 13 - Resposta em um filtro passa-baixas ideal

Nosso passa-baixas consiste em um resistor de 10kΩ e um capacitor de 100nF conectados em série, ambos baratos e de fácil obtenção em qualquer loja de circuitos eletrônicos. Assim, a constante de tempo desse circuito é $\tau = 1\ ms$ e a resposta no tempo da tensão sobre o capacitor pode ser vista na figura a seguir:

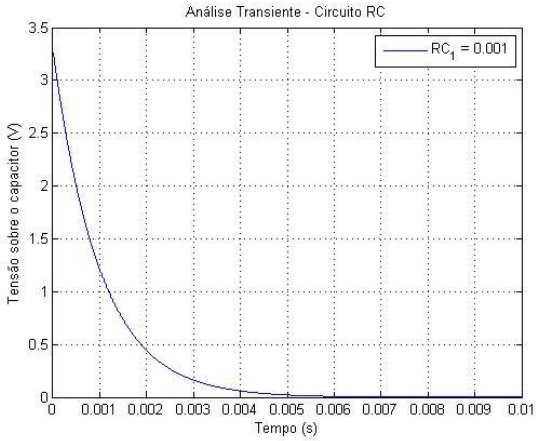

Figura 14 - Análise transiente do circuito RC

Para se certificar de que a frequência da onda gerada F_{pwm} será necessariamente maior do que a F_{bw}, basta configurar o clock do Arduino para operar em uma dada frequência grande o suficiente.

A frequência de banda passante é dada por:

$$F_{bw} = \frac{1}{RC} = \frac{1}{10 * 10^3 * 100 * 10^{-9}} = 100hz$$

Na descrição das configurações em que o Arduino trabalha em nosso projeto, veremos adiante que a condição das frequências é respeitada com $F_{pwm} = 7{,}81kHz$. A figura 15 mostra a FFT da onda PWM configurada nessa frequência, mostrando que de fato a constante de tempo do filtro é capaz de retirar toda a parte AC gerada pelo PWM.

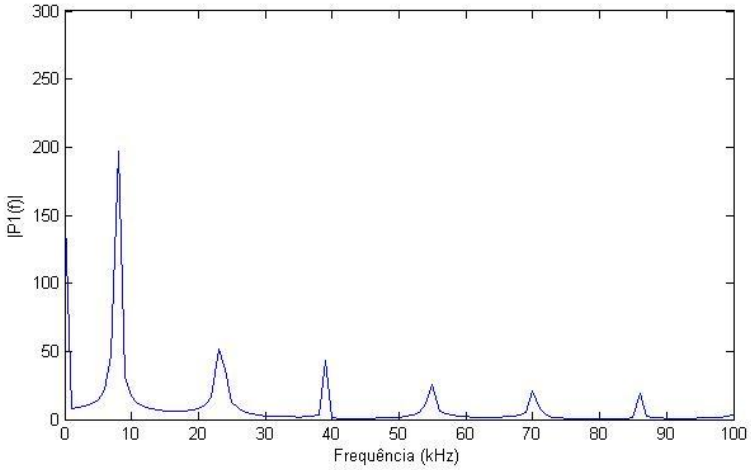

Figura 15 - Resposta em frequência do PWM

3.2. Montagem do protótipo

Para organizar a montagem de nosso experimento, fez-se a adaptação de todos os circuitos, placas e Arduino para que estes nos permitissem realizar testes e para que fossem mais simples a medição de valores de tensão e possíveis correções de erros. Além destas modificações, foram feitos pequenos orifícios nas margens da placa original, do circuito e do Arduino para que estes fossem presos em uma base no novo sistema por parafusos e porcas.

3.2.1. A placa:

Remove-se um trecho da trilha do circuito que liga o potenciômetro ao restante do controle na intenção de instalar um *switch* nos controles analógicos esquerdo e direito, para que se pudesse alternar entre controlar o sistema via Arduino e via *joystick* do controle original. Deste modo, a comparação entre saída esperada e saída real do circuito se tornou extremamente mais simples e rápida.

Dado que o controle original não possuia controle de estabilidade, os resultados esperados não foram sempre os triviais. Um bom exemplo deste fato é observado se acionado apenas o controle da aceleração do motor. Nota-se que o VANT a todo momento se desloca para a direita, logo, a observação deste evento em um teste traduz um êxito na implementação do mesmo.

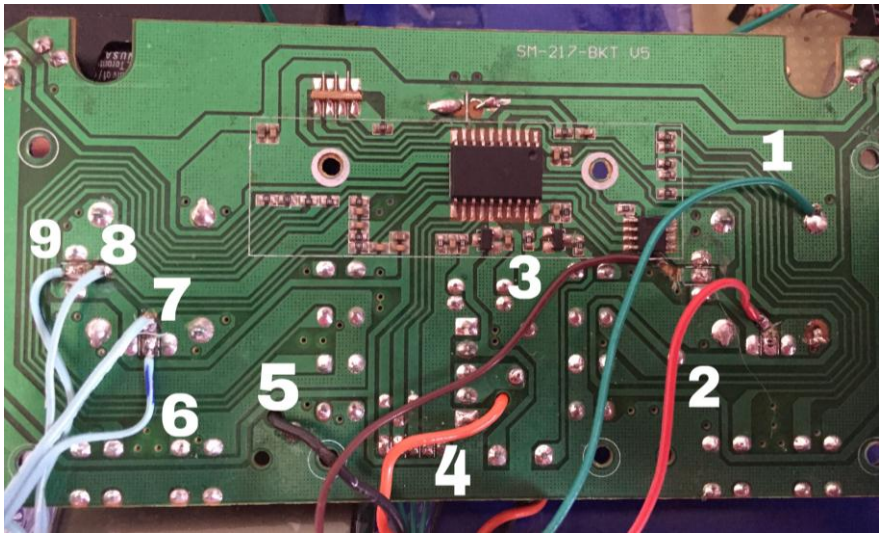

Figura 16 - Placa original com adaptações

3.2.2. O circuito adicional:

A montagem do circuito adicional se deu em uma placa padrão onde todos os componentes, fios e filtros foram soldados para assegurar a resiliência do sistema.

Este conta com:

- Quatro circuitos que controlam sua respectiva direção de entrada única V_{in} e saída V_{out}: cada um com seu próprio filtro passa-baixas, mas possuindo uma referência comum a todos;

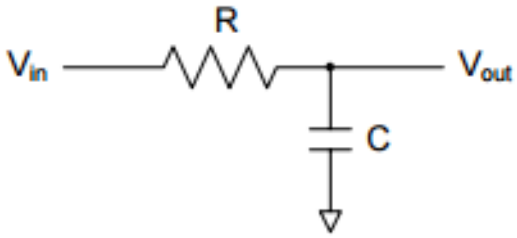

Figura 17 - Esquemático de um filtro passa-baixas de primeira ordem

- Um módulo *Bluetooth*, HC-05, que foi implementado para realizar a comunicação entre a interface Android e o microcontrolador. De baixo custo, o HC-05 é frequentemente utilizado em projetos com Arduino. Ele se comunica por um perfil chamado SSP (*Serial Port Profile*), que é como uma porta serial sem fio. Ademais, pode funcionar em modo mestre, aceitando a conexão de outros, ou modo escravo.
  São 6 pinos disponíveis para uso: WAKEUP e STATE (não utilizados), VCC e GND (5V e 0V, respectivamente), TXD (saída do transmissor), RXD (entrada do transmissor) [10].

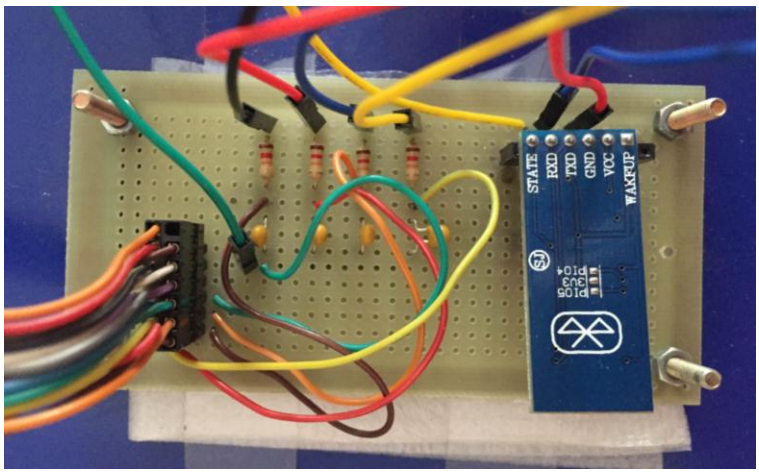

Figura 18 - Circuito adicional: filtros e módulo Bluetooth HC-05

3.2.3. Os demais componentes:

- Arduino: nenhuma mudança significativa foi feita no hardware do Arduino;
- Conjunto de baterias: um soquete simples para quatro pilhas AA foi usado para o arranjo da fonte de alimentação do sistema;
- Display LCD: herdado do controle original, este visor mostra os comandos recebidos pelo controle em tempo real.

Ao final da montagem, tem-se o resultado final do protótipo:

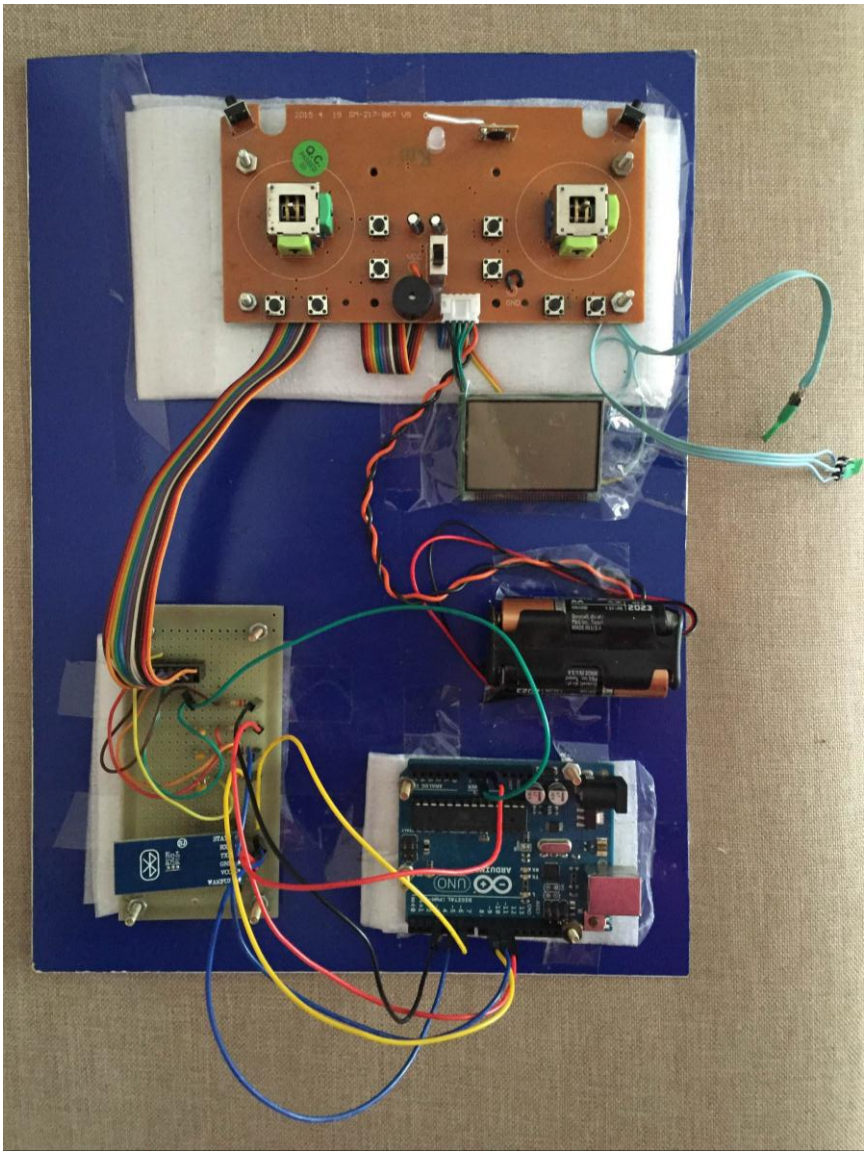

Figura 19 - Plataforma fixa final do protótipo

4. Software - Android

Com o objetivo de criar um aplicativo que funcionasse como a interface entre o usuário e o *drone* a ser controlado, foi utilizado o Android por sistema operacional. Este foi escolhido pelas facilidades que apresenta um *software* livre, como já mencionado anteriormente. Para criar um projeto com o Android Studio, algumas configurações iniciais são necessárias, tais como nome do aplicativo, domínio da companhia ou pacote onde ficarão os códigos, local do projeto dentro do computador e a versão mínima da plataforma que o projeto irá suportar. Elas foram definidas da seguinte forma:

- Nome da aplicação: VANT CTRL;
- Domínio: com.example.android.vantcrl;
- SDK mínimo: API 18: Android 4.3.1;

Versões mais baixas de API atendem mais dispositivos, mas tem menor número de atributos disponíveis. O dispositivo Android utilizado para testes é um Samsung GT-I9300 com Android 4.3, o que justificou a escolha da API mínima, de forma a permitir ao aplicativo rodar neste dispositivo.

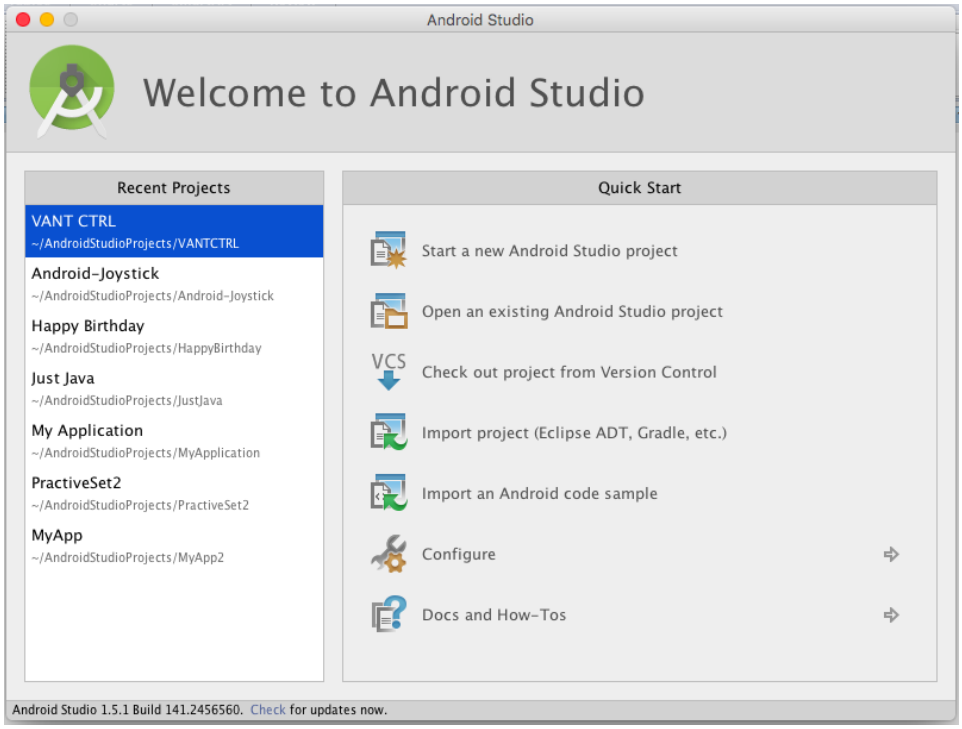

Figura 20 - Tela de abertura do Android Studio

Um aplicativo Android é dividido em *activities*, que na prática são as diferentes telas pelas quais o usuário navega, já que toda tela deve estender uma *activity*. Cada *activity* é formada por pelo menos um arquivo Java e um arquivo xml. O primeiro contem a lógica a ser utilizada, armazenando os diversos métodos a serem usados na tela em questão em uma mesma classe; o segundo é responsável pelo design da mesma tela, ou seja, a interface gráfica. A *activity* realiza a interação entre esses arquivos e entre outras funcionalidades e bibliotecas. Normalmente uma *activity* é definida como a principal, sendo esta apresentada ao

usuário quando o aplicativo é iniciado pela primeira vez. Cada *activity* pode iniciar uma outra à medida que seja necessário realizar atividades distintas e mudar de tela. Além disso, sempre que uma nova *activity* começa, a anterior é parada, porém preservada pelo sistema em uma pilha. Dessa maneira, sempre que é iniciada, a nova é lançada para o início da pilha para ter a atenção do usuário. Ao pressionar o botão de retorno, a atual é retirada da pilha e destruída e a *activity* anterior volta a ser o foco (seguindo o princípio LIFO: *last in, first out*).

Quanto às transições entre uma *activity* e outra, elas são realizadas por meio de métodos de callback que funcionam como ganchos que podem se sobrepor uns aos outros para realizar o trabalho mais apropriado quando o estado dela se altera. Esses métodos definem o ciclo de vida de uma *activity* [11]. São eles:

- onCreate(): chamado quando a *activity* é criada pela primeira vez. É onde deve ser realizada toda a configuração estática, como a criação do layout da tela correspondente;
- onRestart(): chamado depois que a *activity* parou, imediatamente antes de ser iniciada novamente;
- onStart(): chamado imediatamente antes de a *activity* ser visível ao usuário;
- onResume(): chamado logo antes de a *activity* interagir com o usuário;
- onPause(): chamado quando o sistema está para retomar outra *activity*;
- onStop(): chamado quando a *activity* não é mais visível ao usuário
- onDestroy(): chamado antes que a *activity* seja destruída;

A Figura 21, a seguir, ilustra o ciclo de vida de uma *activity*.

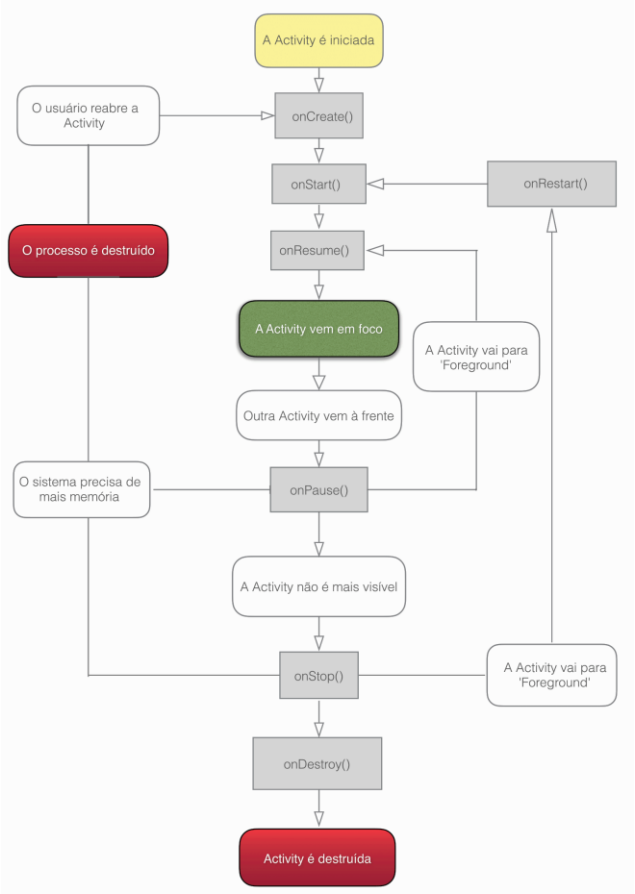

Figura 21 - Ciclo de vida de uma activity

No projeto do VANT CTRL são utilizadas quatro *activities*:

- MainScreen: *activity* principal, tela inicial do aplicativo;
- DeviceListActivity: activity para configuração de conexão *Bluetooth*;
- porToque: *activity* de controle por toque;
- porGyro: *activity* de controle utilizando o giroscópio do aparelho;

4.1. Activity MainScreen

A *activity* denominada MainScreen é responsável pela tela principal do aplicativo, a tela que aparece assim que ele é iniciado. Nela, além do nome do aplicativo e dos autores, estão dispostos três botões por entre os quais o usuário pode navegar: "CONTROLE POR TOQUE", "CONTROLE POR GIROSCÓPIO" e "CONFIGURAR BLUETOOTH".

Figura 22 - Tela da activity MainScreen

Esta tela funciona de modo bem intuitivo, já que apenas redireciona o aplicativo por entre as demais *activities* do projeto. Isso significa que o arquivo Java que rege a MainScreen também é bastante simples: composto de apenas três métodos além do *onCreate* e do *onResume*. Cada um desses três métodos é chamado sempre que seu respectivo botão é acionado, iniciando a *activity* correspondente. Os botões são definidos como no arquivo activity_main_screen.xml, onde também é definido o método da *activity* que cada botão aciona.

Os métodos encontrados no arquivo Java da *activity* MainScreen são:

- onCreate(): apenas configura o layout da tela chamando o arquivo .xml correspondente;
- onResume(): pega o endereço MAC da DeviceListActivity via *intent*;

- porToque(): inicia a *activity* porToque quando chamado pelo usuário ao apertar o botão correspondente;
- porGyro(): inicia a *activity* porGyro quando chamado pelo usuário ao apertar o botão correspondente;
- configBT(): inicia a *activity* DeviceListActivity quando chamado pelo usuário ao apertar o botão correspondente;

O único requisito para bom funcionamento das demais *activities* é que da tela principal se vá para a configuração do *Bluetooth*, já que tanto o controle por toque quanto o controle por giroscópio precisam do *Bluetooth* pronto para operação.

4.2. Activity DeviceListActivity

Ao ser pressionado o botão "Configurar *Bluetooth*" na MainScreen, a *activity* DeviceListActivity é iniciada. Nela, são realizadas as primeiras configurações de *Bluetooth*, imprescindíveis para o bom funcionamento do aplicativo. Assim que iniciada esta *activity*, o usuário é questionado se permite acionar a conexão *Bluetooth* caso o aparelho esteja com essa opção desativada. Em seguida, são listados na tela todos os aparelhos pareados que o dispositivo guarda na memória. Se o HC-05 (módulo *Bluetooth* do Arduino) não estiver na lista, o pareamento precisará ser feito primeiramente nas configurações de *Bluetooth* do dispositivo, e só após estarão satisfeitas as condições para o uso do aplicativo.

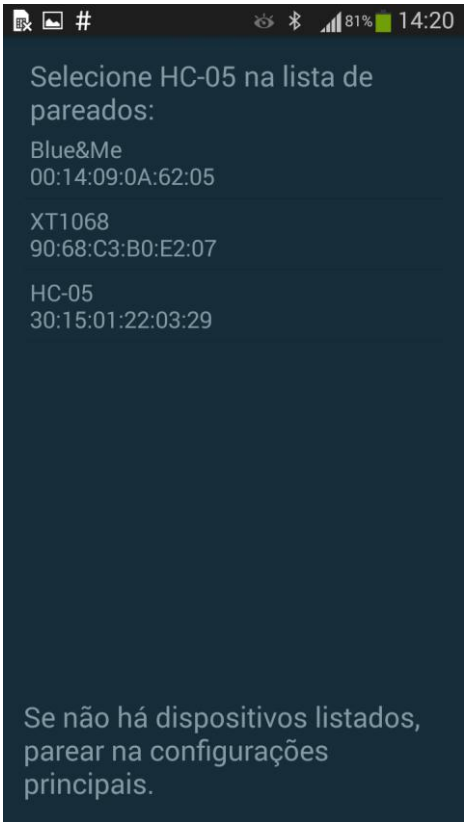

Figura 23 - Tela da activity DeviceListActivity

Esta *activity* em especial lida com dois arquivos de layout: activity_device_list.xml e device_name.xml. O primeiro é responsável pelo design como o vemos, ou seja, o texto de cima, a lista em si e o texto de baixo, enquanto que o segundo é utilizado para escrever o nome do dispositivo pareado e é então adicionado à lista.

O código Java desta *activity* executa as primeiras configurações para a conexão *Bluetooth*, viabilizando a troca de dados com outros dispositivos. O framework da aplicação provê acesso a essa funcionalidade por meio das APIs *Bluetooth* do Android, permitindo ao aplicativo buscar por dispositivos *Bluetooth* emparelhados ou disponíveis na área, conectar os dispositivos e transferir dados entre eles. Todas as APIs *Bluetooth* são encontradas no pacote *android.bluetooth*. Além disso, o *Bluetooth* precisa ser ativado por programação, e isso é feito dando as devidas autorizações no arquivo AndroidManifest.xml: o arquivo responsável por definir as características do projeto, tais como versão, logo, nome, componentes e permissões.

Os métodos da *activity* em questão, a DeviceListActivity, são:

- onCreate(): apenas configura o layout da tela chamando o arquivo .xml correspondente;
- checkBTState(): checa se o aparelho tem *Bluetooth* e se está ligado e pede permissão para ligar caso não esteja;
- onResume(): é onde acontece a configuração do *Bluetooth* e será detalhado em seguida;

Por comporem métodos mais complexos, os algoritmos do checkBTState() e do onResume() são descritos de forma mais detalhada a seguir.

4.2.1. Método checkBTState()

Antes de tudo, é importante que o *Bluetooth* seja suportado pelo dispositivo e que esteja ativado. Sendo suportado, pode ser ativado de dentro da aplicação. Esse processo precisa do BluetoothAdapter, que representa o próprio adaptador *Bluetooth* do sistema. O método que retorna esse adaptador é o getDefaultAdapter(), que se retornar *null*, significa simplesmente que não é suportado e os recursos dependentes dele não poderão ser utilizados.

Em seguida, é preciso certificar que o *Bluetooth* esteja ativado. O método isEnabled() retorna falso se o mesmo estiver desativado e verdadeiro caso contrário. Para ativá-lo, é necessário que seja disparada uma requisição através das configurações do sistema sem interromper a aplicação. Isso é feito pelo startActivityForResult() com uma intent de ação (ACTION_REQUEST_ENABLE). Uma caixa de diálogo solicita permissão ao usuário para ativar o *Bluetooth*.

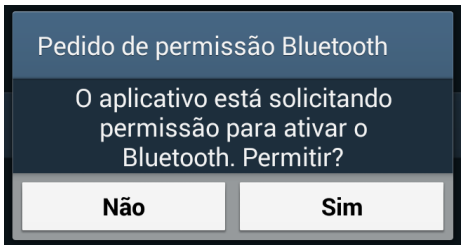

Figura 24 - Caixa de diálogo de solicitação do Bluetooth

4.2.2. Método onResume()

Como citado anteriormente, o onResume() é um método de callback e é chamado logo antes que a *activity* interaja com o usuário. Assim, para a DeviceListActivity, é importante que este método prepare o que é necessário ao usuário responder para ter sucesso na conexão *Bluetooth*. Por isso, o primeiro processo a ser realizado por este método é o checkBTState(),

que é chamado logo na primeira linha de código para verificar o suporte ao *Bluetooth* e já solicitar sua ativação caso necessário. Intuitivamente, o próximo passo é a busca na área local pelo dispositivo com o qual se deseja trocar dados. O aparelho que realiza a busca necessita de informações para iniciar uma conexão, tais como nome do dispositivo, classe e seu endereço MAC único. Para obtê-las, se faz uma solicitação ao usuário do dispositivo.

A busca por dispositivos descobertos não é realizada de dentro deste aplicativo em específico, apesar de ser possível. De forma a tornar o código mais simples, o aplicativo lista apenas os dispositivos pareados, ou seja, cuja conexão já tenha sido feita uma primeira vez. Quando emparelhado, as informações básicas sobre o dispositivo são salvas e podem ser lidas pelas APIs *Bluetooth* e a conexão pode ser iniciada sem necessidade de pesquisa. É importante perceber que um dispositivo emparelhado não necessariamente está conectado: o primeiro estado significa que os dois possuem uma chave de ligação compartilhada que pode ser usada para autenticação, enquanto o segundo quer dizer que os dispositivos compartilham um canal RFCOMM (*radio frequency communication*) e podem transmitir dados entre eles. De forma simples, o RFCOMM é um protocolo de rede que proporciona conexões simultâneas para dispositivos *Bluetooth* (mais informações em *developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx*).

Os dispositivos emparelhados são retornados em um set de BluetoothDevice's quando o método getBoundedDevices() é utilizado e são consultados por meio de uma Array-Adapter. Além do nome do dispositivo, o endereço MAC também é salvo como uma parte de um Array-Adapter e pode ser extraído mais tarde para iniciar a conexão [12].

Além desses métodos, um objeto da classe OnItemClickListener também é criado e utilizado dentro do método onResume(). Esse objeto executa uma ação quando se clica em um item: no caso, adquire o endereço MAC do dispositivo pareado selecionado pelo usuário, mostra o texto "Conectando..." na tela e dispara uma intent para chamar a MainScreen enquanto passa o endereço MAC para essa nova *activity* por meio de uma variável estática extra. Essa informação será utilizada em outras telas para o envio de dados.

4.3. Activity porToque

A *activity* porToque é responsável pelo controle via *touchscreen*. Ela é composta de um arquivo Java (porToque.java) e um arquivo de *layout* (activity_por_toque.xml). Para cumprir com sua finalidade, dois *joysticks* precisam ser simulados nela, de modo a funcionar como o controle original do Syma. Para isso, é utilizada uma *View* para Android customizada chamada JoystickView, disponível na sua página oficial de desenvolvimento no github [13]. A JoystickView é importada à raiz do projeto como um arquivo JAR, sendo o suficiente para referenciá-la e utilizá-la no projeto como ferramenta.

4.3.1. JoystickView

O *joystick* representado na JoystickView é composto de duas partes principais: a imagem de fundo, que corresponde ao círculo maior e é uma região do *joystick* e uma alavanca de *pop-up*, que corresponde ao círculo menor. Quando tocada e movida, a alavanca acompanha o movimento, imitando a própria coisa real.

Um listener fica atento às coordenadas da alavanca em relação ao *joystick*, que são referenciadas ao seu centro (0,0), e calcula a distância de um centro ao outro, bem como o ângulo. Além disso, uma variável do estado inicial da alavanca determina se ela deve voltar ao centro ou não [14].

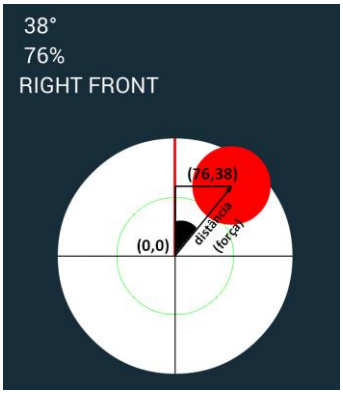

Figura 25 - Análise trigonométrica da distância de um centro ao outro

É importante perceber que para esta aplicação, o eixo que é referência ao ângulo é o eixo vertical, ou eixo das ordenadas, já que a direção padrão é a frente. Assim, tendo distância e ângulo, tem-se a informação da posição em coordenadas polares. Uma conversão para coordenadas retangulares será necessária mais a frente, para que a tensão equivalente seja aplicada na saída do que antes eram os potenciômetros do controle original. Além disso, três views de texto apresentam esses valores ao usuário: ângulo, força e direção. A direção é determinada de acordo com o quadrante para o qual a alavanca é movida.

O arquivo de *layout* desta *activity* é então composto de duas JoystickViews, para simular o *joystick* da esquerda e o da direta do controle original, e de seis views de texto para imprimir as informações na tela. Os métodos utilizados no arquivo Java são:

- onCreate(): configura o *layout* da tela chamando o arquivo .xml correspondente, muito além de configurar o *listener* dos *joysticks* e algumas configurações de *Bluetooth*, muito além de enviar as mensagens de acordo com o que é interpretado no *listener*;
- onResume(): basicamente realiza as configurações de *Bluetooth* para envio dos dados desta *activity*;
- checkBTState(): checa se o aparelho tem *Bluetooth* e se está ligado e pede permissão para ligar caso não esteja;

Também duas classes são criadas neste mesmo arquivo para auxiliar nas tarefas que os métodos operam:

- JoystickAtt: cria *views* de texto para os atributos do *joystick*;
- ConnectedThread(): é uma extensão da classe *thread*, e permite a conexão de uma nova *thread* para envio de dados de forma simultânea às outras tarefas da *activity*;

O método checkBTState() já foi detalhado na subseção anterior e portanto dispensa comentários extras. Os demais são comentados a seguir.

4.3.2. Método onCreate()

Dentro do método onCreate(), poucas configurações de *Bluetooth* são realizadas: apenas o método checkBTState() é chamado e o BluetoothAdapter padrão é salvo em uma variável para uso posterior. Um loop *for* é utilizado para criar quantos *joysticks* o desenvolvedor quiser, para que não fosse necessário adicioná-los manualmente e a todos os seus atributos, já que é possível replicá-los, tornando o código mais robusto. Nele, é configurado o *listener* do *joystick* e os textos de acordo com o que for detectado. Um *switch* de diferentes casos realiza essa função. Também dentro do *listener* a mensagem a ser enviada é formada, e ela tem a seguinte forma:

$$\#\ +\ n\acute{u}mero\ do\ joystick\ +\ "\ "\ +\ \hat{a}ngulo\ +\ "\ "\ +\ for\c{c}a\ +\ ;$$

#: indica o início da mensagem;
número do joystick: número 1 para o da esquerda e 2 para o da direita;
“ “: indica término da informação anterior;
ângulo: valor do ângulo do desvio de direção;
força: valor da força, ou magnitude da distância, do desvio em relação ao centro;
; :indica o fim da mensagem;

O método write() da classe ConnectedThread() é chamado para, por meio da *thread* inicializada nas declarações iniciais, enviar a mensagem por *Bluetooth* por uma *stream* de saída. Uma *thread*, em português encadeamento de execução, é de forma simplificada uma maneira de dividir um processo em diferentes tarefas que podem ser executadas paralelamente. Por meio dela, a *activity* consegue enviar a mensagem e detectar as novas posições do *joystick* ao mesmo tempo.

4.3.3. Método onResume()

Ao ser chamado, o método onResume() realiza basicamente configurações de *Bluetooth* necessárias à conexão. Como discorrido anteriormente, emparelhamento e conexão são coisas distintas e apenas o emparelhamento e seleção do dispositivo por meio da aquisição de seu endereço MAC foram realizadas na activity DeviceListActivity().

Os dispositivos são considerados conectados quando ambos possuírem um BluetoothSocket conectado no mesmo canal. *Streams* de entrada ou saída são obtidos por cada dispositivo e a transferência de dados se inicia. Os procedimentos de conexão são diferentes para cliente e servidor, sendo que um dispositivo deve abrir um servidor e o outro precisa iniciar a conexão. É possível preparar cada dispositivo como servidor para que cada um tenha um *socket* aberto e à espera de conexões, mas podendo também se tornar um cliente, iniciando a conexão. Porém, neste caso em específico, o módulo *Bluetooth* do Arduino é configurado como servidor, então a aplicação é preparada como cliente [12].

De forma mais prática, essa conexão é iniciada criando-se um objeto BluetoothDevice que represente o dispositivo remoto. Com ele, se obtém um socket por meio do método createRfcommSocketToServiceRecord(UUID), em que UUID é simplesmente a identificação do serviço com o qual se deseja conectar. Com o socket, se inicia a conexão através da chamada ao método connect() e se a busca pela UUID for bem sucedida e o dispositivo aceitar a conexão, o canal RFCOMM é compartilhado e o método retorna. Com o sucesso da conexão, a *thread* é iniciada.

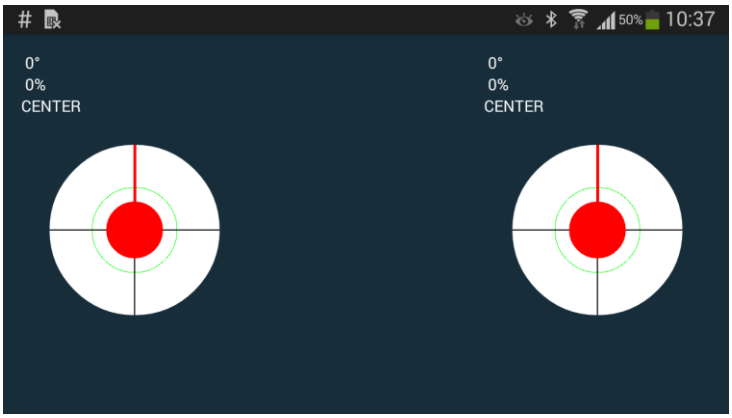

Figura 26 - Tela da activity porToque()

4.4. Activity porGyro

A *activity* porGyro é muito similar à *activity* porToque(), já que ambas tem o mesmo objetivo final de controlar o VANT, enviando mensagens que definem os movimentos a serem reproduzidos pelo mesmo. Também formada por um arquivo Java (porGyro.java) e um arquivo de *layout* (activity_por_gyro.html), esta *activity* se diferencia da anterior porque o *joystick* da direita é substituído pelo sensor giroscópio do aparelho móvel. Desse modo, o único *joystick* na tela permanece responsável pelo movimento do *drone* na vertical e em torno do próprio eixo e a posição do celular, medida por esse sensor, fica responsável pelo deslocamento para frente e para trás e para os lados. O arquivo de *layout* é composto por um *joystick* e por seis *views* de texto que mostram os valores medidos pelo giroscópio em cada orientação.

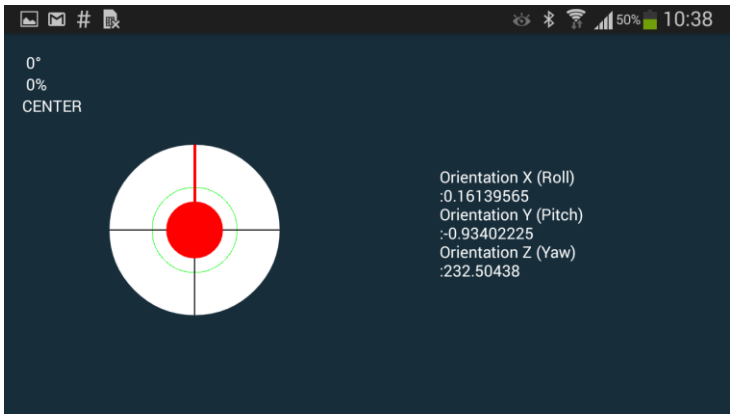

Figura 27 - Tela da activity porGyro

O arquivo Java, por sua vez, é composto dos métodos onCreate() e onResume() que funcionam praticamente da mesma forma que os mesmos métodos da *activity* anterior, com algumas particularidades, além de métodos próprios do sensor giroscópio:

- onCreate(): chama o arquivo de *layout*, configurando o listener para apenas um *joystick*, e inicializando as views de texto do sensor;
- onResume(): responsável pela conexão *Bluetooth* propriamente dita, além de configurar o *listener* para o sensor;
- onStop(): desconfigura o *listener* do sensor;

- onSensorChanged(): modifica o texto das *views* para alterações no sensor, além de chamar o método que envia a mensagem;
- transformAndSend(): recebe os valores medidos pelo sensor, os converte para coordenadas polares e envia utilizando um objeto da classe ConnectedThread();

Devido à grande similaridade dos métodos de callback com os mesmos da *activity* porToque e da simplicidade do onSensorChanged(), estes não serão detalhados. O método transformAndSend(), por sua vez, é explorado a seguir para que fique clara a forma com que as medidas do sensor são realizadas e como esses dados são tratados para fornecerem as informações da maneira necessária ao entendimento e processamento no Arduino.

4.4.1. Método transformAndSend()

O giroscópio mede a taxa de rotação em rad/s em torno dos eixos x, y e z de um aparelho. A rotação é positiva no sentido horário da vista de algum ponto da parte positiva do eixo x e o contrário para o eixo y, como ilustrado na figura abaixo [15]:

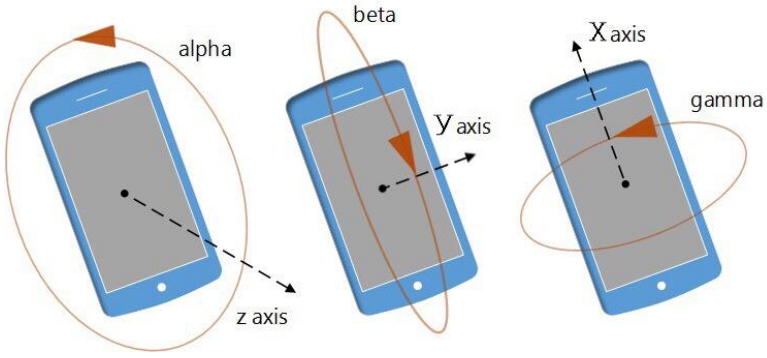

Figura 28 - Orientação dos eixos de um dispositivo

O eixo z não reflete nenhum tipo de comando na movimentação do *drone* e, portanto, pode ser descartado. Assim, os dados obtidos em x e y são os únicos que necessitam de tratamento antes de serem enviados. A transformação é feita de modo a gerar projeções no plano de forma a ter um intervalo de valores para x e y e permitir a transformação em coordenadas polares. As medidas também foram aproximadas para sua parte inteira, de forma a diminuir a instabilidade do sensor, que oscila muito devido à sua precisão. Dessa forma, os valores obtidos para o sensor serão equivalentes aos obtidos pelo *joystick*, permitindo que as mensagens sejam enviadas no mesmo formato e que a sua interpretação no Arduino seja equivalente.

5. Software - Arduino

A plataforma Arduino é bastante popular e vem sendo utilizado em diversos projetos de prototipagem ao redor do mundo. O Arduino UNO é o modelo da plataforma que compõe este projeto, e apesar de ser o primeiro da série de placas Arduino USB, oferece todas as funcionalidades necessárias ao objetivo final do trabalho, que como mencionado anteriormente, é controlar o aeromodelo simulando os antigos potenciômetros do controle com sinais PWM. Estes sinais são pulsos quadrados gerados em uma frequência determinada no Arduino que, filtrados, são sinais analógicos contínuos de tensão. O filtro utilizado é um filtro passa-baixas, conforme especificado no capítulo 3.

O PWM é gerado pelos *timers* ou contadores do microcontrolador. O Arduino UNO possui 3 *timers*: *timer* "0", "1" e "2", sendo o primeiro e o último de 8 *bits* e o segundo de 16 *bits*. Eles podem ser configurados para a geração de um sinal de PWM em até dois pinos cada, ou seja, no total é possível gerar até 6 sinais desse tipo. Esses *timers* serão chamados de agora em diante de CTx, que vem de Contador/Temporizador.

O CT0 não será utilizado diretamente no projeto. Isso significa que não vamos alterar suas configurações padrão e utilizá-lo para gerar PWM. O motivo é bem simples: este temporizador é o *default* do Arduino para funções específicas como delay(), milis() e micros() e, se modificado, alterará o funcionamento dessas funções, que também são utilizadas ao longo do código do projeto. Sendo assim, o CT1 e o CT2 farão o papel de gerar os sinais desejados. Antes, porém, é necessário compreender seu funcionamento e de que forma podem ser configurados para cumprir essa tarefa. O CT2 será apresentado primeiro por ser um *timer* de apenas 8 *bits*, sendo mais simples.

5.1. CT2: Contador/Temporizador 2

O CT2 é um contador de 8 *bits* em *hardware* que possibilita contagens de acordo com a frequência selecionada pelo usuário. Quando a contagem no registrador TCNT2 chega a seu valor máximo, algumas ações podem ser realizadas, assim como quando chega a zero ou coincide com valores armazenados previamente nos registradores de comparação, o OCR2A e o OCR2B. Algumas definições podem ser importantes para o entendimento:

- FUNDO: igual a zero;
- TOPO: pode ir até o máximo, que é 255 (ou 0xFF), ou é definido no OCR2A

Quando TCNT2 fica igual a OCR2A ou OCR2B, o gerador de forma de onda gera uma saída de acordo com o modo de operação (WGM22:0) e o modo de comparação (COM2A1:0 ou COM2B1:0).

O modo de operação selecionado para atender às necessidades do projeto foi o modo 3, *Fast* PWM (PWM rápido). Este modo difere do comum por trabalhar com rampa de alta frequência, que faz este modo adequado para aplicações DAC (*digital-analog* conversion ou conversão digital-analógica). Com relação ao modo de comparação, o não-invertido foi selecionado.

O *clock* utilizado foi o padrão, de 16 MHz, e o fator do pré-escalonador, que determina a frequência do PWM foi definido como N = 8, resultando em uma frequência de 7,81kHz. Esse ajuste na seleção do relógio é feito em CS22:0 [16].

5.2. CT1: Contador/Temporizador 1

O CT1 é um contador de 16 *bits* também em hardware que funciona de forma idêntica ao CT2, com a diferença de que possui três e não apenas dois registradores de comparação e que estes também tem 16 *bits*. A lógica de controle que define a operação a ser realizada também é a mesma e por isso dispensa comentários extras.

Sabe-se porém que o AVR é um processador de 8 *bits* e, portanto, os registradores de 16 *bits* são acessados em duas etapas de 8 *bits*. Como não há necessidade de fazer acessos simultâneos de 16 *bits* para este trabalho, o CT1 é configurado no mesmo modo que o CT2, ou seja, modo de operação *Fast* PWM.

Os *bits* de controle COM1A1:0 e COM1B1:0 tem exatamente as mesmas funções e são fixados da mesma maneira que para o TC2, ou seja, selecionando o modo não-invertido da geração do PWM.

O *clock* utilizado foi o padrão, de 16 MHz, e o fator do pré-escalonador foi definido como N = 8. Esse ajuste na seleção do relógio é feito em CS12:0 [17].

5.3. Demais configurações

A geração dos sinais de PWM se dão nas portas 3 e 11 para o TC2 (registradores OC2B e OC2A, respectivamente) e nas portas 9 e 10 para o TC1 (registradores OC1A e OC1B, respectivamente). Assim, é importante que essas portas sejam configuradas como portas de saída.

O valor máximo MAX é fixado em 160, para que a tensão gerada não ultrapasse 3,1Volts, evitando sobretensão e possíveis danos.

Algumas definições foram adotadas ao longo do código para facilitar o entendimento e serão também utilizadas neste texto com o mesmo objetivo. São elas:

- JEV (*Joystick* Esquerda Vertical) para registrador OCR2B;
- JEH (*Joystick* Esquerda Horizontal) para registrador OCR2A;
- JDV (*Joystick* Direita Vertical) para registrador OCR1B;
- JDH (*Joystick* Direita Horizontal) para registrador OCR1A;

Ademais, fica faltando a configuração da conexão serial *Bluetooth* com o Android, que é operada através do módulo *Bluetooth* HC-05.

5.3.1. Comunicação Serial no Arduino

A plataforma Arduino tem, em *hardware*, suporte para comunicação serial nos pinos 0 e 1 e seu suporte nativo acontece nas portas seriais do microcontrolador Atmega, as UARTs. No entanto, por se tratar de uma aplicação muito simples, foi utilizada a biblioteca própria do Arduino, a *SoftwareSerial*. Sua maior vantagem é a de permitir a comunicação serial em outros pinos digitais de entrada e saída (I/O) da plataforma, o que é feito replicando a funcionalidade das portas seriais via software, e permitindo múltiplas portas com velocidade de até 115200 bps [18].

Das configurações da função de inicialização do código:

- Foram selecionadas as portas 5, para transmissão (*bluetoothTx*), e 6, para recebimento (*bluetoothRx*);
- *begin(taxa de transmissão)*: a taxa de transmissão foi fixada em 9600 bps;
- Função sincroniza(), que será detalhada adiante;

5.3.2. Função sincroniza()

A função sincroniza() existe para realizar a sincronização entre o controle e o *drone*, ou seja, para que se reconheçam mutuamente e fixem um canal de comunicação entre si. A função não cria diretamente esse canal fazendo processo de autenticação de ID como é feito entre o Android e o módulo HC-05, pois isto é feito na programação da placa original, à qual não se tem acesso. Sendo assim, a função simplesmente simula o movimento que deveria ser feito no antigo potenciômetro que corresponde ao *joystick* da esquerda na vertical: causa uma variação de tensão do máximo ao mínimo por meio de um sinal PWM. É importante notar que essa variação precisa ser contínua, como uma função rampa no tempo.

A implementação da sincroniza() se deu por um *loop for* que incrementa o valor alocado em JEV por entre pequenos *delays* para que a mudança não seja brusca a ponto de ser vista como um impulso ao invés de uma rampa. Estando dentro da inicialização do código, a sincronização controle/*drone* acontece de forma automática.

5.4. Função *loop* principal

A função *loop* é, por definição, a parte do programa que se repete infinitamente e é onde se implementa tudo o que se deseja que o Arduino opere. Neste projeto, a função *loop* recebe as informações enviadas do Android por *Bluetooth*, processa essas mensagens, as interpreta e chama a função do *joystick* correspondente, que vai definir o valor de tensão a ser implementado nos PWMs.

A Figura 29 mostra o algoritmo utilizado para recebimento e processamento da mensagem que chega do Android.

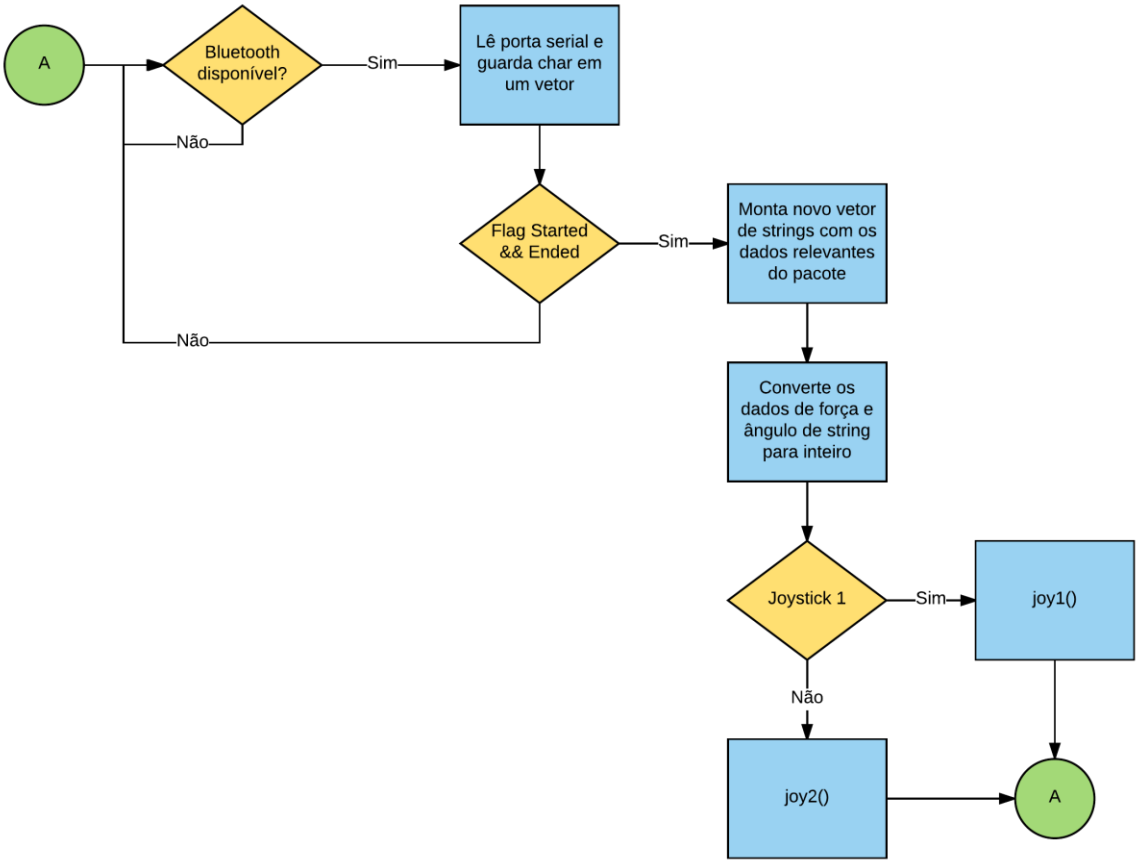

Figura 29 - Fluxograma do algoritmo do loop principal

Conforme apresentado no fluxograma, ao final do processamento do pacote de dados recebidos, é verificado se o pacote se refere ao *joystick* 1 (esquerda) ou ao *joystick* 2 (direita). Essa informação está inserida na mensagem montada pelo Android, como já mencionado anteriormente. Diferentes rotinas são chamadas a depender dessa informação: joy1() ou joy2(), que serão discutidas a seguir. Delas, o programa retorna ao início do algoritmo, indicado pelo ponto "A", de forma a obter um novo pacote de informações.

5.4.1. Rotinas joy1() e joy2()

Quando chamadas, as rotinas joy1() e joy2() recebem como parâmetros os valores de força e ângulo que vieram na mensagem do Android. Esses valores determinam os valores de tensão dos sinais PWM a serem gerados nos registradores JEV e JEH para joy1() e JDV e JDH para joy2().

É importante perceber que a precisão da tela do *smartphone* é baixa e causa grande instabilidade dos valores enviados, que mudam o tempo todo. Para amenizar essa instabilidade, decidiu-se por discretizar os valores que os ângulos e as forças podem assumir, criando-se zonas. Assim, a rotina verifica em qual zona os dados se encaixam e atualiza os valores de ângulo e força de acordo com ela antes de calcular o valor que será enviado ao registrador (entre 0 e 160). Este cálculo é uma simples regra de três, onde o valor máximo para a maioria dos *joysticks* é 160 e o valor mínimo é 0.

Uma outra observação a ser feita é a seguinte: valores de força e ângulo correspondem a parâmetros em coordenadas polares e, no entanto, os antigos potenciômetros respondiam a comandos na vertical e na horizontal, o que se modela de forma mais eficiente em coordenadas retangulares. Por esse motivo se torna mais coerente a obtenção das componentes cartesianas relativas aos valores de força e ângulo. Tal transformação também é calculada dentro das rotinas joy1() e joy2().

6. Ensaios e testes

Após a familiarização com a IDE e suas configurações, notou-se que simplesmente atribuir valores aos registradores para controlar o *drone*, apesar de proporcionar um resultado confiável e esperado, não era o suficiente para apresentar uma boa experiência para o usuário final da interface Android.

Logo após alguns ensaios, observaram-se alguns problemas relacionados à estabilidade que o controle, via aplicativo, aparentava possuir. Imprecisões causadas pelo excesso de informações enviadas ao Arduino e pela não tão alta performance do filtro utilizado descrito previamente também implicariam em uma sensibilidade do controle muito acima do esperado, o que acabaria por dificultar muito a pilotagem do VANT.

Tendo em vista estes problemas, sucederam algumas tentativas de otimização dos nossos processos, que serão detalhadas neste capítulo, e com seus respectivos resultados esperados e conseguidos em seguida.

6.1. Rotinas e estabilidade

Para sincronizar o *drone* com o restante do sistema, assim como no controle original de fábrica, é sempre necessária a realização de certa rotina de sincronização. Deste ponto surgiu a ideia de criar algumas rotinas para simplificar a navegação do *drone* e, se possível, também ajudar no controle de estabilidade do mesmo.

6.1.1. Rotina de sincronização

Para a criação desta, primeiramente foi analisado o funcionamento no sistema de controle original para que pudéssemos entender o padrão de funcionamento por trás da sincronização. Uma vez entendidos os eventos necessários para tal com suas respectivas ordens de ocorrência, desenvolver este comando será simples.

Após vários testes constatou-se que, para atingir nosso objetivo, era necessário ocorrer nesta ordem:

1. Ligar o *drone*;
2. Ligar a placa de controle mantendo a aceleração vertical em seu valor máximo;
3. Esperar a frequência em que as luzes do *drone* piscam diminuir pela metade (em cerca de 4 segundos);
4. Decrementar a aceleração para seu valor mínimo e esperar alguns segundos;

Entendidos os passos percebe-se que, para a parte de programação, basta nos atentarmos aos passos 2 e 4. A ideia é a criação de uma rampa de tensão com alguns *delays* para nos assegurarmos de que o valor se mantenha constante em seus respectivos máximos e mínimos quando necessário.

Assim como anteriormente, controlaremos a tensão de saída do arduíno através do registrador, mais precisamente do OCR2B, que regula o *joystick* esquerdo vertical, ou JEV. Para a da rampa de tensão, um loop é feito com a função *for()* e o auxílio de uma variável de contagem.

A figura a seguir retrata a função descrita acima:

```
void sincroniza(void){
  int valor = 0;
  JEV = 0;  //0V
  delay(3000);
  for (valor=0; valor<MAX; valor++){
    JEV = valor;
    delay(10);
  }
  delay(1500);
}
```

Figura 30 - Rotina de sincronização

Considerando que a automatização do projeto se restringe aos potenciômetros do controle, e não da placa inteira, tornou-se inviável a construção de uma rotina que realizasse os quatro passos. Atividades como ligar o *drone* e, em seguida o controle ainda devem ser, claramente, feitos à mão.

6.1.2. Controle de estabilidade

Os ensaios feitos acima serviram de inspiração e foram o início de um *brainstorming* para responder à pergunta: Que outras funções poderiam ser criadas de modo a simplificar a dirigibilidade do VANT?

A resposta preponderante seria resolver o excesso de informações provindos do programa para evitar a constante instabilidade observada dos controles, quando operados pelo aplicativo. A intenção é limitar o número de possibilidades de comando que o *drone* possa receber com a implementação de zonas de atividade.

Assim como explicado previamente, a informação captada pelo aplicativo nos fornece dados em coordenadas polares, ou seja, nos dão a magnitude e o ângulo presentes em cada um dos *joysticks*. Foi estabelecido, então, um critério de zonas, onde existirão onze possibilidades de intensidade e 8 de angulação. O valor do módulo iniciará em 0% e receberá incrementos de 10 em 10% até o seu valor máximo. Já o ângulo será separado de quarenta e cinco em quarenta e cinco graus, por exemplo: qualquer ângulo lido entre -22,5 e 22,5 graus será passado como 0; entre 22,5 e 67,5 graus será passado como 45 e assim sucessivamente. Deste modo a angulação terá direções assim como a rosa dos ventos. Caso observada a necessidade de aumento na quantidade de direções possíveis, esse valor pode ser reavaliado. Essa rotina será aplicada para ambos os analógicos.

```
// ROTINAS DO JOYSTICK 1
 void joy1(int angle, int power) {
  //Direita
  if (angle >= -22.5 && angle <= 22.5) {
    angle = 0;
  }

if (angle > 22.5 && angle <= 67.5) {
    angle = 45;
  }

  if (angle > 67.5 && angle <= 112.5) {
    angle = 90;
  }

if (angle > 112.5 && angle <= 157.5) {
    angle = 135;
  }

if (angle > 157.5 && angle <= 180) {
    angle = 180;
  }
```

```
  //Esquerda
  if (angle >= -67.5 && angle < -22.5) {
    angle = -45;
  }

if (angle >= -112.5 && angle < -67.5) {
    angle = -90;
  }

  if (angle >= -157.5 && angle < -112.5) {
    angle = -135;
  }

if (angle >= -180 && angle < -157.5) {
    angle = -180;
  }
```

Figura 31 - Rotina do Joystick 1

6.2. Interferências e dificuldades encontradas

No decorrer do projeto foram encontrados alguns desafios a serem superados. Estes estavam presentes tanto na parte mecânica quanto no software. Entre as dificuldades encontradas podem-se citar 3 principais responsáveis por grande partes dos teste. Essas são:

- Sincronia;
- Aspectos físicos do *drone*;
- Uso inapropriado dos registradores.

A sincronia do *drone*, já discutida previamente, tomou uma grande porção do tempo de teste. O kit controle/*drone* adquirido não veio com um manual de instruções, portanto, definir com acurácia o comportamento do mesmo requereu uma maior atenção. Definir os *delays* e as posições corretas dos analógicos foi uma atividade puramente experimental e arbitrária. Experimentos futuros podem determinar maneiras mais eficazes de realização do mesmo.

Após observar comportamentos aleatórios e inesperados, percebeu-se que o *drone* é extremamente sensível a perturbações. Uma pequena diferença no tamanho de uma de suas pernas faz com que ele apresente comportamentos imprevisíveis em se programando como, por exemplo, girar em torno do eixo da perna maior ou provocar instabilidades na hora de descolar do chão e acabando por alterar completamente a direção de viagem. Outro componente responsável por interferir em diversos testes é a câmera de vídeo acoplada ao VANT. O peso da mesma altera a posição do centro de massa do *drone*, consequentemente alterando seu ponto de equilíbrio, fazendo com que ele constantemente também se mova em direções não desejadas.

Na ideia inicial de projeto optou-se por usar a função *analogWrite()*, da biblioteca do Arduino, mas essa função atribui diretamente à saída o valor descrito na mesma. Isto conflitava com a informação por nós colocada nos registradores e acabou por tornar incerta a resposta que teríamos em cada situação. Essa contradição naturalmente faria com que testes apresentassem resultados diferentes do esperado. Para evitar este contratempo descartou-se o uso do *analogWrite()* e definiu-se os valores desejados de madeira direta, utilizando apenas os registradores para tal.

Um desafio à parte foi a familiarização com toda a programação necessária para o advento do projeto. Estes tópicos essenciais para o desenvolvimento do projeto, como programação em C, Java, utilização do Arduino e Android Studio, não fazem parte do fluxo de engenharia elétrica da Universidade de Brasília, e foram aprendidos de forma exclusivamente autônoma.

6.3. Ensaios

Feitas as devidas considerações, diversos testes foram realizados para avaliar o funcionamento do nosso sistema de controle. Dado que o projeto como um todo é composto por duas macroetapas, optou-se por checa-las uma a uma antes de tentar o sistema completo. Deste modo a detecção e correção e erros se daria mais fácil.

Criaram-se, então, duas situações, a Aplicativo-Arduíno, e a Arduíno-Drone que serão descritas a seguir.

Por dificuldades encontradas no projeto e tempo hábil necessário para solucionar as mesmas, a exibição dos testes Aplicativo – Drone não foram totalmente concluídas. Foram realizados testes não documentados onde o funcionamento se provou instável. A comunicação do sistema como um todo é um sucesso, mas a implementação de zonas e

fortes indícios de uma fuga de corrente no filtro estar causando uma grande perda de tensão nos resistores instalados não permitiram o funcionamento pleno do sistema. Otimizações a serem desenvolvidas devem focar nestas duas dúvidas, que quando sanadas, facilitarão a criação de novas funções.

6.3.1. Aplicativo – Arduíno

Nesta seção faremos um teste bem simples. Feito o pareamento *bluetooth* entre celular e módulo HC05, serão mandados alguns comandos, sendo dois deles de fácil identificação para nos assegurarmos de que a informação enviada e a recebida são, de fato, iguais.O primeiro e o último comandos dados são exatamente o centro dos *joysticks,* com 0º de ângulo e 0% de intensidade. Feito isto, outros comandos são enviados para testar outras posições.

A figura a seguir foi retirada do Monitor Serial do Arduíno. A IDE dá a possibilidade de imprimir as informações desejadas através do comando serial.Print. Desta forma, realizar debugs e otimizações do código ocorreu de forma mais rápida. Nota-se pela imagem que todos os quatro quadrantes do joystick operam de acordo com o esperado.

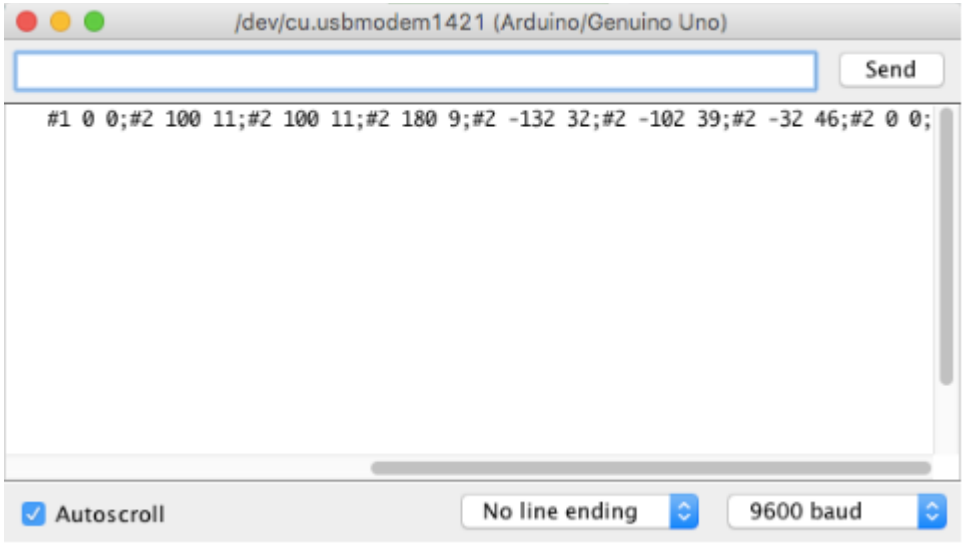

Figura 32 - Ensaio Monitor Serial

6.3.2. Arduíno – Drone

Serão ilustrados aqui dois testes. O primeiro envolvendo uma variação da vertical do joystick esquerdo, e o segundo a horizontal do direito. Assim como previamente discutido, os valores desejados para cada direção são escritos diretamente nos respectivos registradores do arduíno. Neste caso os registradores são o OCR2B e o OCR1A.

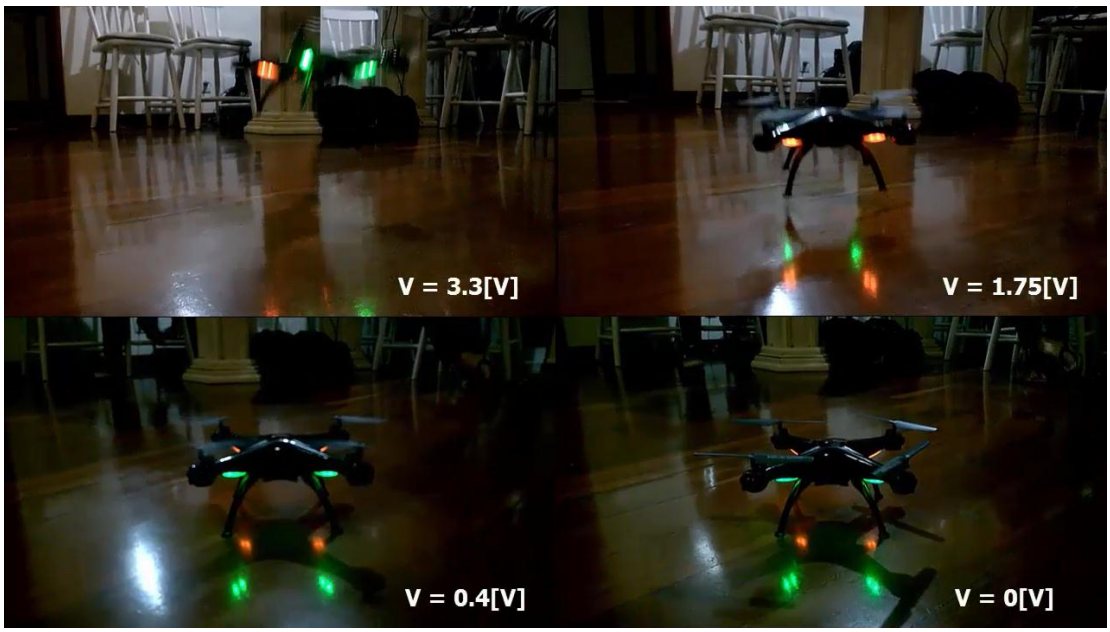

Figura 33 - Ensaio Joystick Esquerdo Vertical

Na figura acima podemos observar como diferentes tensões aplicadas aos motores interferem em suas respectivas velocidades de funcionamento, consequentemente alterando seu tempo de decolagem. Apesar de um pouco desfocado, nota-se pela altura do drone em relação ao chão que, quando submetida a uma tensão de 3.3V, a primeira imagem (superior esquerda) ascende mais rapidamente que as demais. Observou-se também que quando submetido a uma baixa tensão os motores são acionados, mas não com força o suficiente para que aja a decolagem (inferior esquerda).

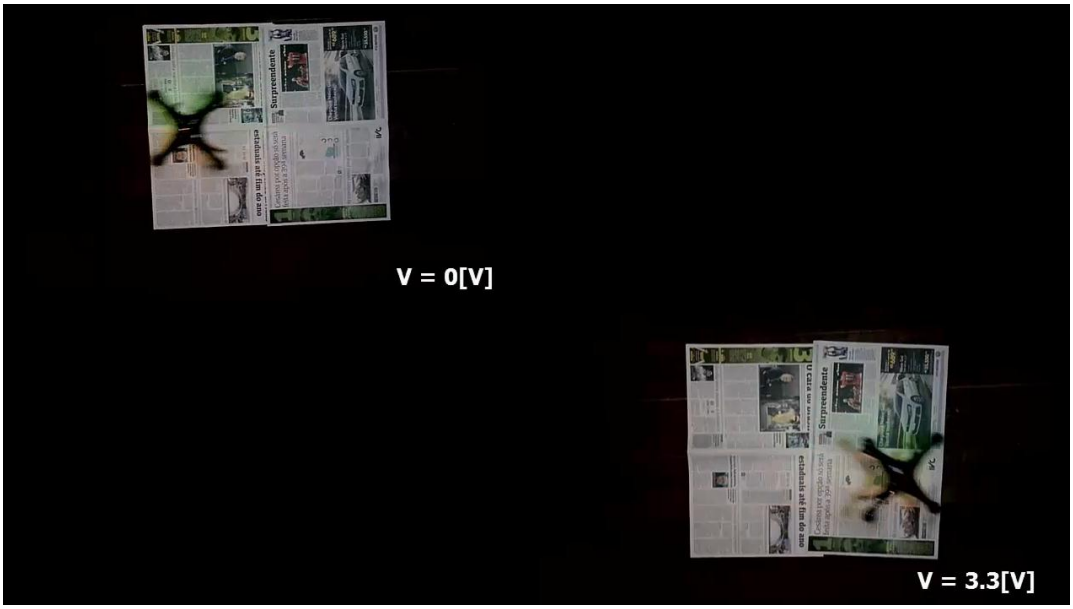

Figura 34 - Ensaio Joystick Direito Horizontal

Desta vez, de forma similar, podemos ver as respostas à duas diferentes tensões aplicadas à horizontal do joystick direito. Lembrando que neste e nos outros dois casos possíveis, a voltagem de equilíbrio a ser aplicada para que não existam alterações de posicionamento do drone é de 1.65V. Os extremos deste valor foram usados para que se tivessem as velocidades máximas em sentidos opostos.

6.4. Conclusão

O avanço da robótica, no que tange ao desenvolvimento tecnológico de robôs aéreos, é notável em todo o mundo e por isso o seu estudo e pesquisa na área dentro das universidades é imprescindível. Partindo-se deste ponto de vista, este trabalho quis dar início a uma série de projetos aplicáveis a esses aeromodelos tão atuais: os *drones* e VANTs. Mergulhados dentro de uma vasta quantidade de possibilidades de projeto, começamos por criar novas alternativas de controle, utilizando principalmente os *smartphones*, tão multifuncionais atualmente.

De fato, é desejável que todo projeto tenha em seu fim uma aplicação prática, que contribua em algum sentido para a sociedade. Por isso, estudos sobre um controle de estabilidade mais preciso, criação de rotas mais complexas auxiliadas por GPS, realização de tarefas simples de vigilância, detecção e prevenção de colisões são exemplos de projetos que dão continuidade ao trabalho presente. Além de promover o desenvolvimento e aumentar o interesse dos estudantes na área de eletrônica e mesmo de automação na Universidade de Brasília, tal como este mesmo projeto.

O Syma X5SW foi o modelo do mercado escolhido para implementação das modificações e novas funcionalidades do equipamento. Pilotado por um controle convencional em rádio frequência que acompanha o aeromodelo, o objetivo principal era flexibilizar seu controle viabilizando o envio de comandos de um dispositivo celular, tanto por *touchscreen* quanto pela movimentação do aparelho que é medida pelo giroscópio. A grande acessibilidade do sistema operacional Android foi um grande motivador dessa ideia, já que outras funcionalidades ainda podem ser adicionadas à aplicação de controle do mesmo.

O primeiro passo de implementação foi descobrir como de fato o controle se comunicava com o *drone* e de que forma o sinal analógico dos potenciômetros eram convertidos e transmitidos, de modo que pudéssemos gerá-los de outra maneira e substituí-los. Isso foi resolvido com o modo *Fast* PWM dos contadores do Arduino e alguns filtros passa-baixas, visto que este microcontrolador não opera como um conversor digital-analógico. Quatro sinais PWM foram gerados em dois diferentes contadores presentes no Arduino: o contador/temporizador 1 e o contador/temporizador 2, já que cada um poderia gerar até dois sinais. Além disso, foi preciso implementar uma lógica de programação para que o Arduino recebesse e processasse os pacotes de informação contendo os comandos a serem enviados ao *drone*.

Finalmente, a interface piloto/*drone* precisava ser reinventada, realizando uma nova comunicação sem fio *Bluetooth*. A programação no Android se deu na linguagem Java principalmente, pois a linguagem xml foi utilizada apenas na implementação do design da aplicação. Assim, a mensagem enviada ao Arduino é composta por três informações indispensáveis: o número do *joystick* sendo movimentado, o ângulo de desvio em relação à direção convencional e a força ou intensidade com que o *joystick* é movimentado, o que se reflete na velocidade do VANT. Com o objetivo de ser inicialmente e principalmente funcional, a aplicação conta com o suficiente em termos de design para que o usuário tivesse uma experiência satisfatória de controle, mas entendemos que é interessante que também a parte gráfica seja aprimorada futuramente, como a questão da sensibilidade do *joystick* e do sensor giroscópio.

Sem dúvida, foram obtidos resultados satisfatórios. A técnica de controle por sinais PWM combinados aos filtros RC é eficiente e pode continuar a ser utilizada em aplicações como a deste trabalho. O sistema operacional Android não deixou a desejar no desenvolvimento da interface com o usuário e parece ainda poder oferecer muito mais recursos para que a aplicação seja aprimorada e forneça uma experiência ainda melhor de imersão e controle a quem o utiliza. Entendemos que o projeto foi bem-sucedido e que

resultados concretos foram colhidos ao final deste tempo de trabalho: o *drone* responde aos sinais gerados pelo Arduino da forma como era esperada, e é possível controlá-lo razoavelmente bem pelo *smartphone*, tanto no modo de gestos na tela, ou seja, por toque, quanto pelo giroscópio do dispositivo, movimentando o celular.

Presenciando inúmeras dificuldades provindas do hardware e do software, percebemos que este progresso é delicado e trabalhoso. Não-linearidades do sistema como potenciômetros imperfeitos, instabilidades naturais da carcaça do *drone*, senbilidades altas aos ventos, bem como alta sensibilidade do dispositivo móvel são apenas alguns desafios a serem constantemente contornados e superados. Entretanto, tendo em vista o quão grande é o potencial deste sistema para atender a diversas necessidades que surgem no mundo de hoje, estes problemas se tornam pequenos e se refletem em motivação para dar continuidade ao projeto.

Com as novas ferramentas do modelo, esperamos que as aplicações e que a eficiência se ampliem em projetos futuros. Grande é o potencial que o sistema tem para atender a diversas necessidades que surgem no mundo de hoje e para se tornar cada vez mais inteligente e útil.

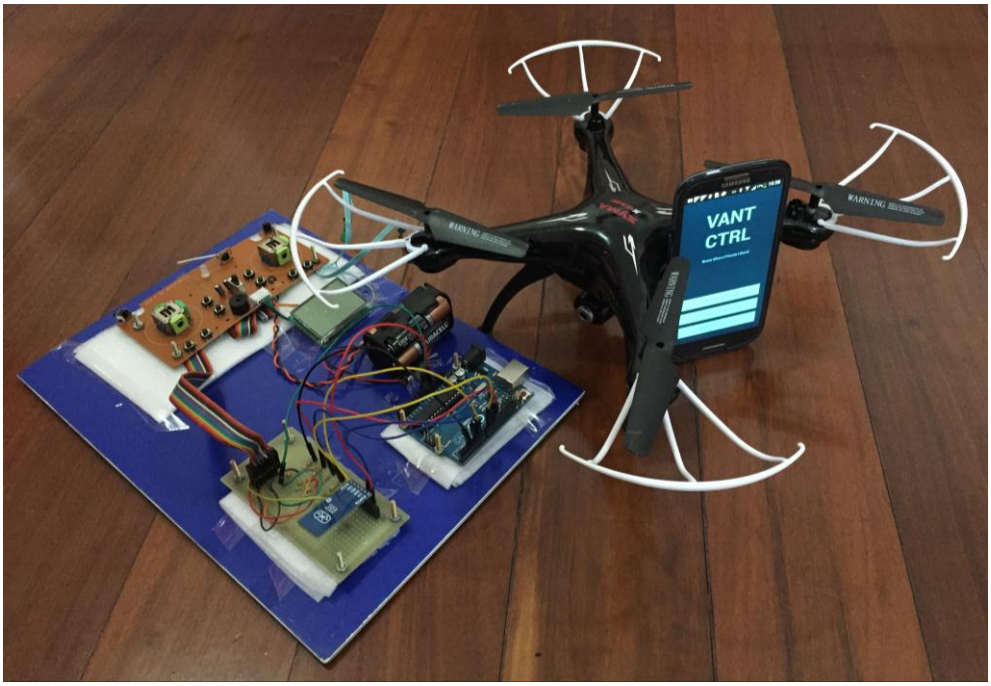

Figura 35 - Sistema final: plataforma de controle, aplicativo Android e o próprio VANT

Referências

[1] BARROS, P., BARROS, S. Historia de los Inventos: La Aeronáutica. *Revista Sucesos*, N° 16, p. 1-57. 14 jan 2012. Disponível em: <http://www.librosmaravillosos.com/> Acessado em: 25/05/2016.

[2] BECKER, M., SAMPAIO, R. C. B. Robôs Aéreos: a invasão já começou… *CIÊNCIAHOJE*. jan/fev, 2013. Vol. 50. Disponível em: <http://cienciahoje.uol.com.br/revista-ch/2013/300/pdf_aberto/robosaereos300.pdf > Acessado em: 25/05/2016.

[3] QUADCOPTER, DRONE FLYERS. Syma X5SW´FPV Real-Time WiFi Quadcopter First Look. Disponível em: <http://www.quadcopterflyers.com/2015/04/syma-x5sw-fpv-real-time-wifi-quadcopter.html> Acessado em: 02/06/2016.

[4] ARDUINO. Arduino UNO: Overview. Disponível em: <https://www.arduino.cc/en/Main/ArduinoBoardUno> Acessado em: 02/06/2016.

[5] CORDEIRO, F. Android Aprendiz: Um guia para iniciantes. *AndroidPro*. Disponível em: <http://www.brodowski.sp.gov.br/novo/wp-content/uploads/2016/04/Android_Aprendiz_Novo.pdf> Acessado em: 06/06/2016.

[6] WIKIPEDIA. Android Studio. *Wikipedia Enciclopédia Virtual*. Disponível em: <https://pt.wikipedia.org/wiki/Android_Studio> Acessado em: 06/06/2016.

[7] SANTOS, G. V. Como funciona um Potenciômetro. 27 maio 2013. Disponível em: <http://eletronicaemcasa.blogspot.com.br/2013/05/como-funciona-um-potenciometro.html> Acessado em: 06/06/2016.

[8] IRWIN, J. D., NELMS, R. M. *Análise Básia de Circuitos para Engenharia*. 2010. P. 21. Tradução: Rio de Janeiro: LTC. Traduzido da 9ª ed.

[9] ARDUINO. Arduino UNO: PWN. Disponível em: <https://www.arduino.cc/en/Main/Tutorial/PWN> Acessado em: 02/06/2016.

[10] HC Serial Bluetooth Products. *User Instructional Manual*. Disponível em: <http://www.rcscomponents.kiev.ua/datasheets/hc_hc-05-user-instructions-bluetooth.pdf.> Acesso em: 20 Junho 2016.

[11] ANDROID DEVELOPERS. Activities. Disponível em: <https://developer.android.com/guide/components/activities.html> Acesso em: 20 Junho 2016.

[12] ANDROID DEVELOPERS. Bluetooth. Disponível em: <https://developer.android.com/guide/topics/connectivity/bluetooth.html > Acesso em: 20 Junho 2016.

[13] ZEROKOL. An Android Joystick View. 13 out 2015. Disponível em: <https://github.com/zerokol/JoystickView> Acesso em: 20 Junho 2016.

[14] ANDROID CODE. JoyStick Controller. Disponível em: <http://www.akexorcist.com/2012/10/android-code-joystick-controller.html> Acesso em: 20 Junho 2016.

[15] Microsoft. Device Orientation and Motion Events. Disponível em: <https://developer.microsoft.com/en-us/microsoft-edge/platform/documentation/dev-guide/device/device-orientation-and-motion-events/> Acesso em: 20 Junho 2016.

[16] ZELENOVSKY, R. Apostila Temporizador/Contador 2 (TC2).

[17] ZELENOVSKY, R. Apostila Temporizador/Contador 1, 3, 4 e 5 (TC1, TC3, TC4 e TC5).

[18] ARDUINO. Arduino UNO: PWN. Disponível em: <https://www.arduino.cc/en/Reference/SoftwareSerial> Acessado em: 21/06/2016.

APÊNDICES

A. Android

A.1 activity_main_screen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"


android:paddingBottom="@dimen/activity_vertical_margin"

android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"

android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"

tools:context="app.com.example.android.vantctrl.MainScreen"
    android:background="#172D3A">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#859CA4"
        android:text="VANT CTRL"
        android:textSize="70sp"
        android:textStyle="bold"
        android:gravity="center"
        android:layout_weight="1"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bruno Silva e Priscila Libardi"
        android:textColor="#859CA4"
        android:layout_gravity="center"
        android:layout_weight="3"/>

    <Button
        android:layout_width="310dp"
        android:layout_height="wrap_content"
        android:id="@+id/botao1_porToque"
        android:text="Controle por Toque"
        android:textAllCaps="true"
        android:textColor="#172D3A"
        android:background="#859CA4"
        android:layout_gravity="center"
        android:layout_marginBottom="16dp"
        android:onClick="porToque"/>


    <Button
        android:layout_width="310dp"
        android:layout_height="wrap_content"
        android:id="@+id/botao2_porAcel"
        android:text="Controle por Giroscópio"
        android:textAllCaps="true"
        android:textColor="#172D3A"
        android:background="#859CA4"
        android:layout_gravity="center"
        android:layout_marginBottom="16dp"
        android:onClick="porGyro"/>

    <Button
        android:layout_width="310dp"
        android:layout_height="wrap_content"
        android:id="@+id/botao3_configBT"
        android:text="Configurar Bluetooth"
        android:textAllCaps="true"
        android:textColor="#172D3A"
        android:background="#859CA4"
        android:layout_gravity="center"
        android:layout_marginBottom="16dp"
        android:onClick="configBT"/>


</LinearLayout>
```

A.2. MainScreen.java

```
package app.com.example.android.vantctrl;

import android.content.Intent;
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainScreen extends
AppCompatActivity {

    public static Intent intent = null;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Configura o layout correspondente
(activity_main_screen)

setContentView(R.layout.activity_main_scree
n);
    }

    @Override
    public void onResume() {
        super.onResume();

        // Pega o endereço MAC da
DeviceListActivity via intent
        this.intent = getIntent();
    }

    public void porToque(View view) {

        // Inicia a activity porToque
        Intent intent = new Intent(this,
porToque.class);
        startActivity(intent);
    }

    public void porGyro(View view) {

        // Inicia a activity porGyro
        Intent intent = new Intent(this,
porGyro.class);
        startActivity(intent);
    }

    public void configBT(View view) {

        // Inicia a activity DeviceListActivity
        Intent intent = new Intent(this,
DeviceListActivity.class);
        startActivity(intent);
    }
}
```

A.3. activity_device_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"

android:paddingBottom="@dimen/activity_vertical_margin"

android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"

android:paddingTop="@dimen/activity_vertical_margin"
   android:background="#172D3A"

tools:context="app.com.example.android.vantctrl.DeviceListActivity">

      <TextView
android:id="@+id/title_paired_devices"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"

android:textAppearance="?android:attr/textAppearanceLarge"
         android:text="Selecione HC-05 na lista de pareados:"
         android:visibility="gone"
         android:background="#172D3A"
         android:textColor="#859CA4"
         android:paddingLeft="5dp"
         />

      <ListView
android:id="@+id/paired_devices"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:stackFromBottom="false"
         android:layout_weight="1"
         />

      <TextView
         android:id="@+id/connecting"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#859CA4"
android:textAppearance="?android:attr/textAppearanceLarge" />

      <TextView
         android:id="@+id/infoText"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#859CA4"
         android:text="Se não há dispositivos listados, parear na configurações principais."

android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>
```

A.4. device_name.xml

```
<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:padding="5dp"
    android:textColor="#859CA4">

</TextView>
```

A.5. DeviceListActivity.java

```
package app.com.example.android.vantctrl;

import
android.support.v7.app.AppCompatActivity;
import java.util.Set;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import
android.widget.AdapterView.OnItemClickListener;

public class DeviceListActivity extends
AppCompatActivity {

    // Debugging
    private static final String TAG =
"DeviceListActivity";
    private static final boolean D = true;


    // textview para status de conexão
    TextView textView1;

    // String EXTRA para enviar endereço para
a MainScreen
    public static String
EXTRA_DEVICE_ADDRESS =
"device_address";

    // Member fields
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String>
mPairedDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);

setContentView(R.layout.activity_device_list);
    }

    @Override
    public void onResume()
    {
        super.onResume();
        //***************
        checkBTState();

        textView1 = (TextView)
findViewById(R.id.connecting);
        textView1.setTextSize(40);
        textView1.setText(" ");

        // Inicializa array adapter para os
dispositivos pareados
        mPairedDevicesArrayAdapter = new
ArrayAdapter<String>(this,
R.layout.device_name);

        // Encontra e configura a ListView para
os dispositivos pareados
        ListView pairedListView = (ListView)
findViewById(R.id.paired_devices);

pairedListView.setAdapter(mPairedDevicesAr
rayAdapter);

pairedListView.setOnItemClickListener(mDev
iceClickListener);

        // Pega o Bluetooth Adapter local
        mBtAdapter =
BluetoothAdapter.getDefaultAdapter();

        // Pega os dispositivos pareados e junta
em pairedDevices
        Set<BluetoothDevice> pairedDevices =
mBtAdapter.getBondedDevices();

        // Adiciona dispositivos pareados
anteriormente ao array
        if (pairedDevices.size() > 0) {

findViewById(R.id.title_paired_devices).setVi
sibility(View.VISIBLE); //torna o titulo visivel
            for (BluetoothDevice device :
pairedDevices) {
```

```
mPairedDevicesArrayAdapter.add(device.getN
ame() + "\n" + device.getAddress());
            }
        } else {
            String noDevices =
getResources().getText(R.string.none_paired).
toString();

mPairedDevicesArrayAdapter.add(noDevices)
;
        }
    }

    // Configura o on-click listener para a lista
    private OnItemClickListener
mDeviceClickListener = new
OnItemClickListener() {
        public void
onItemClick(AdapterView<?> av, View v, int
arg2, long arg3) {

            textView1.setText("Conectando...");
            // Pega o MAC address do dispositivo,
que sao os ultimos 17 char da View
            String info = ((TextView)
v).getText().toString();
            String address =
info.substring(info.length() - 17);


            // Dispara uma intent pra chamar uma
nova activity e pega um extra que é o MAC
address.
            Intent i = new
Intent(DeviceListActivity.this,
MainScreen.class);

i.putExtra(EXTRA_DEVICE_ADDRESS,
address);
            startActivity(i);
        }
    };

    private void checkBTState() {
        // Checa se o aparelho tem bluetooth e se
está ligado

mBtAdapter=BluetoothAdapter.getDefaultAda
pter();
        if(mBtAdapter==null) {
            Toast.makeText(getBaseContext(),
"Device does not support Bluetooth",
Toast.LENGTH_SHORT).show();
        } else {
            if (mBtAdapter.isEnabled()) {
                Log.d(TAG, "...Bluetooth ON...");
            } else {
                // Pede permissao pra ligar o
bluetooth
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST
_ENABLE);

startActivityForResult(enableBtIntent, 1);

            }
        }
    }
}
```

A.6. activity_por_toque.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"

android:paddingBottom="@dimen/activity_vertical_margin"

android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"

android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".porGyro"
   android:background="#172D3A">

   <LinearLayout
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:orientation="vertical">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#FFF"
      android:id="@+id/angleTextView1"/>


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#FFF"
      android:id="@+id/powerTextView1"/>


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#FFF"
      android:id="@+id/directionTextView1"/>

   <com.zerokol.views.JoystickView
      android:layout_width="200dp"
      android:layout_height="200dp"
      android:id="@+id/joystickView1" />

   </LinearLayout>

   <LinearLayout
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:layout_alignParentRight="true">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#FFF"
      android:id="@+id/angleTextView2"/>


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#FFF"
      android:id="@+id/powerTextView2"/>


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textColor="#FFF"
      android:id="@+id/directionTextView2"/>

   <com.zerokol.views.JoystickView
      android:layout_width="200dp"
      android:layout_height="200dp"
      android:id="@+id/joystickView2" />

   </LinearLayout>


</RelativeLayout>
```

A.7. porToque.java

```
package app.com.example.android.vantctrl;

import com.zerokol.views.JoystickView;
import
com.zerokol.views.JoystickView.OnJoystickM
oveListener;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
import android.os.Handler;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;


public class porToque extends Activity {

    Handler bluetoothIn;

    final int handlerState = 0;
//used to identify handler message
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;

    private static final UUID
BTMODULEUUID =
UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");

    private Map<JoystickView, JoystickAtt>
joysticks = new HashMap<JoystickView,
JoystickAtt>(); //cria-se mapa de joysticks
    private boolean BT_IS_CONNECTED;
    private String address;

    private ConnectedThread
mConnectedThread;

    private int[] count = new int[]{0,0};

    @Override
    public void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);

setContentView(R.layout.activity_por_toque);

        btAdapter =
BluetoothAdapter.getDefaultAdapter();       //
pega Bluetooth adapter

        checkBTState();


        for (int i = 1; ; i++) {

            final int joyNumber = i;
            final JoystickAtt att = new
JoystickAtt();
            final JoystickView joystick;

            int idJ =
this.getResources().getIdentifier("joystickView
" + joyNumber, "id", getPackageName());
            joystick = (JoystickView)
findViewById(idJ);

            if (idJ == 0) break;

            int idA =
this.getResources().getIdentifier("angleTextVi
ew" + joyNumber, "id", getPackageName());
            att.setAngleTextView((TextView)
findViewById(idA));
            int idP =
this.getResources().getIdentifier("powerTextVi
ew" + joyNumber, "id", getPackageName());
            att.setPowerTextView((TextView)
findViewById(idP));
            final int[] idD =
{this.getResources().getIdentifier("directionTe
xtView" + joyNumber, "id",
getPackageName())};
            att.setDirectionTextView((TextView)
findViewById(idD[0]));

            joysticks.put(joystick, att);
            final String[] dir = {" "};


            // Listener de eventos que sempre
retornará com a variação do angulo em
            // graus, a força do movimento em
porcentagem e a direção do movimento

            //ATENÇÃO: RIGHT E LEFT ESTÃO
TROCADOS NA DEFINIÇÃO DA CLASSE,
POR ISSO RIGHT TORNA-SE LEFT E LEFT,
RIGHT

joystick.setOnJoystickMoveListener(new
OnJoystickMoveListener() {
                @Override
```

```
            public void onValueChanged(int
angle, int power, int direction) {
                att.getAngleTextView().setText("
" + String.valueOf(angle) + "°");
                att.getPowerTextView().setText("
" + String.valueOf(power) + "%");
                switch (direction) {
                   case JoystickView.FRONT:
                      dir[0] = "F";

att.getDirectionTextView().setText(R.string.fr
ont_lab);
                      break;

                   case
JoystickView.FRONT_RIGHT:

att.getDirectionTextView().setText(R.string.fr
ont_right_lab);
                      dir[0] = "FL";
                      break;

                   case JoystickView.RIGHT:

att.getDirectionTextView().setText(R.string.ri
ght_lab);
                      dir[0] = "L";
                      break;

                   case
JoystickView.RIGHT_BOTTOM:

att.getDirectionTextView().setText(R.string.ri
ght_bottom_lab);
                      dir[0] = "BL";
                      break;

                   case JoystickView.BOTTOM:

att.getDirectionTextView().setText(R.string.bo
ttom_lab);
                      dir[0] = "B";
                      break;

                   case
JoystickView.BOTTOM_LEFT:

att.getDirectionTextView().setText(R.string.bo
ttom_left_lab);
                      dir[0] = "BR";
                      break;

                   case JoystickView.LEFT:

att.getDirectionTextView().setText(R.string.lef
t_lab);
                      dir[0] = "R";
                      break;

                   case
JoystickView.LEFT_FRONT:

att.getDirectionTextView().setText(R.string.lef
t_front_lab);
                      dir[0] = "FR";
                      break;

                   default:

att.getDirectionTextView().setText(R.string.ce
nter_lab);
                      dir[0] = "C";
                }

                String message = "#" +
joyNumber + " " + String.valueOf(angle)
                      + " " +
String.valueOf(power)
                      + ";";

                //count[joyNumber-1]++; isso
aqui é pra descartar as 5 primeiras leituras,
pra enviar menos mensagens
                //if ((count[joyNumber-1] % 5)
== 0)

mConnectedThread.write(message);
             }
          },
JoystickView.DEFAULT_LOOP_INTERVAL);
       }
    }

    // seta os atributos de um joystick
    public class JoystickAtt {

       private TextView angleTextView;
       private TextView powerTextView;
       private TextView directionTextView;

       public TextView getAngleTextView() {
          return angleTextView;
       }

       public void setAngleTextView(TextView
angleTextView) {
          this.angleTextView = angleTextView;
       }

       public TextView getDirectionTextView()
{
          return directionTextView;
       }
```

```
        public void
setDirectionTextView(TextView
directionTextView) {
            this.directionTextView =
directionTextView;
        }

        public TextView getPowerTextView() {
            return powerTextView;
        }

        public void setPowerTextView(TextView
powerTextView) {
            this.powerTextView =
powerTextView;
        }
    }

    @Override
    public void onResume() {
        super.onResume();

        // Pega o MAC address da MainScreen
via intent
        Intent intent = MainScreen.intent;

        address =
intent.getStringExtra(DeviceListActivity.EXT
RA_DEVICE_ADDRESS);

        // cria o dispositivo bluetooth e seta o
mac address
        BluetoothDevice device =
btAdapter.getRemoteDevice(address);

        try {
            btSocket =
createBluetoothSocket(device);
        } catch (IOException e) {
            Toast.makeText(getBaseContext(),
"Socket creation failed",
Toast.LENGTH_LONG).show();
        }
        // Estabelece a conexão do socket
        try
        {
            btSocket.connect();
        } catch (IOException e) {
            try
            {
                btSocket.close();
            } catch (IOException e2)
            {

            }
        }
        mConnectedThread = new
ConnectedThread(btSocket);
        mConnectedThread.start();
        //mConnectedThread.write("HABEMUS
CONEXÃO VIA BLUETOOTH!!!!");
    }

    private BluetoothSocket
createBluetoothSocket(BluetoothDevice
device) throws IOException {

        return
device.createRfcommSocketToServiceRecord(
BTMODULEUUID);
        //creates secure outgoing connecetion
with BT device using UUID
    }

    private void checkBTState() {

        if(btAdapter==null) {
            Toast.makeText(getBaseContext(),
"Device does not support bluetooth",
Toast.LENGTH_LONG).show();
        } else {
            if (btAdapter.isEnabled()) {
            } else {
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST
_ENABLE);

startActivityForResult(enableBtIntent, 1);
            }
        }
    }

    //create new class for connect thread
    private class ConnectedThread extends
Thread {
        private final InputStream mmInStream;
        private final OutputStream
mmOutStream;

        //creation of the connect thread
        public ConnectedThread(BluetoothSocket
socket) {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            try {
                //Create I/O streams for connection
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }
```

```
        public void run() {
            byte[] buffer = new byte[256];
            int bytes;

            // Keep looping to listen for received
messages
            while (true) {
                try {
                    bytes = mmInStream.read(buffer);
//read bytes from input buffer
                    String readMessage = new
String(buffer, 0, bytes);
                    // Send the obtained bytes to the
UI Activity via handler

bluetoothIn.obtainMessage(handlerState,
bytes, -1, readMessage).sendToTarget();
                } catch (IOException e) {
                    break;
                }
            }
        }
        //write method
        public void write(String input) {
            byte[] msgBuffer = input.getBytes();
//converts entered String into bytes
            try {
                mmOutStream.write(msgBuffer);
//write bytes over BT connection via outstream
            } catch (IOException e) {
                //if you cannot write, close the
application
                Toast.makeText(getBaseContext(),
"Connection Failure",
Toast.LENGTH_LONG).show();
                finish();
            }
        }
    }
}
```

A.8. activity_por_gyro.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:background="#172D3A">


   <LinearLayout
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"

android:paddingBottom="@dimen/activity_vertical_margin"

android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"

android:paddingTop="@dimen/activity_vertical_margin"
      tools:context=".porGyro"
      android:background="#172D3A"
      android:orientation="vertical">

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#FFF"

android:id="@+id/angleTextViewAcel"/>


      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#FFF"

android:id="@+id/powerTextViewAcel"/>


      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#FFF"

android:id="@+id/directionTextViewAcel"/>

      <com.zerokol.views.JoystickView
         android:layout_width="200dp"
         android:layout_height="200dp"
         android:id="@+id/joystickViewAcel"
         android:layout_marginLeft="52dp"
         android:layout_marginStart="52dp"/>

   </LinearLayout>


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_vertical"
      android:layout_weight="1"
      android:layout_marginRight="100dp"
      android:layout_marginLeft="100dp"
      android:id="@+id/tv"
      android:textColor="#FFF"/>


</LinearLayout>
```

A.9. porGyro.java

```
package app.com.example.android.vantctrl;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Handler;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

import
com.zerokol.views.JoystickView.OnJoystickM
oveListener;
import com.zerokol.views.JoystickView;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.text.DecimalFormat;
import java.util.UUID;

import java.lang.Math;

public class porGyro extends Activity
implements SensorEventListener {
    //Variaveis do Joystick
    private TextView angleTextViewAcel;
    private TextView powerTextViewAcel;
    private TextView directionTextViewAcel;
    private JoystickView joystickAcel;

    //Variaveis do giroscopio
    private TextView tv;
    private SensorManager sManager;

    //Variaveis do Bluetooth
    Handler bluetoothIn;
    final int handlerState = 0;
//used to identify handler message
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;

    private static final UUID
BTMODULEUUID =
UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");
    ConnectedThread mConnectedThread;
    private String address;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {

        super.onCreate(savedInstanceState);

setContentView(R.layout.activity_por_gyro);

        btAdapter =
BluetoothAdapter.getDefaultAdapter();       //
get Bluetooth adapter
        checkBTState();

        //Cria Joystick e textviews de info
        angleTextViewAcel = (TextView)
findViewById(R.id.angleTextViewAcel);
        powerTextViewAcel = (TextView)
findViewById(R.id.powerTextViewAcel);
        directionTextViewAcel = (TextView)
findViewById(R.id.directionTextViewAcel);
        joystickAcel = (JoystickView)
findViewById(R.id.joystickViewAcel);


joystickAcel.setOnJoystickMoveListener(new
OnJoystickMoveListener() {
            @Override
            public void onValueChanged(int angle,
int power, int direction) {
                angleTextViewAcel.setText(" " +
String.valueOf(angle) + "°");
                powerTextViewAcel.setText(" " +
String.valueOf(power) + "%");
                switch (direction) {
                    case JoystickView.FRONT:

directionTextViewAcel.setText(R.string.front_
lab);
                        break;

                    case
JoystickView.FRONT_RIGHT:

directionTextViewAcel.setText(R.string.front_
right_lab);
                        break;

                    case JoystickView.RIGHT:

directionTextViewAcel.setText(R.string.right_
lab);
                        break;

                    case
JoystickView.RIGHT_BOTTOM:

directionTextViewAcel.setText(R.string.right_
```

```
bottom_lab);
                    break;

                case JoystickView.BOTTOM:

directionTextViewAcel.setText(R.string.botto
m_lab);
                    break;

                case
JoystickView.BOTTOM_LEFT:

directionTextViewAcel.setText(R.string.botto
m_left_lab);
                    break;

                case JoystickView.LEFT:

directionTextViewAcel.setText(R.string.left_l
ab);
                    break;

                case JoystickView.LEFT_FRONT:

directionTextViewAcel.setText(R.string.left_fr
ont_lab);
                    break;

                default:

directionTextViewAcel.setText(R.string.center
_lab);
            }
            String message = "#1" + " " +
String.valueOf(angle)
                    + " " + String.valueOf(power)
                    + ";";
            mConnectedThread.write(message);
        }
    },
JoystickView.DEFAULT_LOOP_INTERVAL);


    //Cria textview para as info do giroscopio
e manager do sensor
    tv = (TextView) findViewById(R.id.tv);
    sManager = (SensorManager)
getSystemService(SENSOR_SERVICE);
  }

  //Para o giroscopio
  @Override
  protected void onResume()
  {
    super.onResume();
    //registra o listener para o sensor do
giroscopio, velocidade setada como a normal
    sManager.registerListener(this,
sManager.getDefaultSensor(Sensor.TYPE_ORI
ENTATION),SensorManager.SENSOR_DELA
Y_NORMAL);

    //Bluetooth
    //Get MAC address from
DeviceListActivity via intent
    Intent intent = MainScreen.intent;

    //Get the MAC address from the
DeviceListActivty via EXTRA
    address =
intent.getStringExtra(DeviceListActivity.EXT
RA_DEVICE_ADDRESS);

    //create device and set the MAC address
    BluetoothDevice device =
btAdapter.getRemoteDevice(address);

    try {
      btSocket =
createBluetoothSocket(device);
    } catch (IOException e) {
      Toast.makeText(getBaseContext(),
"Socket creation failed",
Toast.LENGTH_LONG).show();
    }
    // Establish the Bluetooth socket
connection.
    try
    {
      btSocket.connect();
    } catch (IOException e) {
      try
      {
        btSocket.close();
      } catch (IOException e2)
      {

      }
    }
    mConnectedThread = new
ConnectedThread(btSocket);
    mConnectedThread.start();
    //I send a character when
resuming.beginning transmission to check
device is connected
    //If it is not an exception will be thrown in
the write method and finish() will be called
    //mConnectedThread.write("HABEMUS
CONEXÃO VIA BLUETOOTH!!!!");

  }

  //Quando a atividade não é mais visível
  @Override
```

```
    protected void onStop()
    {
        //unregister the sensor listener
        sManager.unregisterListener(this);
        super.onStop();
    }

    @Override
    public void onAccuracyChanged(Sensor
arg0, int arg1)
    {
        //nada
    }

    @Override
    public void onSensorChanged(SensorEvent
event)
    {
        //retornar vazio se o status do sensor é
unreliable
        if (event.accuracy ==
SensorManager.SENSOR_STATUS_UNRELIA
BLE)
        {
            return;
        }

        //else it will output the Roll, Pitch and
Yawn values
        tv.setText("Orientation X (Roll) :" +
Float.toString(event.values[2]) + "\n" +
                "Orientation Y (Pitch) :" +
Float.toString(event.values[1]) + "\n" +
                "Orientation Z (Yaw) :" +
Float.toString(event.values[0]));

        transformAndSend(event.values[2],
event.values[1]);

    }

    private void transformAndSend (float roll,
float pitch) {
        double angle = 0;
        double power = 0;

        float x = roll/90;
        float y = pitch/90;

        angle = Math.atan(y/x);
        power = Math.sqrt(Math.pow(x, 2) +
Math.pow(y, 2));

        DecimalFormat df = new
DecimalFormat("###,##0");

        //converte de rad pra graus
        angle = angle*180/Math.PI;
        //transforma em porcentagem
        power = power*100;

        //consertando o ângulo a ser enviado de
acordo com o quadrante
        //no primeiro e no ultimo quadrante, o
angulo não sofre alterações
        if (x>0 && y<0)
            angle = 180 + angle;
        else if (x>0 && y>0)
            angle = -180 + angle;

        String message = "#2" + " " +
String.valueOf(df.format(angle))
                + " " +
String.valueOf(df.format(power))
                + ";";
        mConnectedThread.write(message);
    }

    private BluetoothSocket
createBluetoothSocket(BluetoothDevice
device) throws IOException {

        return
device.createRfcommSocketToServiceRecord(
BTMODULEUUID);
        //creates secure outgoing connecetion
with BT device using UUID
    }

    private void checkBTState() {

        if(btAdapter==null) {
            Toast.makeText(getBaseContext(),
"Device does not support bluetooth",
Toast.LENGTH_LONG).show();
        } else {
            if (btAdapter.isEnabled()) {
            } else {
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST
_ENABLE);

startActivityForResult(enableBtIntent, 1);
            }
        }
    }

    //create new class for connect thread
    private class ConnectedThread extends
Thread {
        private final InputStream mmInStream;
        private final OutputStream
mmOutStream;
```

```
        //creation of the connect thread
        public ConnectedThread(BluetoothSocket              }
socket) {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            try {
                //Create I/O streams for connection
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }


        public void run() {
            byte[] buffer = new byte[256];
            int bytes;

            // Keep looping to listen for received
messages
            while (true) {
                try {
                    bytes = mmInStream.read(buffer);
//read bytes from input buffer
                    String readMessage = new
String(buffer, 0, bytes);
                    // Send the obtained bytes to the
UI Activity via handler

bluetoothIn.obtainMessage(handlerState,
bytes, -1, readMessage).sendToTarget();
                } catch (IOException e) {
                    break;
                }
            }
        }
        //write method
        public void write(String input) {
            byte[] msgBuffer = input.getBytes();
//converts entered String into bytes
            try {
                mmOutStream.write(msgBuffer);
//write bytes over BT connection via outstream
            } catch (IOException e) {
                //if you cannot write, close the
application
                Toast.makeText(getBaseContext(),
"Connection Failure",
Toast.LENGTH_LONG).show();
                finish();
            }
        }
    }
```

B. Arduino

B.1. ctrl_jun21.ino

```
//P02

#include <SoftwareSerial.h>

#define MAX 160 //Valor máximo para não ultrapassar 3,1V
#define JEV OCR2B
#define JEH OCR2A
#define JDV OCR1B
#define JDH OCR1A


int bluetoothTx = 5;
int bluetoothRx = 6;
SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);
char inData[80];
byte index;
bool started = false;
bool ended = false;

void setup() {

Serial.begin(9600);
  pinMode(3, OUTPUT);   //OC2B
  pinMode(9, OUTPUT);   //OC1A
  pinMode(10, OUTPUT);  //OC1B
  pinMode(11, OUTPUT);  //OC2A

  // Timer 2 - Joystick esquerda
  TCCR2A = (1<<COM2A1)| (1<<COM2B1)| (1<<WGM21)| (1<<WGM20);  //Fast PWM Modo 3
  TCCR2B = (1<<CS21);   //clk/8
  //OCR2A = MAX/2;  //  0 (0V) Esq e Dir 160 (3,2V) - tem retenção
  //OCR2B = 0;     //  160 (0V) Bx e Alto 0 (3,2V) - livre

  //Timer 1 - Joystick direita
  TCCR1A = (1<<COM1A1)| (1<<COM1B1)| (1<<WGM10);  //Fast PWM Modo 5 (8 bits)
  TCCR1B = (1<<WGM12)| (1<<CS11);  //clk/8
  //OCR1AL = MAX/2;  //  0 (0V) Esq e Dir 160 (3,2V) - tem retenção
  //OCR1BL = MAX/2;  //  0 (0V) Bx e Alto 160 (3,2V) - tem retenção

 //Setup Bluetooth serial connection to android
  bluetooth.begin(115200);
  bluetooth.print("$$$");
  delay(100);
  bluetooth.println("U,9600,N");
  bluetooth.begin(9600);

  sincroniza();

}

void sincroniza(void){
  int valor = 0;
  JEV = 0;  //0V
  delay(3000);
  for (valor=0; valor<MAX; valor++){
    JEV = valor;
    delay(10);
  }
  delay(1500);
}

// ROTINAS DO JOYSTICK 1
 void joy1(int angle, int power) {

  if (angle >= -20 && angle <= 20) {
    angle = 0;
  }

if (angle > 20 && angle <= 40) {
    angle = 30;
  }

  if (angle > 40 && angle <= 60) {
    angle = 50;
  }

if (angle > 60 && angle <= 80) {
    angle = 70;
  }

if (angle > 80 && angle <= 100) {
    angle = 90;
  }
```

```
if (angle >= -40 && angle < 20) {
   angle = -30;
  }

  if (angle >= -60 && angle < 40) {
   angle = -50;
  }

if (angle >= -80 && angle < 60) {
   angle = -70;
  }

if (angle >= -100 && angle < 80) {
   angle = -90;
  }

  int x = sin(angle)*power;
  int y = cos(angle)*power;

  if (angle > 102 || angle < -102) {
   y = 0;
  }

  // Regra de tres que define o valor da tensao gerada pelo pwm em JEV
  // Aqui os valores de maximo e minimo se invertem por causa da conexao da placa que esta invertida
  JEV = MAX*(1 - y);

  // Regra de tres que define o valor da tensao gerada pelo pwm em JEH
  JEH = MAX*(x + 1)/2;
 }



// ROTINAS DO JOYSTICK 2
void joy2(int angle, int power) {
   if (angle >= -12 && angle <= 12) {
   angle = 0;
  }
   else if (angle >= -102 && angle <= -78) {
   angle = -90;
  }
   else if (angle >= 168 || angle <= -168) {
   angle = 180;
  }
   else if (angle >= 78 && angle <= 102) {
   angle = 180;
  }

  int x = sin(angle)*power;
  int y = cos(angle)*power;

  // Regra de tres que define o valor da tensao gerada pelo pwm em JEV
  JDV = MAX*(y + 1)/2;

  // Regra de tres que define o valor da tensao gerada pelo pwm em JEH
  JDH = MAX*(x + 1)/2;
}



void loop() {

  if(bluetooth.available())
  {
   char inChar = (char)bluetooth.read();
   //Serial.print(inChar);
   if (inChar == '#')
   {
    index = 0;
    inData[index] = inChar;
    started = true;
    ended = false;
    index++;
   }
   else if (inChar == ';')
   {
    inData[index] = inChar;
    ended = true;
    return;
   }
   else
   {
    inData[index] = inChar;
    index++;
   }
  }
  // Verifica se as mensagens sao montadas corretamente no array ao fim da mensagem
  if (ended){
  for (int i = 0; i <= index; i++)
   Serial.print(inData[i]);
  }
  /* Aqui, o pacote da mensagem acabou  */
```

```
  if(started && ended)
  {
    // processar pacote
    String message[3];
    String aux = "";
    int count = 0;
    int data[2];

    // i começa em 1 para ignorar a # que indica
o início da mensagem
    for (int i = 1; i<= index; i++) {
      if (inData[i] == ' ' || inData[i] == ';') {
        message[count] = aux;
        count++;
        aux = "";
      }
      else {
        aux = aux + inData[i];
      }
    }

    // converte dados de string para inteiro
    data[0] = atoi(message[1].c_str());
    data[1] = atoi(message[2].c_str());

    // identifica o joystick que gerou a
mensagem e chama a rotina correspondente
    if (message[0] == "1")
      joy1(data[0], data[1]);
    else
      joy2(data[0], data[1]);


    // as config iniciais sao retomadas para o
proximo pacote
    started = false;
    ended = false;
    index = 0;
    inData[index] = '\0';
  }
}
```