UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Plataforma Web para Caracterização de Drones

Autor: Bruno Caceres Carrilho

Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2016

Bruno Caceres Carrilho

Plataforma de controle de drone via web

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

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos
2016

Carrilho, Bruno Caceres
C898p Plataforma Web para Caracterização de Drones / Bruno Caceres Carrilho; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2016.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Sistemas Embarcados Linux. 2. VANT. 3. Quadricópetors. 4. Drones. I. Título.

FOLHA DE APROVAÇÃO

Nome: Bruno Caceres Carrilho

Título: "Pataforma Web para caracterização de Drones"

Trabalho de Conclusão de Curso defendido e aprovado em 24 / 11 / 2016,

com NOTA 9,0 (NOVE, ZERO), pela Comissão Julgadora:

Prof. Associado Evandro Luis Linhari Rodrigues - Orientador - SEL/EESC/USP

Mestre André Luis Martins - Doutorando - SEL/EESC/USP

Mestre Leonardo Mariano Gomes - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado José Carlos de Melo Vieira Júnior

Dedicatória

Este trabalho de conclusão de curso é dedicado ao meu irmão, minha mãe, minha tia, ao meu falecido avô e toda a minha família, que sempre estiveram comigo.

Bruno Caceres Carrilho.

AGRADECIMENTOS

Primeiramente a minha mãe, Maria Isabel, que bancou todos os meus estudos durante todo esse tempo, mesmo que sendo difícil para ela se manter, manter meu irmão e a mim estudando em locais diferentes por mais de 7 anos, e sempre incentivando a estudar, e mesmo assim nunca nos deixou faltar nada.

À minha tia, Angela Maria, que sempre ajudou minha mãe no que pode, bem como ao meu irmão e eu, e pelo carinho que tem.

Ao meu irmão, André, que por ser mais velho sempre me ensinou, sempre esteve comigo me ajudando, um exemplo de pessoa.

Ao meu falecido avô, Wenceslau, um homem de caráter, justo e bom, que foi por boa parte da minha vida o meu pai, o homem em quem me inspiro como ser humano, e espero um dia ser tão digno quanto ele foi.

Ao meu orientador, Prof. Evandro L. L. Rodrigues, que nessa caminhada de aprendizagem me ensinou boa parte do que sei, pela paciência que tem, e pelo excelente professor que é.

Aos meus amigos de infância, Lucas R. Sartori; Fernando H. Costa. Que sempre estão comigo, seja o momento que for.

Aos meus amigos de faculdade, Matheus C. Bezerra; Gabriel N. Laureano; Marcelo A. Oliveira, pelos momentos sofridos que passamos estudando juntos, e pelos momentos que nos divertimos jogando vídeo game.

Aos meus amigos, Bruno F. M. Callegaro; Tiago Martins, que me auxiliaram neste trabalho, e pela paciência que tiveram comigo.

Bruno Caceres Carrilho.

"*Duas coisas são infinitas: o universo e a estupidez humana. Mas, no que respeita ao universo, ainda não adquiri a certeza absoluta*."
Albert Einstein

"*A primeira regra de qualquer tecnologia utilizada nos negócios é que a automação aplicada a uma operação eficiente aumentará a eficiência. A segunda é que a automação aplicada a uma operação ineficiente aumentará a ineficiência*."
Bill Gates

"*Talk is cheap. Show me the code*."
Linus Torvalds

RESUMO

Caceres, Bruno **Plataforma de Controle de Drone via Web**. Trabalho de Conclusão de Curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2016.

Os veículos aéreos não tripulado (VANT) quadrotores, também conhecidos como quadricópteros ou drones, tem ganhado espaço. Comercialmente, para aplicações em agropecuária, e agricultura de precisão, bem como aplicações militares em exploração ou vigilância de fronteiras, por exemplo. Este trabalho tem por objetivo o desenvolvimento de uma plataforma de testes e caracterização de drones via web em Sistemas Embarcados Linux, que possibilitará ao usuário realizar comandos via web e analisar a resposta do drone aos comandos aplicados. Isso possibilitará uma rápida caracterização do drone, assim possibilitando que futuramente a automação do drone possa ocorrer via web, por meio de outros algoritmos que determine rotas que o mesmo deverá percorrer.

Palavras-chave: Veículos Aéreos Não Tripulado (VANT), Quadrotores, Quadricópteros, Drones, Sistemas Embarcados Linux.

ABSTRACT

The unmanned aerial vehicle (UAV) quadrattors, also known as quadcopters or drones, has gained ground. Commercially, for applications in agriculture, and precision agriculture, and military applications in operation or border surveillance, for instance. This work aims at the development of a platform for testing and characterization of drones through web in Linux embedded systems, which enable the user to perform commands through internet and analyze the drone's response to the applied commands. This will enable a fast characterization of the drone, thus enabling future automation of the drone may occur through web by making use of others algorithms that determine which routes it should go.

Keywords: Unmanned Aerial Vehicle (UAV), Quadrattors, Quadcopters, Drones, Linux Embedded Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS

AM	Amplitude Modulation
AMDSB	Amplitude Modulation Double Side-Band
AMVSB	Amplitude Modulation Vestigial Sideband
ANDSB/SC	Amplitude modulation Double SIde-Band / Supressed Carrier
ARM	Advanced RISC Machine – Máquina Avançada RISC
ASK	Amplitude Shift Keying
ESC	Electronic Speed Controller – Controlador Eletrônico de Velocidade
FM	Frequency Modulation
FSK	Frequency Shift Keyong
GFSK	Gaussian Frequency Shift Keying
HDMI	High-Definition Multimedia Interface – Interface Multimídia de Alta Definição
HTTP	Hypertext Transfer Protocol – Protocolo de Transferência de Hipertexto
IMU	Inertial Measurement Unit – Unidade de Medida Inercial
MVC	Model-View-Controller – Modelo-Visão-Controlador
PHP	Hypertext Preprocessor – Pre-processador de Hipertexto
PM	Phase Modulation
PSK	Phase Shift Keying
RF	Rádio Frequência
SSH	Secure Shell – Capsula Segura
UAV	Unmanned Aerial Vehicle - Veículo Aéreo Não Tripulado
USB	Universal Serial Bus – Barramento Serial Universal
VANT	Veículo Aéreo Não Tripulado

SUMÁRIO

CAPÍTULO 1

1. Introdução

Originalmente, os veículos aéreos não tripulado (VANT) foram desenvolvidos pelos militares, com a intenção de explorar campos inimigos e realizar operações de ataques sem que a vida de nenhum soldado fosse exposto aos perigos oriundos de um ambiente em batalha. Foram empregados nos anos 80 em atividades de vigilância urbana, costeira, em buscas e resgates, etc [1]. Os primeiros VANTs militares eram baseados em aeronaves tripuladas, possuindo um ou dois motores, e com aerodinâmica baseada em asas, com princípio de voo de planadores. Após uma breve evolução dos VANTs, levou-se a uma popularização e com uma redução do custo passou então a fazer parte do ambiente civil. Com a evolução dos VANTs, surgiu o quadricóptero, popularmente conhecido como drone, que por sua vez tem o princípio de voo semelhante aos helicópteros, porém possuem quatro motores e seu deslocamento baseia-se na alteração de rotação dos motores [2].

Uma vantagem para a fabricação de VANTs quadricóptero (drones), é a possibilidade de ele pairar no ar, e também uma simplicidade construtiva de operação comparado aos helicópteros, pois não demanda de mecanismos sofisticados e caros para a geração do ângulo de ataque e assim produzir um movimento [3]. Em compensação para que os drones pairem no ar, é exigido um processo de controle preciso, fazendo-se necessário um sistema de aquisição de dados utilizando sensores presentes na unidade de medida inercial.

Atualmente, vários modelos permitem acesso à leitura de sensores embarcados e ao seu controle via rádio frequência (RF). Isso possibilita pleno controle dos drones incluindo a realização de determinadas tarefas, inicialmente simples, como a determinação e execução de rotas que devem ser percorridas. Existem várias aplicações de drones desse tipo, como por exemplo, em

agricultura de precisão em que os drones, em rotas definidas, sobrevoam áreas com o intuito de identificar falhas nutricionais, estresse hídrico em plantações e falta de uniformidade nos sistemas de irrigação [4].

O desenvolvimento da plataforma de teste de drones via web em Linux embarcado, possibilitará ao usuário realizar comandos, composto por movimentos simples já predeterminados, como rotacionar o drone sobre o próprio eixo e movimentos de subida e descida, entre outros e observar e realizar medidas do comportamento apresentado pelo drone. Isso possibilitará uma fácil implementação de algoritmos de voos autônomos acompanhados e sensoriados pela internet, e também, por exemplo, uma tarefa audaciosa como a troca de uma lâmpada queimada de uma residência, mesmo que o usuário não esteja em casa. Uma outra aplicação possível seria a utilização em ambientes industriais, transferindo uma peça de uma esteira de produção para outra, ou por exemplo combinar movimentos predeterminados para criar uma sequência que atenda a tarefas mais complicadas, podendo realizar manutenção de equipamentos.

1.1. Motivação

A primeira motivação deste trabalho de conclusão de curso é a utilização do drone e a aprendizagem do software de estabilização de voo que nele é utilizado, e a segunda motivação deste trabalho é a aprendizagem de programação web que atualmente se faz necessário para um profissional de engenharia elétrica. Então, por meio do desenvolvimento de uma central de controle, instigará com que o aluno investigue os softwares de voo para um correto e mais eficiente uso na construção da central de controle web, proporcionando um fácil acesso ao controle de drones, e assim propor uma estação de comandos determinados que poderá ser facilmente utilizada.

Uma outra motivação é o desenvolvimento eficiente deste trabalho para que possa facilitar o desenvolvimento de voo autônomo via web, portanto auxiliando trabalhos futuros tanto do aluno quanto da escola.

1.2. Objetivos

1. Estudo de linguagens de programação Web.
2. Desenvolvimento do sistema de comunicação RF da central e comandos básicos.
3. Desenvolvimento do *Web Server* que servirá de interface de controle do drone.

1.3. Justificativa

Este trabalho se justifica devido a necessidade de se testar drones, identificar características do VANT e falhas, para que o mesmo não apresente defeitos ou incompatibilidade entre os componentes, utilizando-se a internet como rede de comunicação para envio e recepção de dados, de tal forma que possa se fazer testes e monitoramento onde possua conectividade com a internet.

1.4. Organização do Trabalho

A organização deste trabalho apresenta a fundamentação teórica para o desenvolvimento do projeto da estação de controle de drone via web, o material utilizado e suas especificações, a metodologia cientifica empregada no projeto, os resultados obtidos pelos testes realizados com o protótipo, e por fim as conclusões sobre os resultados apresentados pelo projeto.

CAPÍTULO 2

2. Embasamento teórico

2.1. Programação Web

Analisando-se o problema proposto se faz necessário um software que estruture e gerencie a aplicação web, e que seja capaz de receber e transmitir dados via web para que o *webserver* seja implementado, tais estruturas são conhecidas como "*Frameworks Web*". Esses frameworks web fornecem suporte para utilização de bancos de dados, gerenciamento de sessões e torna o código web reutilizável. As aplicações web são feitas por requisições HTTP de tal forma que para cada requisição a aplicação retorne uma resposta adequada [5]. O modelo mais popular de arquitetura para implementação de interfaces para usuário usado nos frameworks de aplicações web, tem como base o modelo MVC "*model-view-controller*", que se baseia em 3 camadas de aplicação, a camada de interação do usuário (*view*), a camada de manipulação dos dados (*model)* e a camada de controle (*controller*).

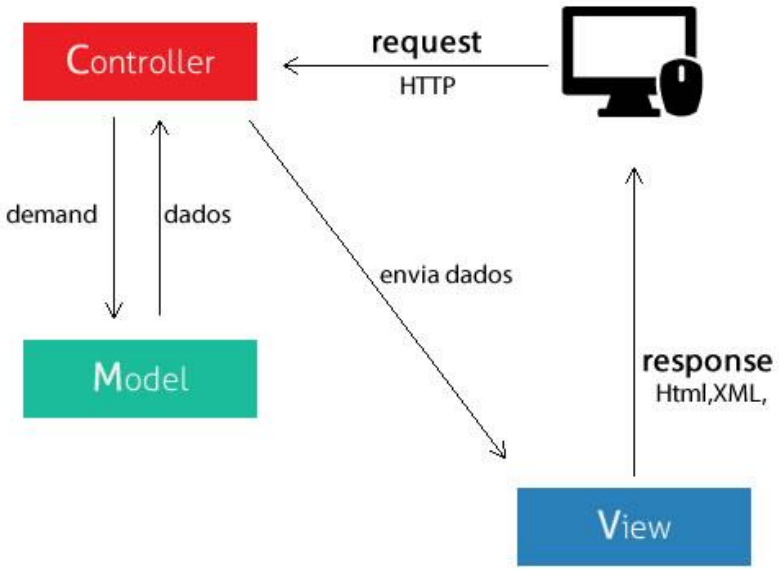

Figura 1: Modelo MVC. FONTE: Google.

A camada *Model* se responsabiliza pela leitura e escrita de dados, regras e lógica da aplicação. A camada *View* funciona como camada de saída de dados, exibindo os dados e realizando a interação com o usuário. A camada *Controller* é a responsável por receber as requisições e dados do usuário, convertendo em comando para as camadas *Model* ou *View*, assim ditando qual *model* será usado e qual *vie*w será mostrado ao usuário. Então, as interações entre as camadas ocorrem da seguinte forma: um *model* salva as informações que são recuperadas de acordo com os comandos do *controller* e são apresentados na camada *view*, a camada *view* gera a nova saída a ser vista pelo usuário baseado nas mudanças realizadas na camada *model*, e o *controller* pode enviar comandos para o *model* para realizar o update do estado do *model*, como por exemplo editar um documento, ou então enviar comandos associados a camada *view* alterando a apresentação ao usuário, como por exemplo a rolagem de página [6].

Outro aspecto de interesse com relação aos frameworks web é de como ocorre o fluxo de operação e execução. Notavelmente a maioria dos frameworks web possuem uma estrutura de fluxo de execução, que é baseada em todas as requisições a serem passadas por comandos definidos pelo framework web, como por exemplo requerer o acesso à página index.php que se encontra no diretório raiz de uma aplicação em PHP [5].

Contudo, existem diversos frameworks web, cada um é mais indicado para determinados tipos de linguagem de programação, e linguagens de programação web.

. Para Java

- . Apache Structs
- . JavaServers Faces
- . Jt Design Pattern Framework
- . Apache Wicket

. Para PHP

- . CakePHP
- . Laravel
- . CodeIgniter
- . Symfony
- . Yii
- . Zend Framework

. Para Python

- . Django
- . Flask
- . Pyjamas
- . web2py
- . Pylons
- . Zope

. Para Ruby

- . Ruby on Rails
- . Ramaze

Para o projeto proposto, se utilizou do framework web Flask, por questões de simplicidade de utilização, por ser um framework leve e próprio para Python, que é uma linguagem em alta nos dias atuais, e possuir bibliotecas que proporcionam um fácil acesso ao hardware, além do suporte web que possui.

2.2. Drone

Os drones possuem quatro partes fundamentais: estrutura, motores, circuito de navegação, e bateria. A estrutura do drone define qual composição do aspecto construtivo mecânico, indicando se o mesmo será trimotor, quadrimotor, e assim por diante, pode-se observar na Figura 2 um modelo hexacóptero. Os motores definem a agilidade que o drone terá, uma vez que a potência do motor influi diretamente na rotação média do motor, causando então que pequenas angulações causem um movimento rápido. O circuito de navegação pode ser dividido em duas partes: circuito de potência que reduz a tensão da bateria para 5V, além da alimentação e controle dos motores e o circuito microcontrolado que recebe o software de navegação. Modelos comerciais de circuitos de navegação, também conhecidos como "*pilots*", podem vir de fábrica integrados em um único circuito.

Figura 2: Hexacóptero. FONTE: Google.

Os modelos mais comuns são quadricópteros, entretanto podem ser separados em dois tipos de quadricópteros referente ao mecanismo de voo, uma vez que os quadricópteros podem voar em xis ou em cruz. A mecânica de voo pode ser compreendida levando-se em consideração a direção em que o movimento deve ocorrer, o comportamento dos motores e a posição da estrutura. Na mecânica de voo em xis dois motores adjacentes devem ter sua velocidade de rotação aumentada, enquanto que os outros dois devem ter sua velocidade de rotação reduzida causando o movimento do quadricóptero, assim sua estrutura ficará alinhadamente em xis com a direção do movimento. Para a mecânica de voo em cruz, apenas um motor deve ter sua velocidade de rotação aumentada, enquanto que o motor oposto deve ter sua velocidade reduzida, assim sua estrutura ficará em cruz com a direção do movimento. Outro ponto importante, e para que ambos os mecanismos de voo funcionem devidamente, deve-se orientar a rotação dos motores adjacentes em sentido contrário, assim garantindo um momento de inercia resultante nulo sobre o drone [7].

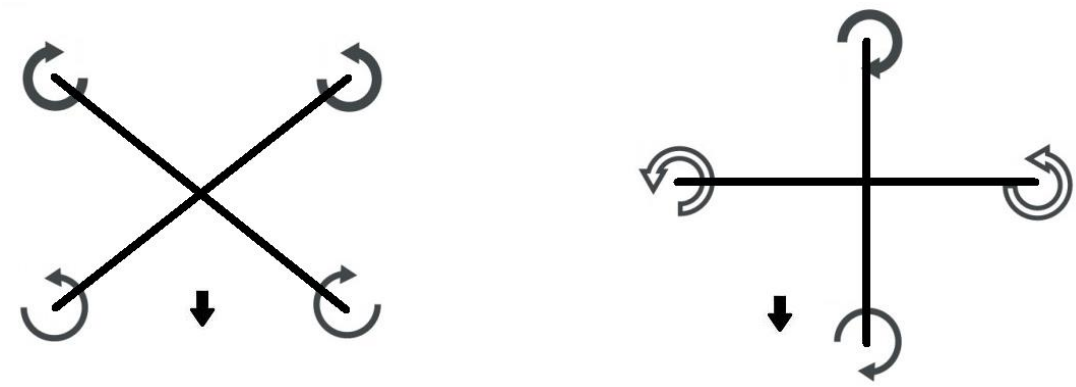

Figura 3: Mecanismos de voo.

A velocidade dos motores é composta de quatro componentes para compor os movimentos possíveis. Para cada motor existe um equacionamento dessas componentes que depende de sua posição espacial no drone, bem como o sentido de rotação das hélices. As quatro componentes são divididas em: velocidade de rotação media, e ângulos *pitch*, *yaw* e *roll*.

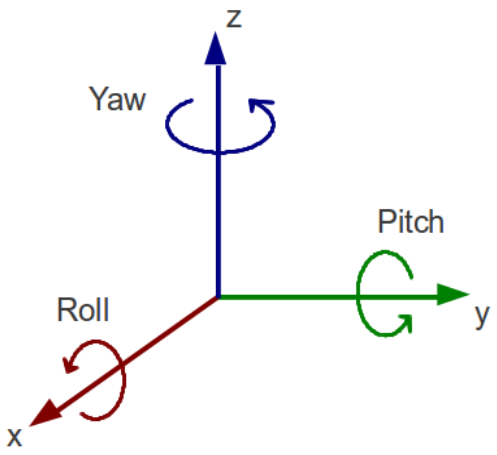

Figura 4: Eixo de rotação. FONTE: Google.

Tendo como referência de movimento o eixo X, pode-se então definir o ângulo *pitch* como o ângulo de ataque, ou seja, o ângulo que proporcione os movimentos de avanço e retorno. Já o ângulo *roll* proporciona movimentos laterais, também conhecidos como "rolagem", e o ângulo *yaw* proporciona uma rotação sobre o próprio eixo.

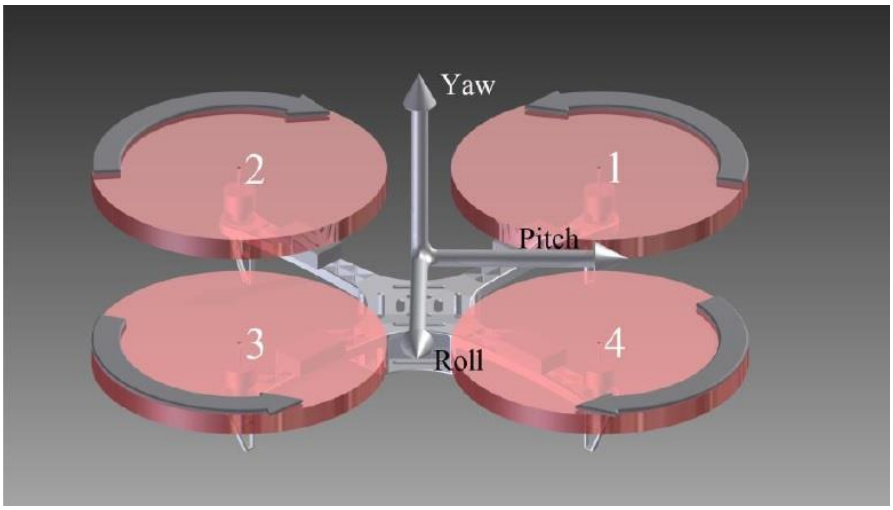

Figura 5: Posicionamento dos motores e do eixo de rotação. FONTE: CALLEGARO, B. F. M.,2014 p. 39.

Como apresentado na Figura 5, pode-se fazer o equacionamento da rotação de cada motor levando-se em consideração as componentes do eixo de rotação, e os efeitos de empuxo dos motores e o equilíbrio de torque sobre o drone podemos equacionar a velocidade dos motores referente as ações de controle: U_{pitch}, U_{roll} e U_{yaw} [8]. Resultando em:

$$M_1 = \sqrt{W_{med} + U_{pitch} + U_{roll} - U_{yaw}} \quad (1)$$

$$M_2 = \sqrt{W_{med} + U_{pitch} - U_{roll} + U_{yaw}} \quad (2)$$

$$M_3 = \sqrt{W_{med} - U_{pitch} - U_{roll} - U_{yaw}} \quad (3)$$

$$M_4 = \sqrt{W_{med} - U_{pitch} + U_{roll} + U_{yaw}} \quad (4)$$

A partir das equações de (1) à (4) pode-se implementar um controle por sinais PWM que indicam as ações de controle W_{med}, U_{pitch}, U_{roll} e U_{yaw}, de tal modo que em 10% do duty clycle corresponda a uma entrada mínima, atingindo valor máximo em 90% do duty cycle, que é o padrão utilizado em aeromodelismo.

2.3. Comunicação RF

A comunicação RF (Rádio Frequencia) é um recurso tecnológico de telecomunicações utilizado para proporcionar comunicação por intermédio da transcepção de informações previamente codificadas em sinal eletromagnético que se propaga através do espaço [11]. Baseia-se em três componentes que possibilitam a comunicação entre dois pontos, são eles: o transmissor, o meio de transmissão e o receptor. O transmissor converte os sinais de informações que se deseja transmitir em ondas eletromagnéticas, possibilitando a transmissão através do espaço com o auxílio de uma antena. O receptor é o responsável pela recepção e decodificação dos sinais eletromagnéticos captados pela antena, de tal forma que regenere a informação que foi transmitida.

Para que a comunicação RF seja possível os sinais contendo as informações são modulados, assim o espectro do sinal é deslocado para uma banda mais adequada para transmissão, uma vez que as antenas irradiam com eficiência quando o comprimento de onda do sinal corresponde à sua abertura física. Outro benefício da modulação é facilitar a multiplexação de canais, permitindo que um número grande de usuários partilhe o mesmo canal ou a mesma informação. Em RF a modulação consiste em variar algum parâmetro de um sinal senoidal (chamado de portadora) com o sinal modulante (informação), portanto a modulação analógica gira em torno de três possibilidades (AM – *Amplitude Modulation*, FM – *Frequency Modulation*, PM – *Phase Modulation*) divididas em duas categorias, modulação em amplitude (AM) ou modulação em ângulo, dentro da modulação em ângulo existem duas possibilidades, modulação em frequência (FM) ou modulação em fase (PM) [12].

Os métodos mais comuns de modulação AM analógica são conhecidos como: AMDSB (*Amplitude Modulation Double Side-Band*), ANDSB/SC (*Amplitude modulation Double SIde-Band / Supressed Carrier*), SSB (*Single Side Band*) e AMVSB (*Amplitude Modulation Vestigial Sideband*).

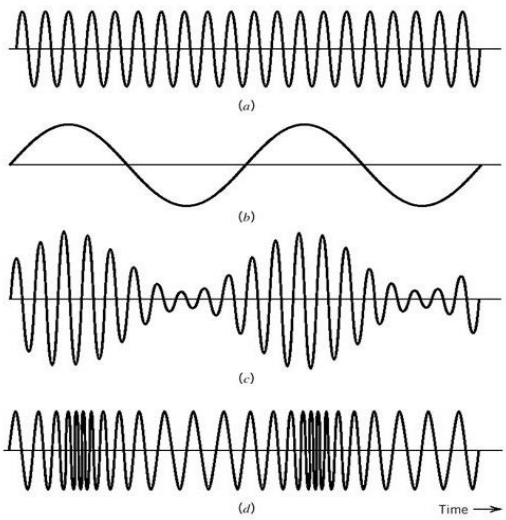

Figura 6: (a) Portadora. (b) Sinal. (c) Modulação em Amplitude. (d) Modulação em frequência.
FONTE: Google.

A modulação RF digital também gira em torno de se variar algum parâmetro de um sinal senoidal, entretanto baseada em uma deslocação (*shift*), portanto das modulações digitais são: ASK (*Amplitude Shift Keying*), FSK (*Frequency Shift Keyong*), PSK (*Phase Shift Keying*).

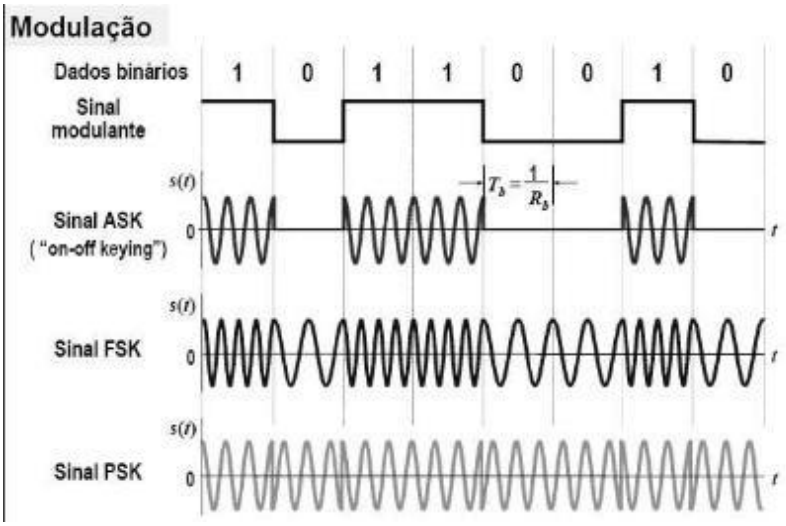

Figura 7: Modulação ASK, FSK, PSK. FONTE: Google.

O modulo nRF24L01 trabalha com a modulação FSK/GFSK (*Gaussian Frequency Shift Keying*) uma versão do FSK que funciona similarmente modulando a frequência utilizando os símbolos digitais, porem ao invés de abruptamente variar a frequência no início de cada símbolo o GFSK filtra os dados digitais com um filtro Gaussiano, fazendo com que as transições sejam mais suaves. Isso traz a vantagem de reduzir a banda, reduzir a interferência com os canais vizinhos, ao custo de aumentar a interferência entre símbolos [14].

Em um único canal o modulo nRF24L01 é capaz de se comunicar, ou melhor receber dados de outros seis módulos nRF24L01, isso é possível graças ao método de se utilizar "*pipes*", ou tubos, que são endereços de 40 bits que determinam quando deve-se ler mediante todo o "ruído de rádio"[10].

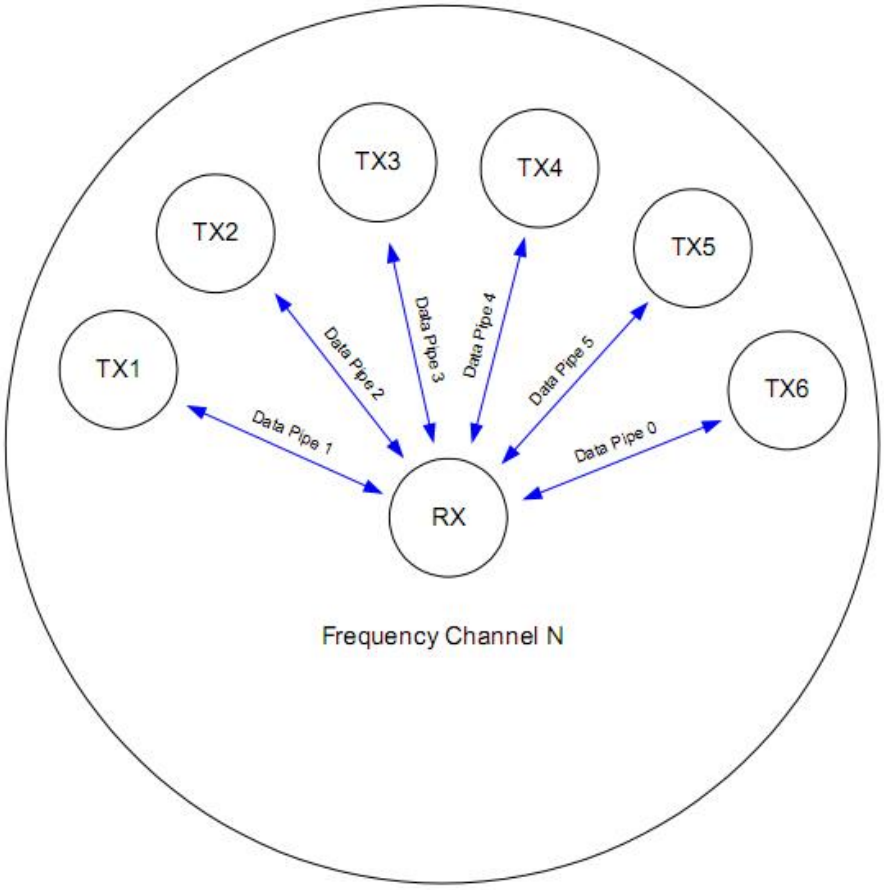

Figura 8: Diagrama de pipes da nRF24L01. FONTE: Google.

CAPÍTULO 3

3. Materiais e Métodos

3.1. Materiais

O projeto utilizou-se de uma plataforma Raspberry Pi modelo B (Figura 9), que é utilizada como *webserver* para a central de controle, uma vez que as configurações desta plataforma permitem o acesso à internet, e é capaz de receber um sistema operacional. As especificações da Raspberry Pi modelo B podem ser vistas na Tabela 1.

Processador	Broadcom BCM2835 (Contem ARM1176JZF-S (ARM11 usando uma arquitetura ARMv6) com floating point, rodando a 700MHz)
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, OpenVG 1080p30 H.264 high-profile encide/decode, 250MHz
Memória	512MB RAM
Armazenamento	Secure Digital/SD / MMC / slot SDIO
Periféricos	2 USB, Ethernet, 3.5mm jack para saída de áudio, HDMI
Consumo	700mA
Pinos de GPIO	26, incluindo I2C, UART, SPI, PWM
Peso	40g

Tabela 1: Especificações Raspberry Pi modelo B.

Figura 9: Raspberry Pi modelo B. FONTE: Google.

Para complementar a plataforma de drone, faz-se necessário a utilização de um Arduino Uno (Figura 10) em comunicação USB com a Raspberry Pi. O Arduino Uno faz o gerenciamento de dados a serem transmitidos pelo módulo RF para o drone, uma vez que a tentativa de se utilizar o módulo RF conectado diretamente na Raspberry Pi trouxe algumas falhas de transmissão. As especificações do Arduino Uno podem ser vistas na Tabela 2.

Microcontrolador	ATmega328P rodando a 16MHz
Memória	Flash Memory 32KB, SDRAM 2KB (ATmega328P)
Armazenamento	EEPROM 1KB (ATmega328P)
Periféricos	USB, 14 pinos de I/O (incluindo 6 PWM), 6 pinos de entradas analógicas
Operação	1,8V a 5,5V
Peso	25g

Tabela 2: Especificações Arduino Uno.

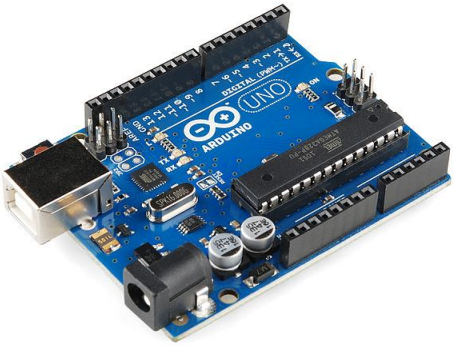

Figura 10: Arduino Uno. FONTE: Google.

Tendo em vista que o projeto se baseia em uma plataforma de teste de drone, faz-se necessário a utilização de um drone para o devido desenvolvimento e testes, porem os drones comerciais não cabiam dentro do orçamento. Então foi proposto que o drone fosse montado baseado no trabalho de conclusão de curso "Desenvolvimento de um Veículo Aéreo quadrirotor com sistema de estabilização baseado no filtro de Kalman" [8]. Portanto, utilizou-se como base para o software de navegação a placa STM32F4-*Discovery* (Figura 11), como pode ser visto na Tabela 3, as especificações da plataforma utilizada no drone.

Microcontrolador	ARM Cortex-M4 STM32F407VGT6 rodando a 168MHz
Memória	192KB RAM
Armazenamento	1MB Flash memory
Periféricos	USB, 3.5mm jack para saída de áudio, LIS302DL ou LIS3DSH ST MEMS acelerômetro de 3 eixos, MP45DT02 ST MEMS áudio sensor.
Consumo	Máximo 244µA/MHz
Pinos de IO	100 pinos
Peso	60g

Tabela 3: Especificações STM32F4-Discovery.

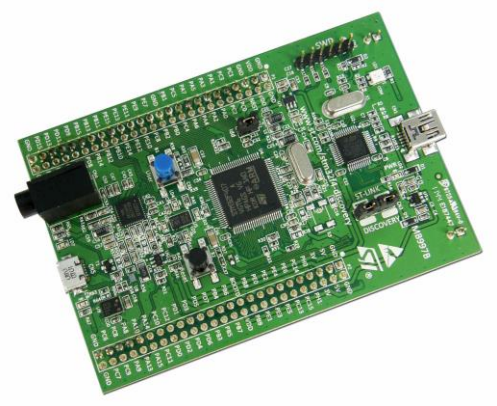

Figura 11: STM32F4-Discovery. FONTE: Google.

Além da plataforma base para o navegador, o drone necessita de uma estrutura, motores, controladores eletrônicos de velocidade (ESC) para os motores, hélices, bateria, receptor RF e a unidade inercial (IMU). Para tal utilizou-se a estrutura Frame X525 (Figura 12) que é uma estrutura de alumínio, leve e resistente. Os motores escolhidos foram quatro Turnigy motor de 1250kv (Figura 13), para este motor optou-se por hélices de fibra de carbono de dimensões 9x4.7 polegadas, portanto a escolha dos quatro ESCs foi baseada na necessidade de corrente que os motores podem exigir, portanto utilizou-se o ESC Hobby King de 30A (Figura 14) e uma bateria LiPo de três células de 2200mAh com tensão entre 11,1V e 12,6V (Figura 15).

Figura 12: Frame X525. FONTE: Google.

Para que o drone possua uma característica veloz, com agilidade de movimentos, optou-se por uma combinação de ESC e motor razoavelmente fortes, como o Turnigy motor de 1250kv e o ESC de 30A.

Figura 13: Turnigy motor de 1250kv. FONTE: Google.

Figura 14: ESC Hobby King de 30A. FONTE: Google.

Figura 15: Bateria LiPo de três células. FONTE: Google.

Para o controle realizado pelo software de navegação faz-se necessário a utilização de uma unidade de medida inercial (IMU), utilizou-se os módulos GY-521 (Figura 16) e GY-271 (Figura 17). O módulo GY-521 contém o acelerômetro ADXL345 e o giroscópio L3G4200D de três eixos, já do GY-68 contém apenas o magnetômetro HMC5883L de três eixos. A escolha de ambos os sensores foi baseada no baixo custo e fácil obtenção.

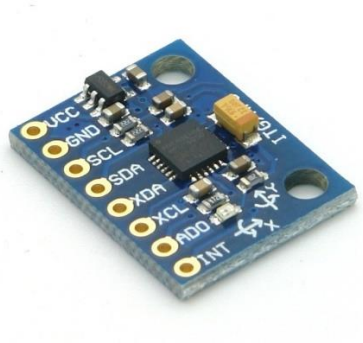

Figura 16: GY-521 Acelerômetro e giroscópio. FONTE: Google.

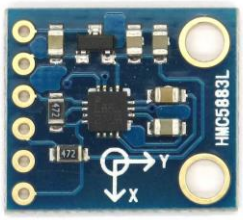

Figura 17: GY-271 Magnetômetro. FONTE: Google.

Por fim o modulo nRF24L01 (Figura 18) foi utilizado para fazer a comunicação entre a central e o drone, enviando os comandos de voo e recebendo dados da telemetria, o modulo trabalha em uma frequência de 2,4GHz. Este modulo tem a capacidade de transmitir pacotes de até 32 bytes e uma frequência de atualização máxima de 2Mbps.

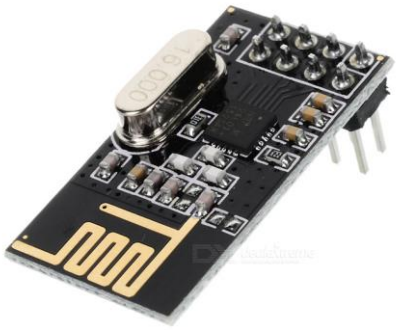

Figura 18: Modulo nRF24L01. FONTE: Google.

3.2. Métodos

Primeiramente realizou-se a instalação do sistema operacional Raspbian com o auxílio de um gravador de cartões SD na plataforma raspberry pi através do software Win 32 Disk Imager. Imediatamente após a instalação do sistema operacional realizou-se a configuração de rede na plataforma raspberry pi, com o auxílio de um cabo HDMI, um monitor e teclado, uma vez que ainda não havia modos de acesso remoto à plataforma justamente por ser a primeira vez em que se fazia o boot no sistema e as configurações de rede ainda não permitiam o acesso remoto. Para se configurar a rede editou-se o arquivo “interfaces” no diretório “etc/network”. Os seguintes comandos indicam o procedimento realizado para a configuração de rede e o conteúdo final do arquivo “interfaces”:

```
pi@raspberry: sudo nano /etc/network/interfaces

Conteudo Interfaces:

source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static

#your static IP
address 10.235.10.47

#your gateway IP
gateway 10.235.10.1
netmask 255.255.255.0

#your network address `family`
network 10.235.10.0
broadcast 10.235.10.255

#DNS
dns-nameservers 8.8.8.8
dns-nameservers 8.8.4.4

allow-hotplug wlan0
iface wlan0 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

allow-hotplug wlan1
iface wlan1 inet manual

    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Uma vez com as configurações de redes pronta, configurou-se a porta de acesso da 22 para a 22000 do acesso remoto SSH (Secure Shell). O SSH que é um protocolo para acessar uma máquina remota em segurança, permitindo a execução de linha de comando (e com alguns ajustes, programas gráficos), transferência de arquivos, e até mesmo criar redes privadas virtuais seguras através da Internet. A grande vantagem de usar esse protocolo é a segurança e a simplicidade do acesso. Com ele é permitido trabalhar como se estivesse usando um terminal localmente e também fazer tudo que normalmente é possível neste tipo de interface. Por ser uma ferramenta de acesso remoto, pela questão de segurança justifica-se a troca da porta de acesso remoto. Para tal basta procurar no arquivo "etc/ssh/sshd_conf" por Port 22 e altera-lo para Port 22000. Após a edição realiza-se o restart do SSH utilizando a linha de comando:

```
pi@raspberry: sudo /etc/init.d/ssh restart
```

O desenvolvimento da central iniciou-se pela programação do *webserver*, primeiramente instalando os pacotes python na Raspberry Pi, pois o flask necessita do python para poder ser instalado, posteriormente instalou-se o flask. Os comandos Linux abaixo indicam o procedimento para essa instalação:

```
pi@raspberry: sudo apt-get update
pi@raspberry: sudo apt-get install python-pip
pi@raspberry: sudo pip install flask
```

Com o flask instalado na plataforma desenvolveu-se o *webserver* contendo inicialmente botões de comandos básicos, como por exemplo, movimento de direita e esquerda. A composição do webserver foi dividida em duas páginas web, a primeira de apresentação contendo um botão que leva para a segunda página onde os comandos foram implementados.

Uma vez feito o desenvolvimento web iniciou-se a implementação da comunicação da central com o drone, para isso utilizou-se do Arduino Uno em comunicação USB com a Raspberry

Pi, e o Arduino Uno se comunicando com o modulo nRF24L01 que realiza a transmissão dos dados. Para realizar a comunicação da Raspberry Pi com o Arduino Uno utilizou-se a biblioteca pyserial-2.5 em python para comunicação serial. Os comandos abaixo resumem a instalação da biblioteca pyserial-2.5:

```
pi@raspberry: wget
http://iweb.dl.sourceforge.net/project/pyserial/pyserial/2.5/pyserial-
2.5.tar.gz

pi@raspberry: unzip pyserial-2.5.tar.gz

pi@raspberry: tar –xvf pyserial-2.5.tar
```

Com a biblioteca instalada conectou-se ambas as plataformas via serial e identificou-se qual dispositivo foi atribuído pelo sistema operacional para o Arduino Uno, para isso utilizou-se os comandos de identificação de dispositivos seriais antes de conectar o Arduino e após conectar o Arduino para se identificar o dispositivo por contraste.

```
pi@raspberry: ls /dev/tty*
```

Observou-se que a Raspberry Pi atribuiu o Arduino Uno como ‘ttyACM0’, essa informação viabilizou uma rápida implementação da comunicação entre ambas as plataformas ao se alterar poucas linhas de código do *webserver*. Por final desenvolveu-se o código em linguagem C para o Arduino Uno para receber os dados vindos da Raspberry Pi e transmitir pelo modulo nRF24L01.

A programação do software de navegação na plataforma STM32F4 ocorreu pela obtenção do código fonte do trabalho “Desenvolvimento de um Veículo Aéreo quadrirotor com sistema de estabilização baseado no filtro de Kalman” [8], esse código foi obtido pelo Bitbucket usando Github e salvo em um desktop e não mais na Raspberry Pi. Para obtenção e gravação do código de navegação utilizou-se uma máquina Linux. Instalação do Git e obtenção do código fonte através do método de clonagem do Git:

```
usr@pc: sudo apt-get install git

usr@pc: mkdir Quad

usr@pc: cd Quad

usr@pc: git clone https://bitbucket.org/bcallegaro/controlador-
quadcoptero.git
```

Com o código de navegação em mãos basta compilar e gravar na plataforma, entretanto, as máquinas compilam o código para suas respectivas arquiteturas, e como queremos utilizar em uma plataforma ARM, deve-se então instalar o compilador para arquiteturas ARM, e também é importante ressaltar que para realizar a gravação do código na placa faz-se necessário a instalação do "st-flash" que é próprio para a plataforma STM32F4. O seguinte trecho indica a instalação do compilador e a instalação do gravador:

```
#INSTALACAO DO GCC-ARM:

usr@pc: sudo apt-get install gcc-arm-none-eabi

#INSTALACAO DO ST-FLASH:

usr@pc: sudo apt-get install libusb-1.0.0-dev git

usr@pc: git clone https://github.com/texane/stlink stlink.git

usr@pc: cd stlink.git

usr@pc: mkdir build && cd build

usr@pc: cmake -DCMAKE BUILD TYPE=Debug ..

usr@pc: make

usr@pc: cd build

usr@pc: sudo cp st-flash /usr/bin

usr@pc: sudo cp *.rules /etc/udev/rules.d

usr@pc: sudo restart udev
```

Criou-se então as rotinas de determinação de ações de controle via RF, integrando a comunicação serial entre a Raspberry Pi e o Arduino, bem como a comunicação SPI entre o Arduino e o modulo nRF24L01. Para tal, se fez o uso da biblioteca "NRF24.h", download

disponível no site do Arduino, com a biblioteca devidamente instalada, implementou-se a comunicação RF.

Uma vez que os comandos são enviados via web para a Raspberry Pi o software do *webserver* faz a codificação do comando e transmite via USB para o Arduino que realiza a transmissão do dado via RF em conjunto com o modulo nRF24L01 e na outra ponta da comunicação RF encontra-se outro conjunto de Arduino e modulo nRF24L01, responsáveis por captar o sinal e aplicar as ações de controle, baseado nos sinais recebidos.

Posteriormente, observou-se que o software original de navegação levava em conta a saída de um PWM em uma frequência de 200Hz para as ações de controle, porém, o ESC utilizado neste projeto trabalha em uma frequência de 500Hz. Realizou-se então um estudo do código fonte do software de navegação e foi possível encontrar o arquivo que possui os parâmetros do PWM, e assim realizar a devida alteração para uma correta utilização dos ESCs. Notou-se que no arquivo "controle_motores.c" encontra-se a configuração do timer, segue os trechos de códigos originais onde ocorre a configuração do mesmo:

```
42        /* Compute the prescaler value */
43        PrescalerValue = (uint16_t) ((SystemCoreClock / 2) / 100000) - 1;
44
45        /* Time base configuration */
46        TIM_TimeBaseStructure.TIM_Period = 500

79        TIM4->CCR1 = 400;  //Pwm tem 1mS de pulso baixo -> 500 - 400 = 100*1/(100e3Hz)
80        TIM4->CCR2 = 400;
81        TIM4->CCR3 = 400;
82        TIM4->CCR4 = 400;

92  void ajustar_velocidade(uint8_t motor, uint16_t velocidade) {
93
94        if(velocidade > 100)
95                  velocidade = 100;
96
97        //Saída no pwm é invertida, buffer com mosfet source comum, logo pulso é invertido.
98        if((motor&0x01) != 0)
99                  TIM4->CCR1 = 400-velocidade;
100
101       if((motor&0x02) != 0)
102                 TIM4->CCR2 = 400-velocidade;
103
104       if((motor&0x04) != 0)
105                 TIM4->CCR3 = 400-velocidade;
106
107       if((motor&0x08) != 0)
108                 TIM4->CCR4 = 400-velocidade;
109 }
```

Nesses trechos originais observa-se que a linha 43 realiza o “prescaler” que funcionará como o *clock* do timer 4, ele está sendo configurado com uma frequência de 100kHz, e portanto, um período de 10µs. Já na linha 46 ocorre a configuração da frequência do PWM, onde o timer recebe o valor de contagem, no caso 500, resultando em uma contagem de 5ms correspondente à uma frequência de 200Hz. As linhas de 79 à 82 configuram o PWM de *duty cycle* minimo recebendo o valor de 400 correspondente à 4ms de nível logico alto, mas como na placa base existe um MOSFET para efeito de proteção da STM32F4 os pulsos são invertidos, ou seja 4ms de pulso em nível logico baixo e 1ms em nível logico alto, resultando em um *duty cycle* mínimo de 10%. A função “ajustar_velocidade” realiza a variação do PWM recebendo valores de controle.

Para alterar a frequência do PWM para 500Hz deve-se alterar na linha 46 para um valor de contagem de 200, bem como substituir todos os valores de contagem 400 por 180 no *duty cycle* mínimo e na função “ajustar_velocidade”, assim mantendo o *duty cycle* mínimo em 10%. Segue os trechos devidamente alterados:

```
42        /* Compute the prescaler value */
43        PrescalerValue = (uint16_t) ((SystemCoreClock / 2) / 100000) - 1;
44
45        /* Time base configuration */
46        TIM_TimeBaseStructure.TIM_Period = 200

79        TIM4->CCR1 = 180;
80        TIM4->CCR2 = 180;
81        TIM4->CCR3 = 180;
82        TIM4->CCR4 = 180;

92  void ajustar_velocidade(uint8_t motor, uint16_t velocidade) {
93
94        if(velocidade > 100)
95                  velocidade = 100;
96
97
98        if((motor&0x01) != 0)
99                  TIM4->CCR1 = 180-velocidade;
100
101       if((motor&0x02) != 0)
102                 TIM4->CCR2 = 180-velocidade;
103
104       if((motor&0x04) != 0)
105                 TIM4->CCR3 = 180-velocidade;
106
107       if((motor&0x08) != 0)
108                 TIM4->CCR4 = 180-velocidade;
109 }
```

Uma vez com as frequências de PWM dos motores ajustadas, fez-se o estudo variando a frequência de PWM a ser utilizada como ação de controle. Chegou-se à conclusão que o software de navegação realiza medidas em 60Hz. Entretanto a frequência padrão do PWM que o Arduino produz é de 500Hz, para alterar a frequência foi utilizada uma biblioteca "PWM" que possui funções de ajuste de frequência para cada pino do Arduino em que se deseja utilizar PWM.

O sistema desenvolvido possui um fluxo de dados apresentado pela Figura 19. Onde nota-se a utilização de cada plataforma, bem como a comunicação entre elas.

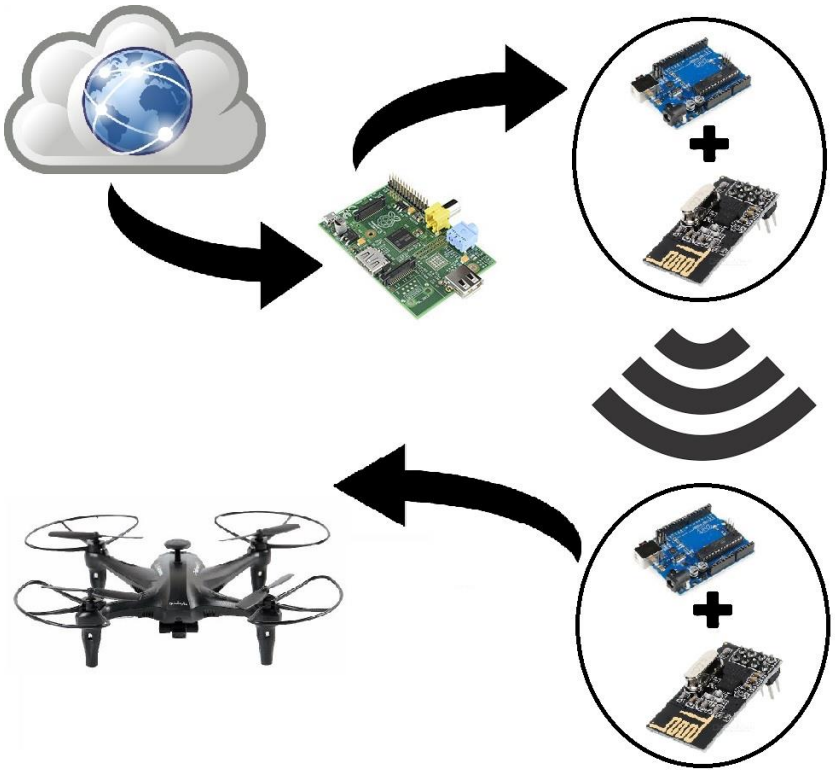

Figura 19: Sistema.

O *webserver* desenvolvido possui um funcionamento baseado nas requisições feitas pelo usuário, pois o programa desenvolvido para o *webserver* recebe como entrada "links", e esses links possuem informações sobre qual ação o usuário deseja realizar, uma vez identificada a ação

desejada o programa retorna a devida página web. A Figura 20 apresenta o fluxograma do *webserver*.

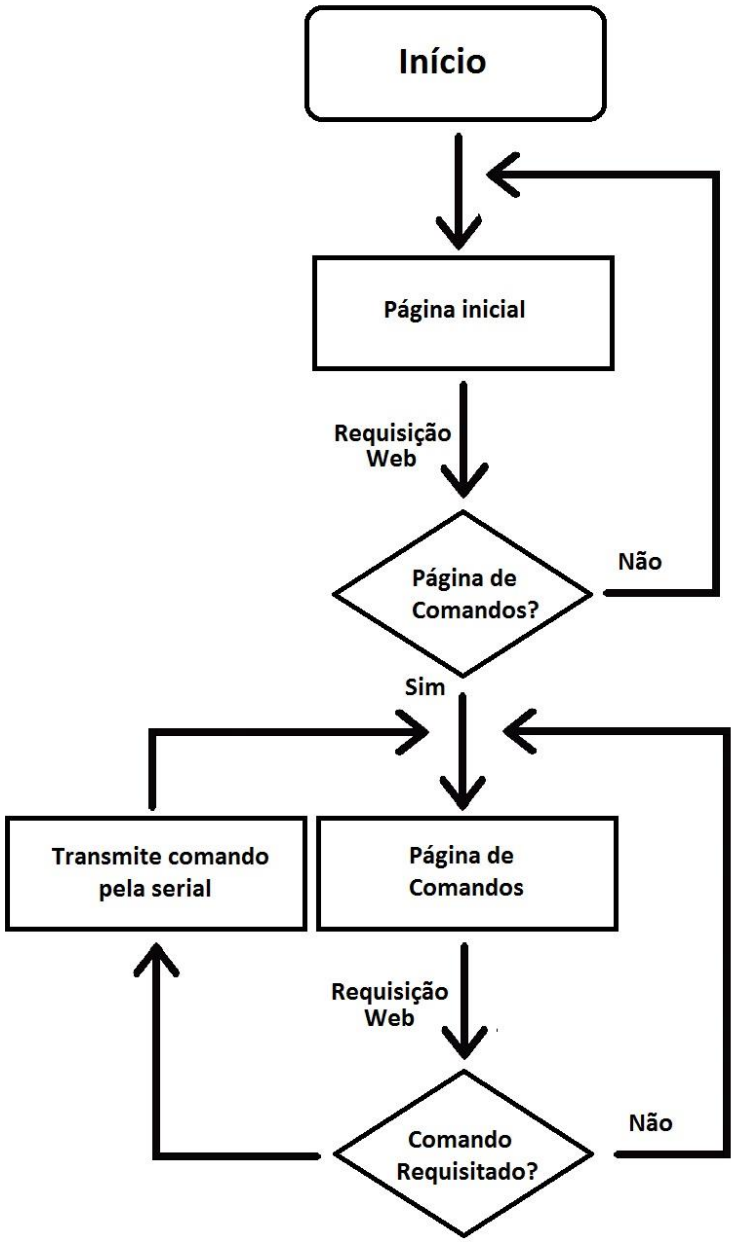

Figura 20: Fluxograma do *webserver*.

CAPÍTULO 4

4. Resultados

A Figura 21 apresenta o design final do *webserver* implementado em html, a página possui conjuntos de botões separados pelas classes de movimento e ações auxiliares, bem como uma pequena explicação sobre as ações de cada botão.

Figura 21: Página web de comandos.

Como cada botão possui um único comando associado pode-se observar o funcionamento de envio de sinal da central utilizando o Arduino receptor do drone como auxilio para identificar se os movimentos estão de acordo com o previsto. Cada botão envia apenas um caractere para o Arduino transmitir e esse caractere é recebido pelo Arduino no drone que o identifica e realiza as ações de controle para gerar o movimento desejado, conectando-se o Arduino utilizou-se no computador como auxilio, uma vez que se pode utilizar o terminal serial do computador para identificar os sinais da recepção RF e contrastar com os que deveriam ser enviado.

Turn On	8
Turn Off	7
Up	6
Down	b
Forward	0
Backward	3
Left	4
Right	9
Cw	1
Ccw	2
Apply	x

Tabela 4: Tabela de codificação dos comandos.

A Tabela 4 apresenta o conteúdo que deve ser enviado ao se pressionar cada botão, então pressionando sequencialmente os botões de “Turn On” à “Apply” deve-se obter como resposta no Arduino a sequência apresentada na Tabela 4. A Figura 20 apresenta os dados recebidos pelo Arduino de recepção.

```
Got payload... 8
Got payload... 7
Got payload... 6
Got payload... b
Got payload... 0
Got payload... 3
Got payload... 1
Got payload... 2
Got payload... 4
Got payload... 9
Got payload... x
```

Figura 22: Dados recebidos pelo Arduino de recepção.

Com o auxílio do osciloscópio realizou-se medidas de variação máxima e mínima do PWM da ação de controle, que por padrão deve possuir uma variação de *duty cycle* aproximadamente dentro do "range" de 10% à 90%. A Figura 23 apresenta a medida de *duty cycle* e frequência realizada para o caso de mínimo *duty cycle*, e a Figura 24 apresenta a medida de *duty cycle* e frequência realizada para o caso de máximo *duty cycle*.

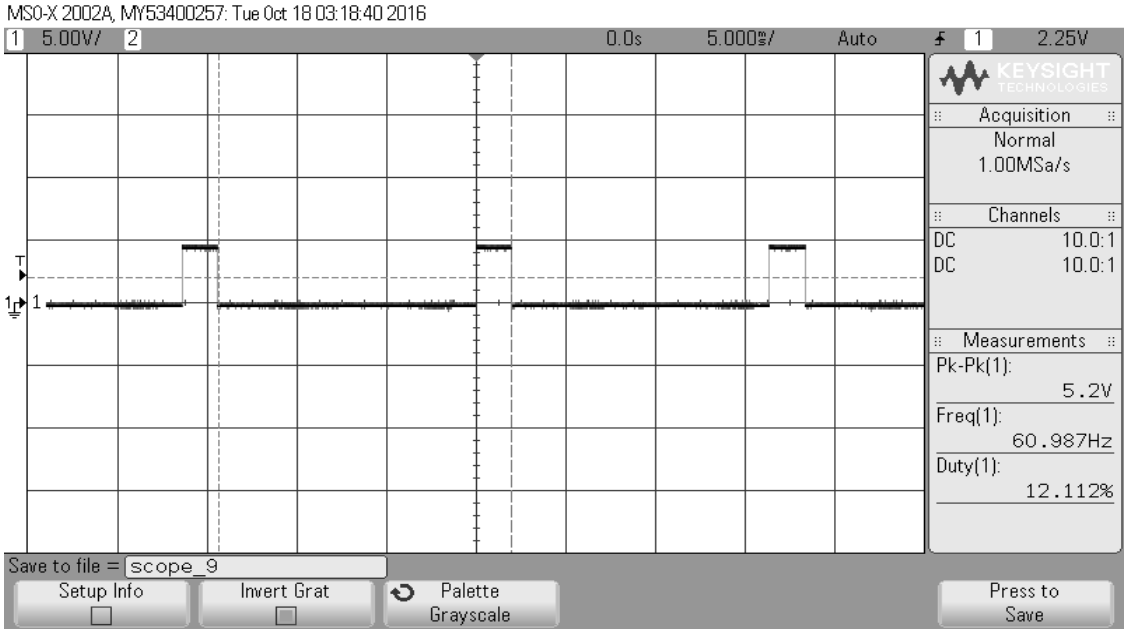

Figura 23: *Duty cycle* mínimo do PWM de ação de controle.

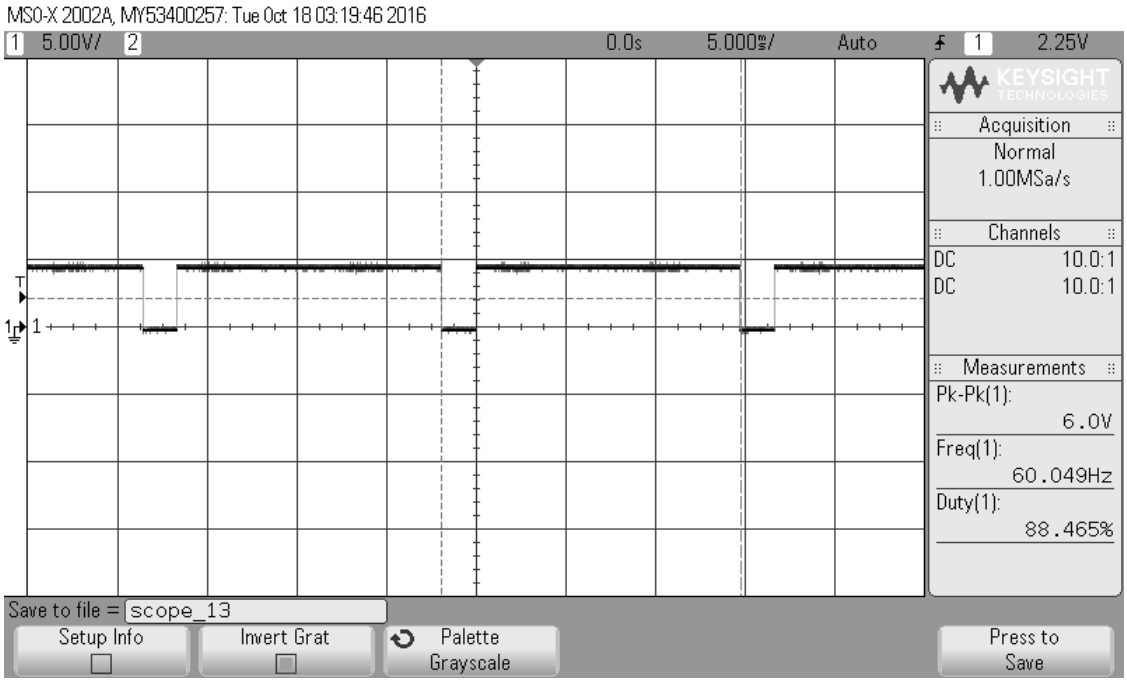

Figura 24: *Duty cycle* máximo do PWM de ação de controle.

Foram realizados testes com os ESCs conectados e as hélices removidas com o intuito de se inspecionar os valores de calibração do ESC, assim utilizando a central para realizar o envio de comandos criou-se rotinas para a análise de rotação dos motores utilizando o Arduino de recepção e conectando diretamente o sinal proveniente do Arduino ao ESC. Observou-se que os motores produziram rotação apenas para o caso em que a calibração do ESC atingiu a variação de *duty cycle* de 10% à 90% aproximadamente. O processo de calibração ocorre inicialmente por manter aproximadamente 3 segundos o sinal em *duty cycle* de 10% e posteriormente mantendo-se o sinal de *duty cycle* em 90% por aproximadamente 3 segundos. Pode-se observar os sinais necessários para calibração do ESC nas Figuras 25 e 26.

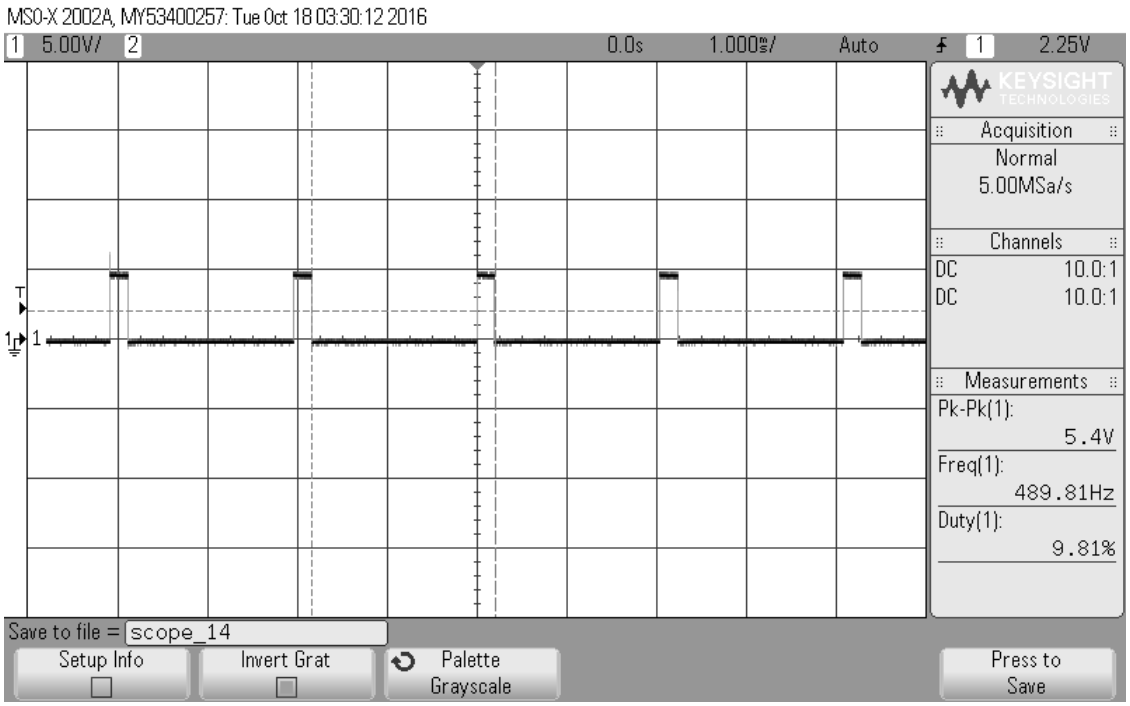

Figura 25: *Duty cycle* mínimo para devida calibração do ESC.

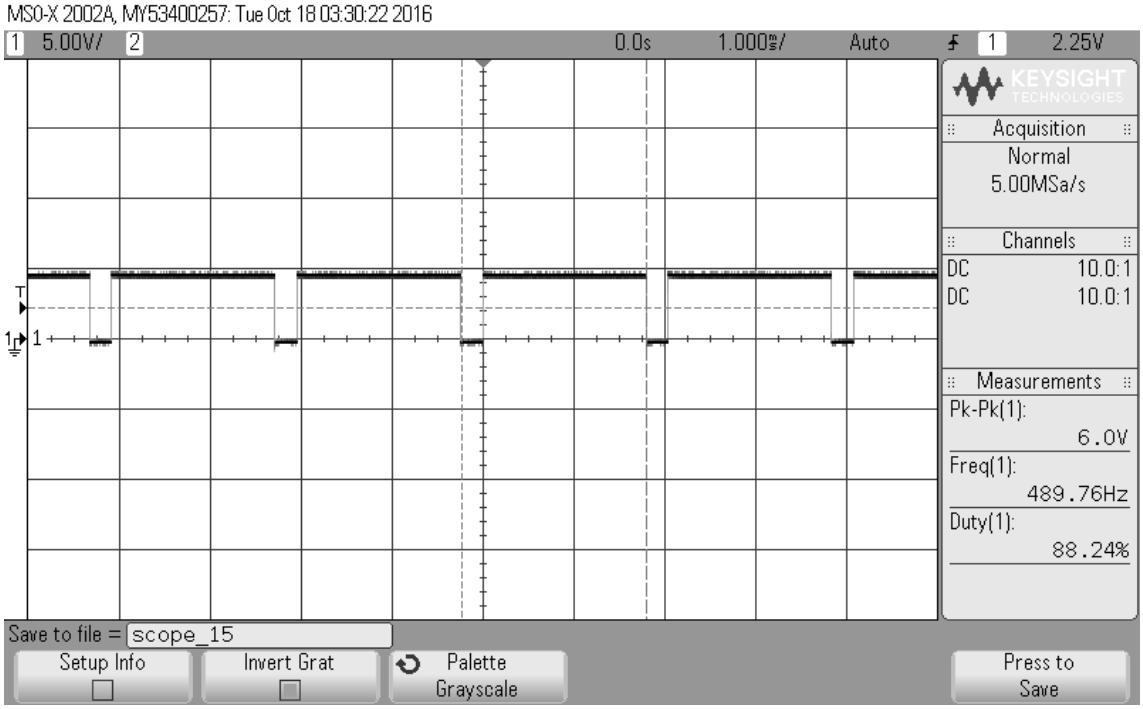

Figura 26: *Duty cycle* máximo para devida calibração do ESC.

Realizou-se de forma semelhante medidas dos PWMs provenientes do software de navegação inseridos nos ESCs que efetuam a rotação dos motores, com o intuito de constatar a frequência e os *duty cycles* mínimos e máximos, que podem ser observados nas Figuras 25 e 26. O *duty cycle* mínimo girou em torno de 10,40% enquanto que o máximo *duty cycle* girou em torno de 60,20%. Assim constatando uma variação de *duty cycle* insuficiente para a calibração do ESC, ocasionando os motores a ficarem parados. Observou-se que o software de navegação possui de fato uma limitação na variação de PWM, que pode ser encontrado no arquivo "controle_motores.c", mencionado no Capítulo 3 – item 3.2 Métodos, onde a função que ajusta a velocidade é limitada em valores de 0 à 100, assim proporcionando que os valores de contagem para o timer do PWM de rotação dos motores varie de 180 à 80, correspondente respectivamente aos valores de *duty cycle* de 10% e 60%.

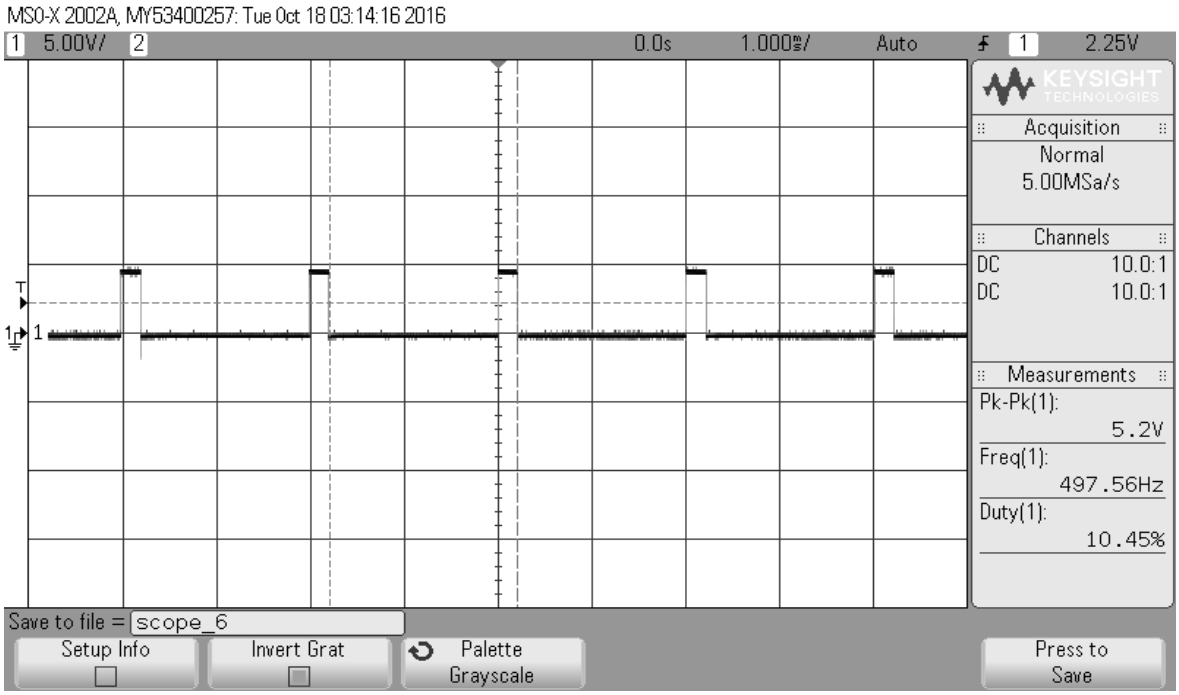

Figura 27: *Duty cycle* mínimo do PWM de rotação.

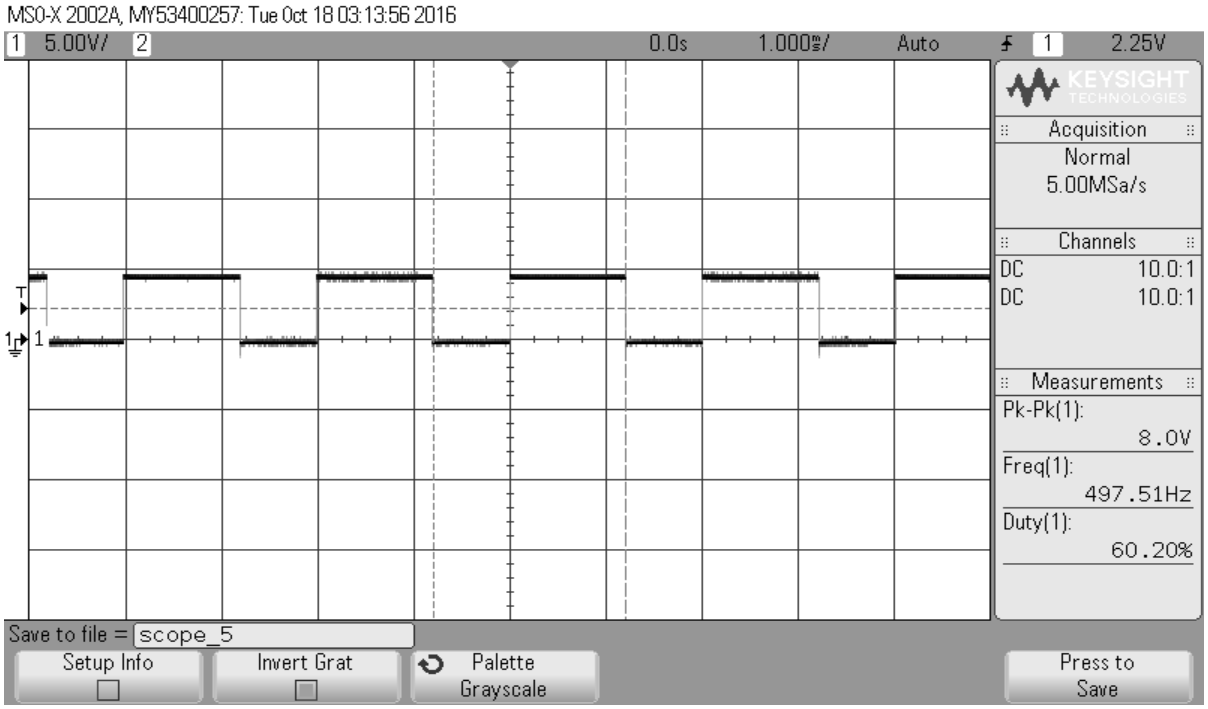

Figura 28: *Duty cycle* máximo do PWM de rotação.

Foi escolhido arbitrariamente sinais de ação de controle que foram aplicados no drone com o objetivo de se obter os sinais que cada ESC receberia do software de navegação, assim realizou-se as medidas tanto dos sinais de ação de controle, quanto as saídas geradas pelo software de navegação. As Figuras 29, 30, 31 e 32 apresentam respectivamente a ação de controle de velocidade média, *pitch*, *roll*, *yaw*.

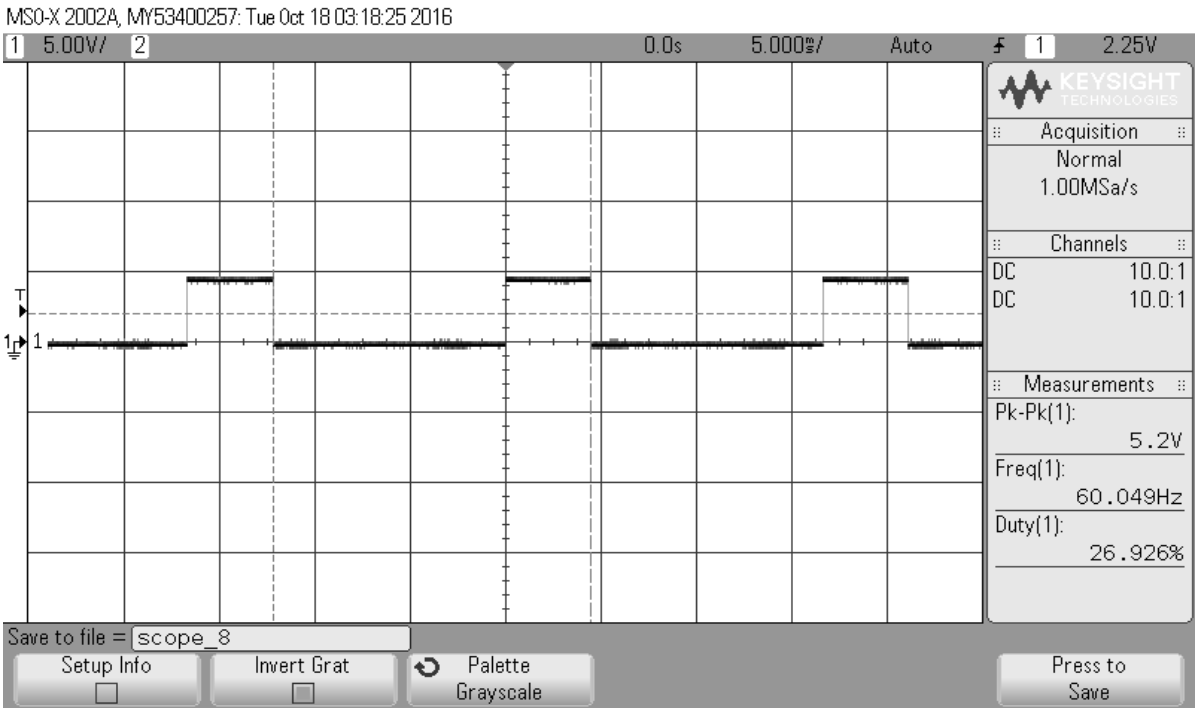

Figura 29: Ação de controle de velocidade média.

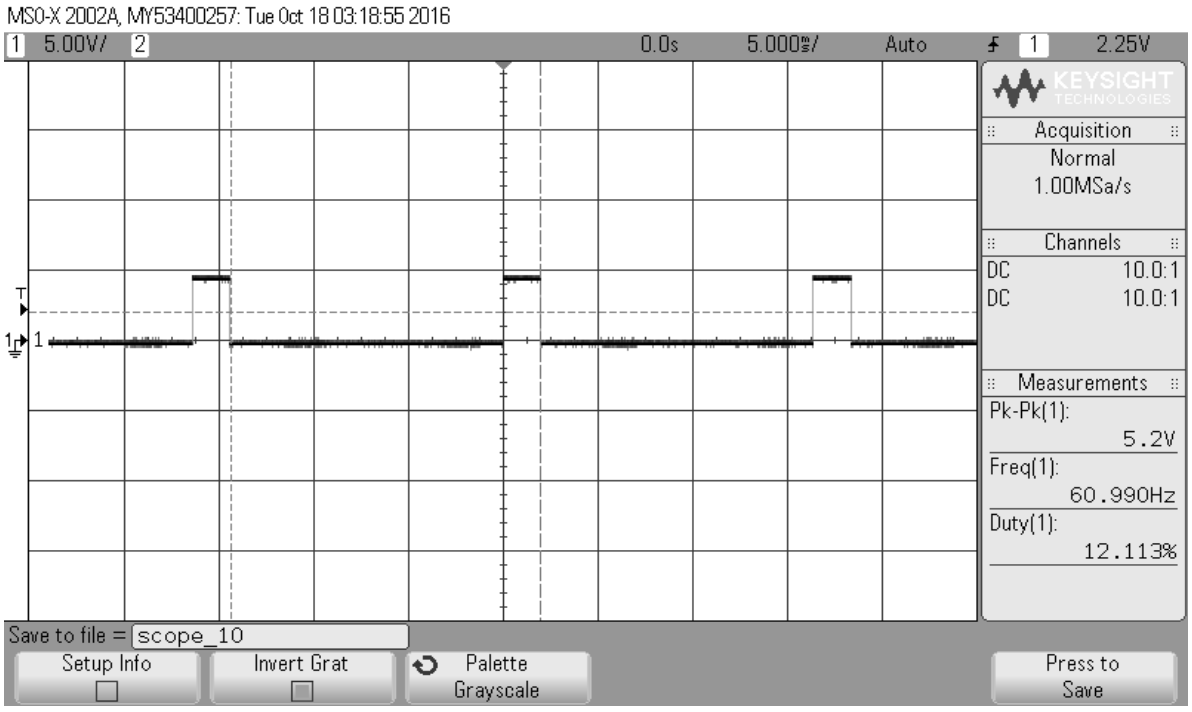

Figura 30: Ação de controle de *pitch*.

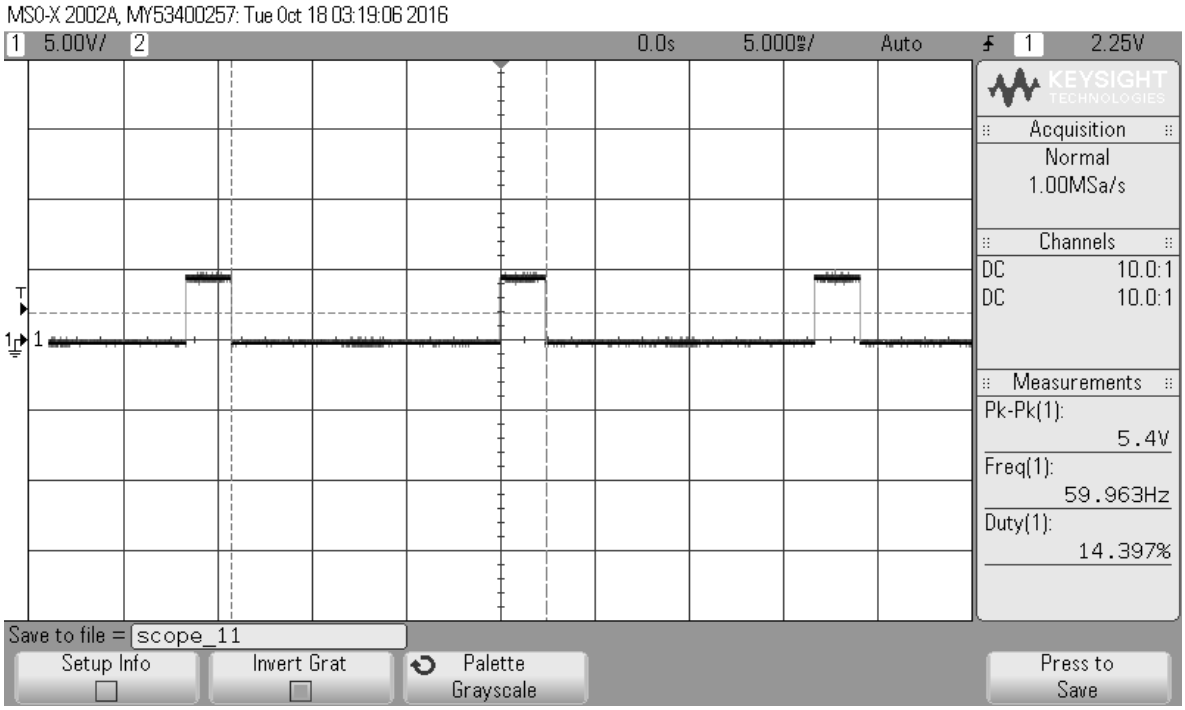

Figura 31: Ação de controle de *roll*.

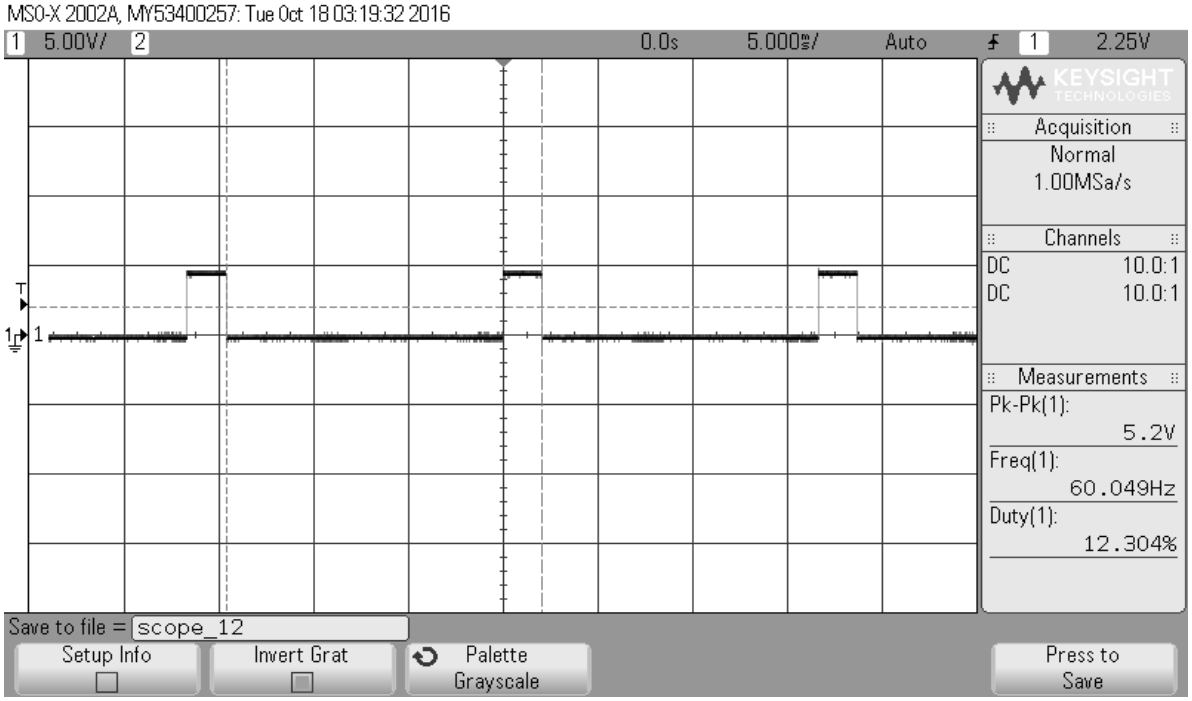

Figura 32: Ação de controle de *yaw*.

Observando-se as Figuras de 29 à 32 pode-se medir os valores iniciais de saída para cada motor uma vez que ele esteja na horizontal, pois após atingir os ângulos de inclinação o sistema de controle utilizado no software de navegação tendera ao “equilíbrio” de rotação dos motores. As Figuras de 33 à 36 apresentam respectivamente os PWMs de rotação dos motores de 1 à 4.

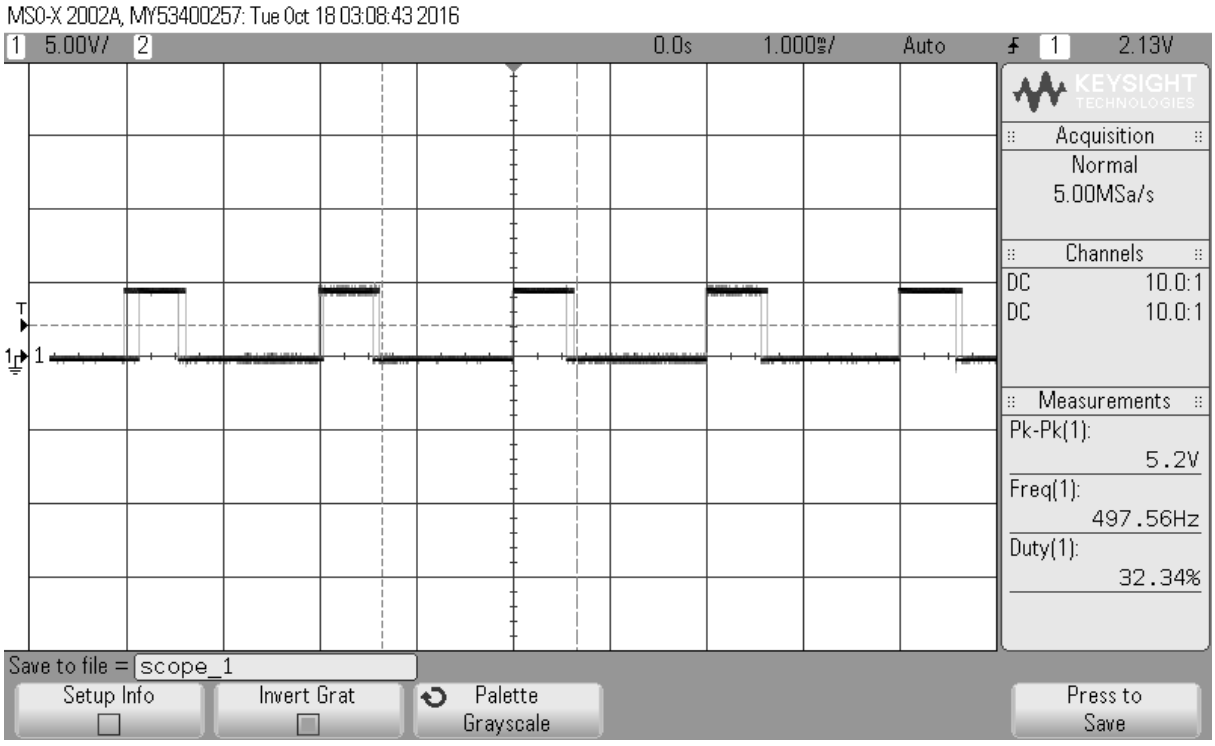

Figura 33: *Duty cycle* de rotação do motor 1.

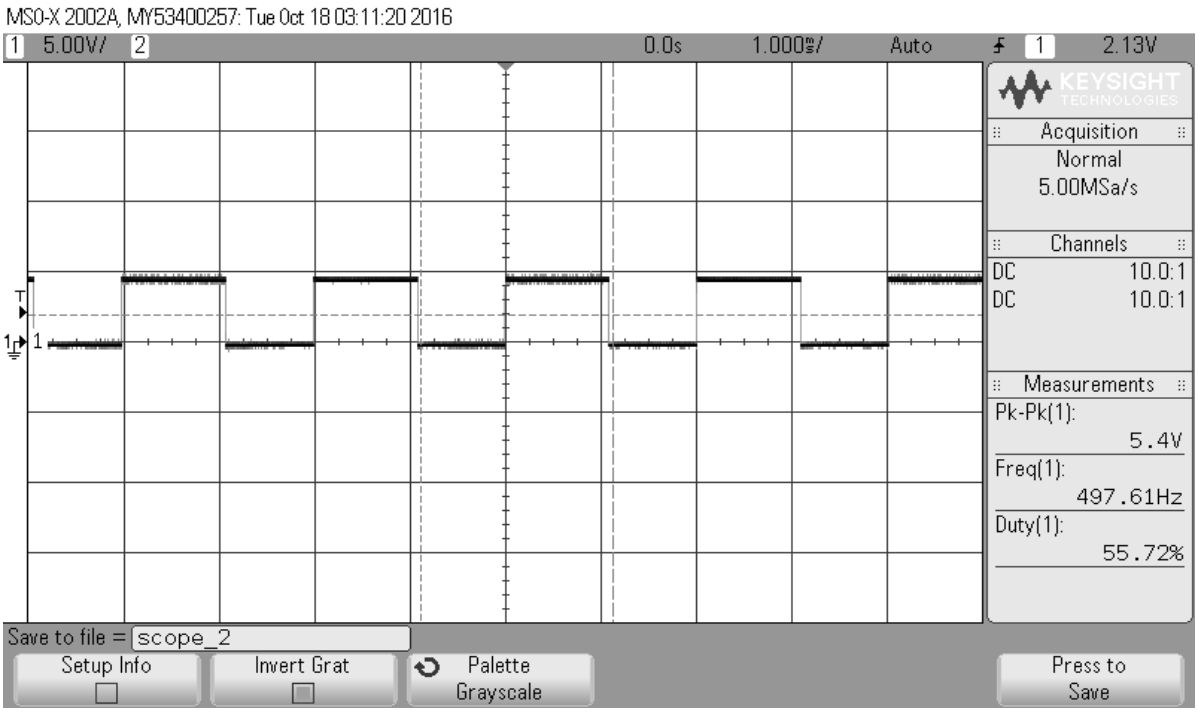

Figura 34: *Duty cycle* de rotação do motor 2.

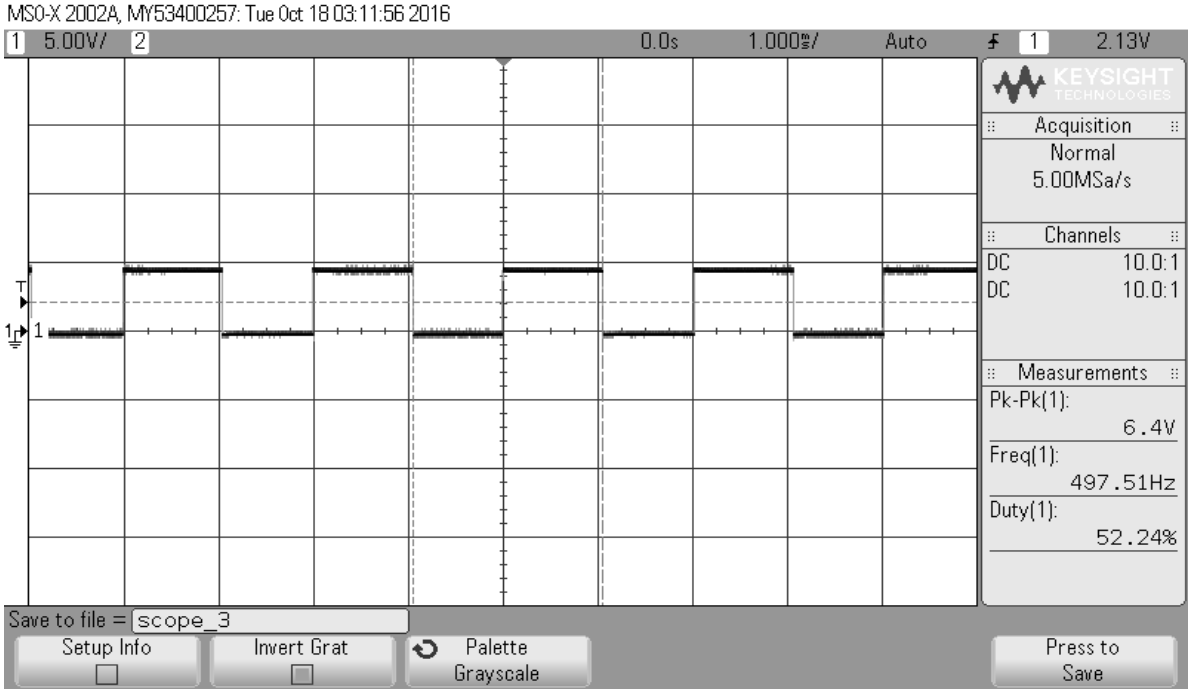

Figura 34: *Duty cycle* de rotação do motor 3.

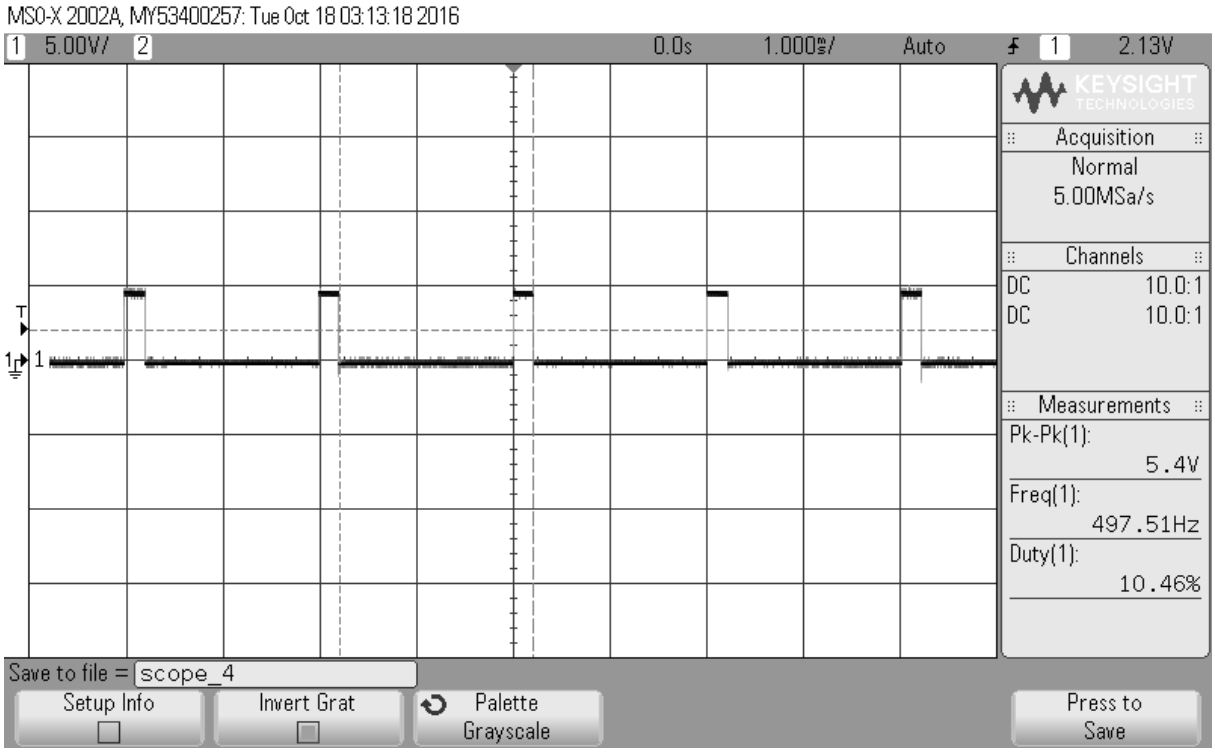

Figura 36: *Duty cycle* de rotação do motor 4.

Observando as Figuras de 29 à 32 espera-se que a inclinação do drone seja máxima de *pitch* e *roll*, e rotação *yaw* anti-horária, que é constatada observando as Figuras de 33 à 36, com referência baseada na Figura 5.

O programa desenvolvido para envio utiliza 7,456 bytes de espaço de armazenamento (correspondente a 23% do total) e utiliza 252 bytes de memória dinâmica (12% do total). Já o desenvolvido para recepção utiliza 9,330 bytes de espaço de armazenamento (28% do total) e 274 bytes de memória dinâmica (13%). Portanto a plataforma Arduino Uno selecionada para essas funções trabalha com folgas e poderia receber expansões de códigos mais complexos de envio/recepção.

O *webserver* é composto de dois processos no sistema operacional, o primeiro com um consumo desprezível de processamento e 3.1% de memória, 17192KB de memória virtual e 14032KB de memória utilizada, o segundo processo consome 5,1% de processamento e 3,3% de

memória, 27772KB de memória virtual e 15056KB de memória utilizada. Isso também indica um baixo consumo de recursos da plataforma Raspberry Pi.

É importante ressaltar também problemas relacionados ao tempo de resposta da internet, mesmo sem ter realizado medidas para este caso pode-se fazer uma análise crítica, pois durante os testes se deparou com a problemas de latência da internet, assim algumas das vezes os comandos levaram um tempo maior do que outros comandos a serem efetuados. Deste modo, ocorreram falhas de perda de pacotes quando não se obteve resposta após um tempo longo de espera. Em outros casos o usuário reenviou comandos e os dois comandos foram executados em um curto período de tempo, em consequência do reenvio.

CAPÍTULO 5

5. Conclusão

A plataforma de testes apresentou desempenho satisfatório, realizando os comandos e produzindo informações que foram capazes de demonstrar o comportamento do drone e as características que o mesmo apresenta, com relação aos componentes utilizados e as características apresentadas pelo software de navegação. Pode-se ressaltar que a plataforma serviu não só para uma análise de drones, mas também de uma forma individual para os componentes do veículo aéreo.

Obteve-se um ganho de experiência na escolha do conjunto motor e ESC para drones, pois os que foram utilizados possuem características diferentes do implementado no software de navegação utilizado, o que acarretou um estudo detalhado sobre os mesmos para determinação de suas características consumindo parte do tempo empregado no projeto.

Ao longo do projeto observou-se que a biblioteca utilizada para comunicação RF apresenta problemas, na qual possui rotinas de inicialização de transmissão e recepção com erros, portanto uma melhora no projeto seria ao invés de se utilizar a biblioteca disponibilizada no site de suporte, reescrever a própria biblioteca da nRF24L01, e também escrever uma biblioteca para a Raspberry Pi, pois a plataforma possui comunicação SPI a mesma utilizada pelo modulo nRF24L01, o que permite uma integração direta entre o *webserver* e o modulo transmissor/receptor, o que excluiria o Arduino Uno que foi utilizado para essa integração. Com isso possibilita uma telemetria que poderá ser implementada no *webserver*.

Observou-se que o método escolhido para teste do VANT é impraticável para comanda-lo, devido à internet não ser preditiva e devido ao software de navegação não possuir métodos de

frenagem, pois uma vez que os ângulos são setados o VANT ganha velocidade e com isso possuirá um momento linear que permaneceria mesmo que os ângulos sejam setados para que o VANT paire sobre o ar. E mesmo que tente-se opor o movimento com ângulos opostos não pode-se garantir uma devida frenagem, então para que o VANT termine a ação de movimento seria necessário um algoritmo de controle no software de navegação que realize a frenagem.

5.1. Trabalhos Futuros

Uma outra possibilidade para a futura implementação autônoma seria alterar o método de comandar o VANT, ao invés de se passar ações de controle para setar as inclinações do drone, implementar um software de navegação mais complexo, que seja capaz de receber posições utilizando dados do GPS, assim o drone não possuiria dependência direta de movimentos com a central de controle, assim evitando problemas com a latência da internet.

Portanto, o rumo do projeto tem foco destinado no desenvolvimento de um software de navegação que seja capaz de receber dados de posições pela central de controle e que utilize um GPS para tal. Desenvolvimento de algoritmos no software para que o mesmo possua um controle de frenagem. Integração da central com o modulo RF e reestruturação do *webserver* com objetivo de se utilizar o sistema de posicionamento global, por exemplo utilizando a API do google maps.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] TICE, Brian P. **Unmanned Aerial Vehicles – The Force Multiplier of the 1990s**. p. 1991. (em inglês) (*When used, UAVs should generally perform missions characterized by three Ds: dull, dirty, and dangerous.).* Disponível em: <http://archive.is/7Jk1F>. Acesso 3 de junho de 2016.

[2] BASTA, O. P. **Quad Copter Flight**. California State University, Northridge. [S.1.], p. 2. 2012. CMSIS – Cortex Microcontroller Software Interface Standard ARM. Disponível em: <http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php>. Acesso em: 4 de junho de 2016.

[3] CALLEGARO, B. F. M. **Desenvolvimento de um Veículo Aéreo quadrirotor com sistema de estabilização baseado no filtro de Kalman**. Escola de Engenharia de São Carlos – USP. São Carlos, p. 19, 2014.

[4] USP. <http://www5.usp.br/31918/uso-de-drones-na-agricultura-de-precisao-e-tema-de-estudo-na-esalq/>. Acesso em: 3 de junho de 2016.

[5] Web Archive. "Web Application framework". <http://web.archive.org/web/20150723163302/http://docforge.com/wiki/Web_application_framework>. Acesso em: 5 de junho de 2016.

[6] W3schools. <http://www.w3schools.com/aspnet/mvc_intro.asp>. Acesso em: 6 de junho de 2016.

[7] LEISHMAN, J. G. A History of Helicopter Flight. Unicersity of Maryland. <http://terpconnect.umd.edu/~leishman/Aero/history.html>. Acesso em: 6 de julho de 2016.

[8] CALLEGARO, B. F. M. **Desenvolvimento de um Veículo Aéreo quadrirotor com sistema de estabilização baseado no filtro de Kalman**. Escola de Engenharia de São Carlos – USP. São Carlos, 2014.

[9] NORDIC SEMICONDUCTOR. **Single chip 2.4GHz Transceiver**. Nordic Semiconductor ASA. Tiller, Noruega, 2004.

[10] Elecfreaks. <http://www.elecfreaks.com/wiki/index.php?title=2.4G_Wireless_nRF24L01p>. Acesso em: 20 de setembro de 2016.

[11] VIEIRA, M. A. **Comunicação via RF**. UNIVASF – Universidade do Vale do São Fransisco.

Disponível em: <http://www.univasf.edu.br/~gari/futvasf/paginas/download/Apresenta%C3%A7%C3%A3oRF_Manoel%2009-04-2010.pdf>. Acesso em: 25 de setembro de 2016.

[12] JOAQUIM M. B. **Modulação em Amplitude**. Escola de Engenharia de São Carlos – USP. São Carlos. Disponível em: <http://www1.sel.eesc.usp.br:8085/upload/sel360/sel360-3_am_I_sel0360_6.pdf>. Acesso em: 21 de outubro de 2016.

[13] MALBURG M. M. **Trabalho Final de Redes I**. Departamento de Engenharia Eletrônica – UFRJ. Rio de Janeiro, 2004. Disponível em: <http://www.gta.ufrj.br/grad/04_2/Modulacao/>. Acesso em: 21 de outubro de 2016.

[14] GEREZ S. H. **Implementation of Digital Signal Processing: Some Background on GFSK Modulation**. Departamento de Engenharia Elétrica - Universidade de Twente. 2016. Disponível em: <http://wwwhome.ewi.utwente.nl/~gerezsh/sendfile/sendfile.php/gfsk-intro.pdf?sendfile=gfsk-intro.pdf>. Acesso em: 22 de outubro de 2016.

ANEXO 1

Anexo 1

Os Códigos apresentados abaixo foram utilizados respectivamente como servidor web, templates HTML, rotinas de transmissão RF e rotinas auxiliares de teste para recepção de sinal RF.

Webserver.py

```
from flask import Flask, render_template, request
import serial
import time

ser = serial.Serial('/dev/ttyACM0', 57600)

@app.route('/'):
def index():
        return render_template('index.html')

@app.route('/<move>')
def proj(move):
        if move == "on":
                ser.write('8')
        if move == "off":
                ser.write('7')
        if move == "up":
                ser.write('6')
        if move == "down":
                ser.write('b')
```

```
        if move == "forward":
                ser.write('0')
        if move == "backward":
                ser.write('3')
        if move == "left":
                ser.write('4')
        if move == "right":
                ser.write('9')
        if move == "cw":
                ser.write('1')
        if move == "ccw":
                ser.write('2')
        if move == "app":
                ser.write('x')
        if move == "project":
                ser.write('A')

        return render_template('project.html')

if __name__ == '__main__':
        app.run(debug=True, host='project.html')
```

Index.html

```
<html lang="en">
       <head>
               <link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
/>
               <meta charset="utf-8">
               <meta name="viewport" content="width=device-width, initial-
scale=1">
               <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></scr
ipt>
               <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></s
cript>
```

```
	</head>
	<body>
		<div class="jumbotron text-center">
			<h1>   Controle de Drones Via Web  </h1>
			<p> Para acessar os comandos clique no botão
"Controle"  </p>
		</div>
		<div class="container">
			<div class="row">
				<div class="col-sm-4">
				</div>
				<div class="col-sm-4">
					<div class="jumbotron text-center">
						<input type="submit"
class="btn btn-success" value="Controle"
onclick="location.href='http://143.107.235.55:5000/project'">
					</div>
				<div class="col-sm-4">
				</div>
			</div>
		</div>
	</body>
</html>
```

Project.html

```
<html lang="en">
	<head>
		<meta charset="utf-8">
		<meta name="viewport" content="width=device-width,
initial-scale=1">
		<link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min
.css">
```

```
		<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js
"></script>
		<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.
js"></script>
	</head>

	<body>
		<div class="jumbotron text-center">
			<h1> Comandos Disponiveis </h1>
		</div>
		<div class="container">
			<div class="row">
				<div class="col-sm-4">
					<div class="jumbotron text-
center">
						<div class="btn-group-
vertical">
							<input
type="submit" class="btn btn-success" value="Turn ON"
onclick="location.href='http://143.107.235.55:5000/on'">
							<input
type="submit" class="btn btn-danger" value="Turn OFF"
onclick="location.href='http://143.107.235.55:5000/off'">
						</div>
					</div>
				</div>
				<div class="col-sm-4">
					<div class="jumbotron text-
center">
						<div class="btn-group-
vertical">
```

```
                                                        <input
type="submit" class="btn btn-primary" value="Up"
onclick="location.href='http://143.107.235.55:5000/up'">
                                                        <input
type="submit" class="btn btn-primary" value="Down"
onclick="location.href='http://143.107.235.55:5000/down'">
                                                    </div>
                                            </div>
                                    </div>
                                    <div class="col-sm-4">
                                            <div class="jumbotron text-
center">
                                                    <div class="btn-group-
vertical">
                                                        <input
type="submit" class="btn btn-info" value="Forward"
onclick="location.href='http://143.107.235.55:5000/forward'">
                                                        <input
type="submit" class="btn btn-info" value="Backward"
onclick="location.href='http://143.107.235.55:5000/backward'">
                                                    </div>
                                            </div>
                                    </div>
                            </div>
                            <div class="row">
                            </div>
                            <div class="row">
                                    <div class="col-sm-4">
                                            <div class="jumbotron text-
left">
                                                    <h5 class="text-
danger"> Os comandos estao divididos em classes de movimentos </h5>
                                                    <h6 class="text-muted">
"Turn ON": configura o drone para voo </h6>
```

```
                                                <h6 class="text-muted">
"Turn OFF": desliga os motores </h6>
                                        </div>
                                </div>
                                <div class="col-sm-4">
                                        <div class ="jumbotron text-
center">
                                                <div class="btn-group-
vertical">
                                                        <input
type="submit" class="btn btn-primary" value="cw"
onclick="location.href='http://143.107.235.55:5000/cw'">
                                                        <input
type="submit" class="btn btn-primary" value="ccw"
onclick="location.href='http://143.107.235.55:5000/ccw'">
                                                </div>
                                        </div>
                                </div>
                                <div class="col-sm-4">
                                        <div class="jumbotron text-
center">
                                                <div class="btn-group">
                                                        <input
type="submit" class="btn btn-info" value="Left"
onclick="location.href='http://143.107.235.55:5000/left'">
                                                        <input
type="submit" class="btn btn-info" value="Right"
onclick="location.href='http://143.107.235.55:5000/right'">
                                                </div>
                                        </div>
                                </div>
                        </div>
                        <div class="row">
                                <div class="col-sm-4">
```

```
                                        <div class="jumbotron text-
left">
                                                <h6 class="text-muted">
"Up" - "Down": movimento vertical </h6>
                                                <h6 class="text-muted">
"Forward" - "Backward": movimento horizontal avanço e retorno </h6>
                                                <h6 class="text-muted">
"Right" - "Left": movimento horizontal direita e esquerda </h6>
                                        </div>
                                </div>
                                <div class="col-sm-4">
                                        <div class="jumbotron text-
center">
                                                <div class="btn-group">
                                                        <input
type="submit" class="btn btn-success" value="Apply"
onclick="location.href='http://143.107.235.55:5000/app'">
                                                </div>
                                        </div>
                                </div>
                                <div class="col-sm-4">
                                        <div class="jumbotron text-
muted">
                                                <h6 class="text-muted">
"cw" - "ccw": giro horário e anti-horário </h6>
                                                <h6 class="text-muted">
"apply": aplica a configuracao escolhida </h6>
                                        </div>
                                </div>
                        </div>
                        <div class="row">
                        </div>
                        <div class="row">
                                <div class="col-sm-4">
                                </div>
```

```
                                <div class="col-sm-4">
                                        <div class="jumbotron text-
center">
                                                <input type="submit"
class="btn btn-warning" value="Home"
onclick="location.href='http://143.107.235.55:5000/'">
                                        </div>
                                <div class="col-sm-4">
                                <div class="col-sm-4">
                                </div>
                        </div>
                </div>
        </body>
</html>
```

Send.ino

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

// Set up nRF24L01 radio on SPI bus plus pins 7 & 8

RF24 radio(7,8);

// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = { 0x1011121314LL, 0x1516171819LL };

char valor;

void setup(void)
{
  Serial.begin(57600);
  Serial.print("\n\rRF24/examples/pingpair/\n\r");
```

```
  // Setup and configure rf radio

  radio.begin();

  // optionally, increase the delay between retries & # of retries
  radio.setRetries(15,15);

  //POWER UP
  radio.powerUp();

  // optionally, reduce the payload size.  seems to
  // improve reliability
  radio.setPayloadSize(8);
  // Open pipes to other nodes for communication

  radio.openWritingPipe(pipes[0]);

  // Start listening

  radio.startListening();

  // Dump the configuration of the rf unit for debugging
  radio.printDetails();
}

void loop(void)
{
  if (Serial.available() > 0)
  {
    valor = Serial.read();
     // First, stop listening so we can talk.
    radio.stopListening();

    // Take the time, and send it.  This will block until complete
    char val = valor;
```

```
    //Serial.print("Now sending... ");
    Serial.println(val);
    bool ok = radio.write( &val, sizeof(char) );

    radio.startListening();

    delay(50);
  }
}
```

Receive.ino

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "PWM.h"

// Set up nRF24L01 radio on SPI bus plus pins 7 & 8

RF24 radio(7,8);

// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = { 0x1011121314LL, 0x1516171819LL };

//Escalator
int lad_d = 20;
int lad_d1 = 30;
int lad_d2 = 30;
int lad_d3 = 200;

void setup(void)
{

  Serial.begin(57600);

  // Setup and configure rf radio
```

```
radio.begin();

// optionally, increase the delay between retries & # of retries
radio.setRetries(15,15);

//POWER UP
radio.powerUp();

// optionally, reduce the payload size.  seems to
// improve reliability
radio.setPayloadSize(8);

// Open pipes to other nodes for communication

radio.openReadingPipe(1,pipes[0]);

radio.startListening();

// Dump the configuration of the rf unit for debugging

radio.printDetails();

InitTimersSafe();

bool success = SetPinFrequency(3,60);
bool success1 = SetPinFrequency(5,60);
bool success2 = SetPinFrequency(6,60);
bool success3 = SetPinFrequency(10,60);

//PWM pins

int Med = 3;
int Row = 5;
int Pit = 6;
int Yaw = 10;

pinMode(Med,OUTPUT);
pinMode(Row,OUTPUT);
```

```
  pinMode(Pit,OUTPUT);
  pinMode(Yaw,OUTPUT);

  analogWrite(3,65);
  analogWrite(5,126);
  analogWrite(6,126);
  analogWrite(10,1025);

}

void loop(void)
{

    if ( radio.available() )
    {
      // Dump the payloads until we've gotten everything
      char got_value;
      bool done = false;
      while (!done)
      {
        // Fetch the payload, and see if this was the last one.
        done = radio.read( &got_value, sizeof(char) );

        // Spew it
        Serial.print("Got payload... ");
        Serial.println(got_value);
        pwm_write(got_value,lad_d,lad_d1,lad_d2,lad_d3);
      }
    }
}

void pwm_write(char value, int lad, int lad1, int lad2, int lad3)
{
  //on
  if (value == '8')
  {
    analogWrite(3,16);
    analogWrite(5,126);
```

```
    analogWrite(6,126);
    analogWrite(10,1025);

    delay(250);

    analogWrite(3,115);
    analogWrite(5,126);
    analogWrite(6,126);
    analogWrite(10,1035);

    delay(250);

    analogWrite(3,65);
    analogWrite(5,126);
    analogWrite(6,126);
    analogWrite(10,1035);
  }

  //off
  if (value == '7')
  {
    analogWrite(3,16);
    analogWrite(5,26);
    analogWrite(6,26);
    analogWrite(10,260);
  }

  //up
  if (value == '6')
  {
    //LAD

//_____________________________________________________________________________
    if (lad > 110)
    {
      lad = 115;
      lad_d = lad;
    }
```

```
    if (lad <= 110)
    {
      lad = lad + 5;
      lad_d = lad;
    }
  }

  //down
  if (value == 'b')
  {
    //LAD

//_____________________________________________________________________________
    if (lad < 16)
    {
      lad = 15;
      lad_d = lad;
    }
    if (lad >= 20)
    {
      lad = lad - 5;
      lad_d = lad;
    }
  }

  //forward
  if (value == '0')
  {
    //LAD2

//_____________________________________________________________________________
    if (lad2 > 220)
    {
      lad2 = 225;
      lad_d2 = lad2;
    }
    if (lad2 <= 220)
    {
```

```
      lad2 = lad2 + 5;
      lad_d2 = lad2;
    }
  }

  //backward
  if (value == '3')
  {
    //LAD2

//_____________________________________________________________________________
    if (lad2 < 26)
    {
      lad2 = 25;
      lad_d2 = lad2;
    }
    if (lad2 >= 30)
    {
      lad2 = lad2 - 5;
      lad_d2 = lad2;
    }
  }

  //left
  if (value == '4')
  {
    //LAD1

//_____________________________________________________________________________
    if (lad1 < 26)
    {
      lad1 = 25;
      lad_d1 = lad1;
    }
    if (lad1 >= 30)
    {
      lad1 = lad1 - 5;
      lad_d1 = lad1;
```

```
    }
  }

  //right
  if (value == '9')
  {
    //LAD1

//_____________________________________________________________________________
    if (lad1 > 220)
    {
      lad1 = 225;
      lad_d1 = lad1;
    }
    if (lad1 <= 220)
    {
      lad1 = lad1 + 5;
      lad_d1 = lad1;
    }
  }

  //cw
  if (value == '1')
  {
    //LAD3

//_____________________________________________________________________________
    if (lad3 > 1830)
    {
      lad3 = 1850;
      lad_d3 = lad3;
    }
    if (lad3 <= 1830)
    {
      lad3 = lad3 + 50;
      lad_d3 = lad3;
    }
  }
```

```
  //ccw
  if (value == '2')
  {
    //LAD3

//_____________________________________________________________________________
    if (lad3 < 260)
    {
      lad3 = 250;
      lad_d3 = lad3;
    }
    if (lad3 >= 260)
    {
      lad3 = lad3 - 50;
      lad_d3 = lad3;
    }
  }

  //apply
  if (value == 'x')
  {
    Serial.println(lad);
    Serial.println(lad1);
    Serial.println(lad2);
    Serial.println(lad3);

    analogWrite(3,lad);
    analogWrite(5,lad1);
    analogWrite(6,lad2);
    analogWrite(10,lad3);
  }
}
```

Motor_test.ino

```
#include <SPI.h>
#include "nRF24L01.h"
```

```
#include "RF24.h"

// Set up nRF24L01 radio on SPI bus plus pins 7 & 8

RF24 radio(7,8);

// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = { 0x1011121314LL, 0x1516171819LL };

void setup(void)
{

  Serial.begin(57600);

  // Setup and configure rf radio

  radio.begin();

  // optionally, increase the delay between retries & # of retries
  radio.setRetries(15,15);

  //POWER UP
  radio.powerUp();

  // optionally, reduce the payload size.  seems to
  // improve reliability
  radio.setPayloadSize(8);

  // Open pipes to other nodes for communication

  radio.openReadingPipe(1,pipes[0]);

  radio.startListening();

  // Dump the configuration of the rf unit for debugging

  radio.printDetails();
```

```
  //PWM pins

  int Med = 3;
  int Row = 5;
  int Pit = 6;
  int Yaw = 10;

  pinMode(Med,OUTPUT);
  pinMode(Row,OUTPUT);
  pinMode(Pit,OUTPUT);
  pinMode(Yaw,OUTPUT);

  analogWrite(3,126);
  analogWrite(5,126);
  analogWrite(6,126);
  analogWrite(10,126);

}

void loop(void)
{

    if ( radio.available() )
    {
      // Dump the payloads until we've gotten everything
      char got_value;
      bool done = false;
      while (!done)
      {
        // Fetch the payload, and see if this was the last one.
        done = radio.read( &got_value, sizeof(char) );

        // Spew it
        Serial.print("Got payload... ");
        Serial.println(got_value);
        pwm_write(got_value);
      }
    }
```

```
}

void pwm_write(char value)
{
  //on
  if (value == '8')
  {
    analogWrite(3,25);
    analogWrite(5,25);
    analogWrite(6,25);
    analogWrite(10,25);

    delay(3000);

    analogWrite(3,225);
    analogWrite(5,225);
    analogWrite(6,225);
    analogWrite(10,225);

    delay(3000);

    analogWrite(3,126);
    analogWrite(5,126);
    analogWrite(6,126);
    analogWrite(10,126);
  }

  //off
  if (value == '7')
  {
    analogWrite(3,125);
    analogWrite(5,125);
    analogWrite(6,125);
    analogWrite(10,125);
  }

  //up
  if (value == '6')
```

```
{
  analogWrite(3,185);
  analogWrite(5,185);
  analogWrite(6,185);
  analogWrite(10,185);
}

//down
if (value == 'b')
{
  analogWrite(3,135);
  analogWrite(5,135);
  analogWrite(6,135);
  analogWrite(10,135);
}

//forward
if (value == '0')
{
  analogWrite(3,225);
  analogWrite(5,225);
  analogWrite(6,225);
  analogWrite(10,225);
}

//backward
if (value == '3')
{
  analogWrite(3,155);
  analogWrite(5,155);
  analogWrite(6,155);
  analogWrite(10,155);
}

//left
if (value == '4')
{
  analogWrite(3,165);
```

```
    analogWrite(5,165);
    analogWrite(6,165);
    analogWrite(10,165);
  }

  //right
  if (value == '9')
  {
    analogWrite(3,200);
    analogWrite(5,200);
    analogWrite(6,200);
    analogWrite(10,200);
  }

  //cw
  if (value == '1')
  {
    analogWrite(3,126);
    analogWrite(5,126);
    analogWrite(6,126);
    analogWrite(10,126);
  }

  //ccw
  if (value == '2')
  {
    analogWrite(3,126);
    analogWrite(5,126);
    analogWrite(6,126);
    analogWrite(10,126);
  }
}
```