Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

RELATÓRIO FINAL

VANT SEMIAUTÔNOMO UTILIZANDO ARDUINO

Nome do orientador:
Edilson Reis Rodrigues Kato

Nome do bolsista:
Caio Cesar Almeida Pegoraro

Processo: 148806/2015-5
Período de vigência: 01/08/2015 a 31/07/2016
Curso: Engenharia de Computação (Enc)
Período: 9º período

São Carlos - SP, 31 de agosto de 2016

Resumo

O material produzido reúne um conjunto de práticas afim de construir um veículo semi-autônomo (no caso um quadricóptero) de modo com que sejam abordados conceitos gerais de computação e engenharia, utilizando como base o micro controlador Arduino (plataforma que permite o desenvolvimento de aplicações utilizando linguagem em alto nível). As etapas abordadas contemplam os diversos setores que precisam funcionar em conjunto para operar o VANT (veículo aéreo não tripulado), desde o software de controle, um dispositivo emissor, um receptor e o controlador (dos motores). O projeto envolve a interação de tecnologias distintas (linguagens e plataformas diferentes) tal como a necessidade de criar um protocolo de comunicação e o desenvolvimento de uma arquitetura própria. Em complemento, também é abordado a questão de projetos de tempo real, no caso existe a necessidade de garantir que determinadas ações sejam executadas em um dado momento específico. Tomando por referência os veículos aéreos comerciais, a maioria faz uso de um rádio controle com um receptor acoplado a uma placa de controle que é encarregada de controlar os motores de maneira apropriada. No projeto não foram utilizadas tecnologias que fazem esse tipo de controle automático, foram desenvolvidos códigos e rotinas nos micro controladores que se encarregam de receber e processar comandos, além de efetuarem análise de sensores para controle da estabilidade. Acessando o material do projeto cada arquivo corresponde a um componente do hardware: o "Painel de controle"é o cliente desenvolvido em C no Visual Studio responsável por mapear os comandos que serão enviados do VANT, foi criado de maneira que possa controlar de maneira dinâmica, isto é, podemos configurar comandos e rotinas da maneira que for necessário, implementando novos botões e áreas para exibição de dados coletados pelos sensores; o "Emissor primário"corresponde ao módulo conectado ao PC pela entrada USB, esse recebe o comando via serial do cliente e envia pelo sensor de rádio para o VANT, também fica responsável por transferir uma mensagem recebida da aeronave para o cliente; o "Receptor primário"é o primeiro de dois módulos que compõem o quadricóptero, a prioridade do algoritmo é tratar de dados recebidos pelo emissor (em tempo real, não se pode "perder"uma informação), em alguns casos o próprio já fica encarregado de executar a ação (como no controle de um buzzer), se for o caso ele pode repassar o comando para o "Receptor secundário", este trata do controle dos motores tal como executar as rotinas de estabilização do VANT.

Todo material desenvolvido, códigos dos receptores/emissores, do painel de controle e a documentação do projeto estão disponíveis em: github.com/CaioPegoraro/vant-DC-UFSCar. Como não há possibilidade de ilustrar apropriadamente os testes praticados os vídeos serão postados no canal: https://www.youtube.com/channel/UCDFCeWVmeYbNF8uYhXh58YA.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

1

Introdução

1.1 Arduino

A proposta foi a construção de um VANT controlado remotamente, definido formalmente como um RPA (`Remotely-Piloted Aircraft`), utilizando um microcontrolador de baixo custo com foco na aplicação em atividades civis (contemplando uma arquitetura de comunicação entre a aeronave e uma base móvel). A execução envolveu uma série de conhecimentos multidisciplinares (entre disciplinas de computação, física e engenharia).

O projeto foi inteiramente estruturado em cima do microcontrolador Arduino, isto é, todo controle de execução e comunicação é executado por um hardware arduino (menos no caso do cliente que executa em um PC). Ao todo foram utilizadas três Arduinos: dois do tipo "Arduino Uno"(um no emissor primário e outro como receptor secundário), o outro é do tipo "Arduino Mega"(funcionando como receptor secundário).

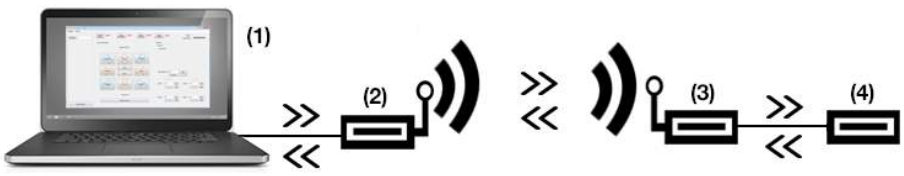

Figura 1.1: Arquitetura do sistema de hardware do VANT.

De acordo com a figura 1.1 temos a arquitetura geral do projeto, os itens enumerados correspondem a cada dispositivo encarregado de uma tarefa específica:

(1) Painel de controle: Um cliente escrito em C executando em um ambiente "Windows", pode tanto exibir quando enviar comandos.

(2): Emissor primário: Uma placa "Arduino Uno"conectada a um componente transceiver, se comunica via USB com o PC para receber e transmitir os comandos para o VANT.

(3) Receptor primário: Outro "Arduino Uno"conectado ao outro transceiver, tem uma rotina que verifica se algum sinal foi recebido e também pode enviar dados de volta ou encaminhar para o receptor secundário.

(4) Receptor secundário: Um "Arduino Mega"que é utilizado para controle de estabilização automática do VANT, pode receber comandos via IC2 do receptor primário para alteração da aceleração dos motores.

Figura 1.2: Hardware do Arduino Uno.

A programação nas duas placas é a mesma (utilizando a linguagem C) e não há diferenças significativas de hardware (considerando o propósito de uso nesse projeto, no geral o MEGA é superior), foram testadas duas alternativas antes de definir esse modelo final: a primeira era utilizar um chip Attiny85 para função de comunicação (o chip daria um processamento dedicado a recepção de dados), como o chip é pequeno seria possível economizar espaço e peso, mas houve incompatibilidade com o sensor de rádio frequência em conjunto com a comunicação IC2 (futuramente será reavaliado essa solução); a segunda tentativa foi utilizar um "Arduino

Figura 1.3: Hardware do Arduino Mega.

Due", principalmente pela característica de processamento paralelo, na prática teriam várias rotinas executando simultaneamente para garantir as tarefas de tempo real, mas novamente houve incompatibilidade com algumas bibliotecas, oque impossibilitou o uso.

1.2 Software e configuração do ambiente

Foram utilizados dois softwares principais para o desenvolvimento do projeto:

Visual Studio 2015

A IDE do VS2015 foi utilizada para o desenvolvimento da interface de comando (em C) que executa no ambiente windows, para estudantes da UFSCar pode ser obtido através do Dreamspark: http://www.dc.ufscar.br/dreamspark.

Arduino IDE 1.6.7

A Arduino IDE foi utilizada para o desenvolvimento dos algoritmos que executam nas três placas arduinos (utilizando C), ela pode ser obtida através do endereço: https://www.arduino.cc/en/Main/Software.

Nesse caso é necessário uma configuração adicional para adicionar as bibliotecas do projeto: IC2 Dev, MPU6050 e RF24-Master (serão abordadas com mais detalhes nos itens a seguir); As bibliotecas também estão presentes no link do repositório do projeto: https://github.com/CaioPegoraro/vant-DC-UFSCar.

Ambos softwares foram executados e um ambiente Windows (Windows 7 versão de 64 bits).

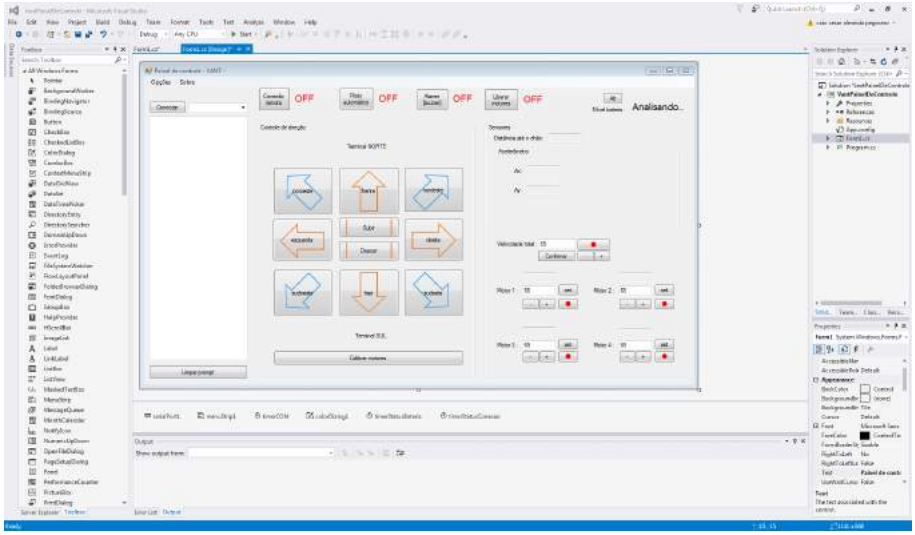

Figura 1.4: Visual Studio 2015.

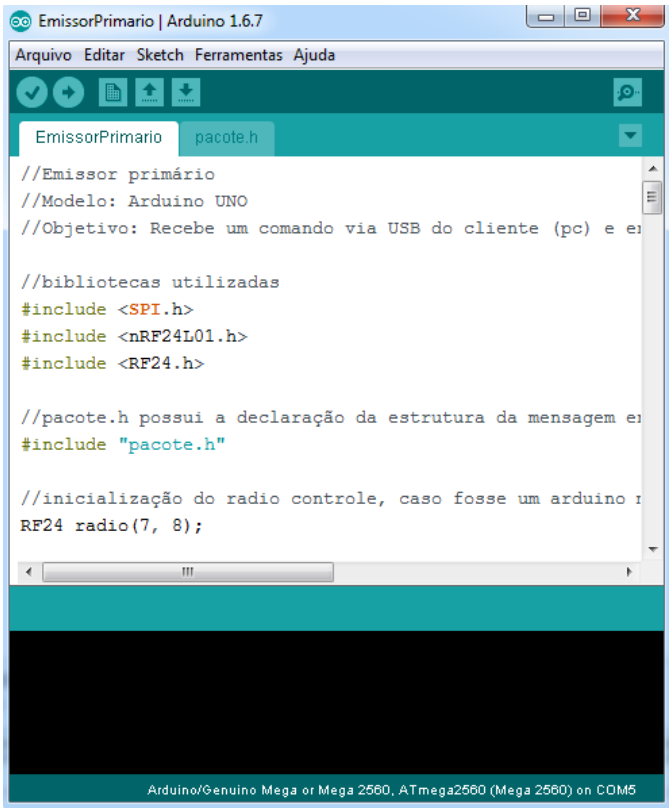

Figura 1.5: Arduino IDE 1.6.7.

1.3 Componentes e periféricos

A lista dos componentes empregados pode ser vista a seguir (o uso, detalhadamento e código de cada parte será abordado no capítulo de resultados e análises):

1. *2x: Módulo transreceptor*: NRF24L01+ Wireless 2,4ghz.

2. *Acelerômetro*: MPU6050.
3. *Alarme*: Buzzer indicador de Tensão 1-8s Lipo.
4. *Regulador de tensão*: Lm2596.
5. *4x: Controlador de velocidade*: ESC 30A.
6. *4x: Motores*: brushless 2212 1000kv.
7. *Carregador*: Lipo 2s e 3s Hobbyking.
8. *Cabo de alimentação*: Xt60 to 4 X 3.5mm Bullet.
9. *Bateria*: Lipo Zippy 2200mah 3s 40/50c 11.1v.
10. *Buzzer*: Oscilador interno.
11. *4x: Hélices*: 1045r 10x4,5.
12. *Bateria*: 9v (alimentação do circuito no VANT).
13. *Componentes gerais*: Resistores, LED's, estanho, placa pcb,tubos termoretráteis, pinos e conectores, fios e chaves.

Figura 1.6: Visão geral dos motores, esc's e hélices.

Como já mencionado anteriormente, além dos sensores foram utilizadas duas versões "Uno"e uma "Mega"do Arduino, além da utilização de ferramentas padrões: Ferro de solda, alicates, régua, fios, etc.

Figura 1.7: Conjunto da bateria, carregador, alarme e transmissor

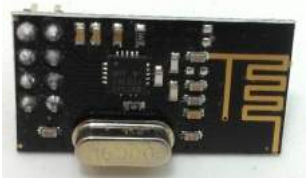

Figura 1.8: Módulo transmissor/receptor RF24.

Figura 1.9: Acelerômetro MPU6050.

Figura 1.10: Alarme para nível da bateria.

Figura 1.11: Regulador de tensão Lm2596 ligado a uma fonte de 12v.

Figura 1.12: Controlador eletronico de velocidade (ESC).

Figura 1.13: Motor brushless 2212.

Figura 1.14: Carregador bateria de Lipo 3s.

Figura 1.15: Cabo de alimentação dos motores.

Figura 1.16: Bateria para os motores, 3s 11v.

Figura 1.17: Hélices de 10’ (polegadas).

Figura 1.18: Pino banana (para fixação da hélice no motor).

Figura 1.19: Buzzer.

1.4 Estrutura metálica

A estrutura do VANT foi construída totalmente em alumínio e fixada por uma série de parafusos; Inicialmente havia a possibilidade de adotar uma estrutura pré-fabricada de plástico ou fibra de carbono, mas isso poderia afetar o real propósito do projeto então foi decidido que a estrutura seria uma montada a partir do zero. O design adotado segue o padrão "H-Cop"em que se utiliza 4 barras retangulares que formam uma figura parecida com a letra "H":

Essa versão apresentava alguns problemas: a fixação feita com rebites não se mostrou tão eficiente e em alguns casos ocasionava uma torção no metal, posteriormente foi substituído por

Figura 1.20: Primeira versão estrutural do VANT.

parafusos (figura 1.21); Outro problema foi no trem de pouso, a versão da figura 1.20 era muito pesado e foi trocado por quatro suportes em cada extremidade da estrutura (será exibido em capítulos posteriores).

Para a construção foram adquiridos quatro tubos de alumínio de 1m de comprimento por 12,7mm de largura, com 0,7mm de espessura:

Figura 1.21: Parafusos utilizados para fixação.

Na parte central (onde foi fixado o hardware da aeronave) e para a fabricação do trem de pouso foi utilizado duas chapas de alumínio com 0,70mm de espessura:

Figura 1.22: Tubos de alumínio utilizados na construção da estrutura.

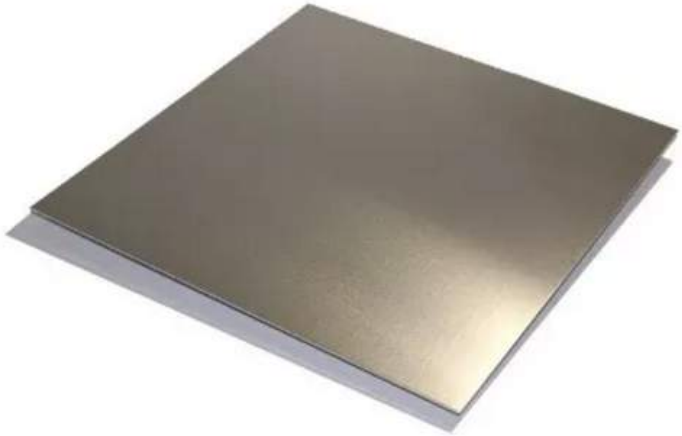

Figura 1.23: Chapa de alumínio utilizada na construção da estrutura.

2

Objetivos

O objetivo foi desenvolver uma arquitetura de hardware e software que possibilitasse controlar um veículo aéreo de modo que o mesmo execute algumas funções de maneira autônoma, e ainda sim responda a comandos de uma central.

Objetivos especificos:

1. Dar continuidade a trabalhos anteriores utilizando os micro controladores "Arduino Mega"e "Arduino Uno", de modo que fique claro a real capacidade operacional dos mesmos quando submetidos a uma carga de processamento que necessite de uma resposta em tempo real.

2. Desenvolvimento de um protocolo de comunicação entre um cliente e um hardware operacional, de modo que a criação e decodificação de novos comandos seja feita de maneira interativa (um código mais genérico e adaptável).

3. Desenvolvimento de um algoritmo de estabilização autônoma baseado em PID.

4. Estudar os efeitos resultantes da interação de diversos sensores e hardwares diferentes, de modo que seja possível concluir se é viável a criação de outras aplicações partindo do mesmo princípio.

3

Metodologia

A metodologia aplicada partiu da análise individual para, posteriormente, avaliar uma interação em conjunto, isto é, primeiro foram avaliados os sensores e componentes individualmente e, uma vez que o funcionamento estivesse dominado, o mesmo foi integrado ao conjunto operacional. Essa estratégia foi utilizada para todos os sensores (até por questão de manter uma abstração no código, então saber como o sensor funciona permite criar funções em um nível de abstração maior para operar cada componente de maneira mais simples).

A escolha dos micro controladores foi devido ao suporte à linguagem C/C++ e compatibilidade com uma série de sensores e bibliotecas já existentes. Foram avaliados outros projetos (de níveis pessoais e comerciais) de aeronaves do tipo quadricóptero para, observando o funcionamento, levantar quais eram as necessidades operacionais e assim determinar quais sensores seriam necessários (para comunicação e estabilização).

Na construção da estrutura metálica foram avaliados pontos como resistência e peso, o alumínio se mostrou uma boa saída já que é facilmente manipulado (é facilmente cortado e furado) e também possibilita um maior nível de personalização, já que o design pode ser alterado da maneira que for necessário para cada caso; caso fosse utilizado uma estrutura de plástico pronta ficaria mais restrito a esse tipo de expansão e adição de hardware.

Seguindo a proposta da figura 1.1, os dois Arduinos que são acoplados no VANT se comu-

nicam por uma interface IC2 e outro Arduino se comunica via USB com o PC, então antes de uma integração foi necessário realizar testes de envio de pacotes entre essas estruturas, o uso de dois Arduinos no VANT foi justamente para priorizar o recebimento de um pacote de dados.

Com relação aos motores foi utilizado a estrutura de "H-Cop"afim de atender mais a premissa de construir um veículo que se mantivesse o mais estável possível (no caso com quatro motores). Foi realizado um estudo para determinar o modelo dos motores mais adequado, todas essas análises foram feitas com base em especificações técnicas e posteriormente comprovadas por testes práticos.

Os detalhes de cada área do projeto, tal como a explicação dos códigos e do hardware, serão apresentados no próximo capítulo.

4

Resultados e discussão

4.1 Visão geral do projeto final

O projeto final é composto pelo VANT, pelo emissor e por uma plataforma de controle (PC):

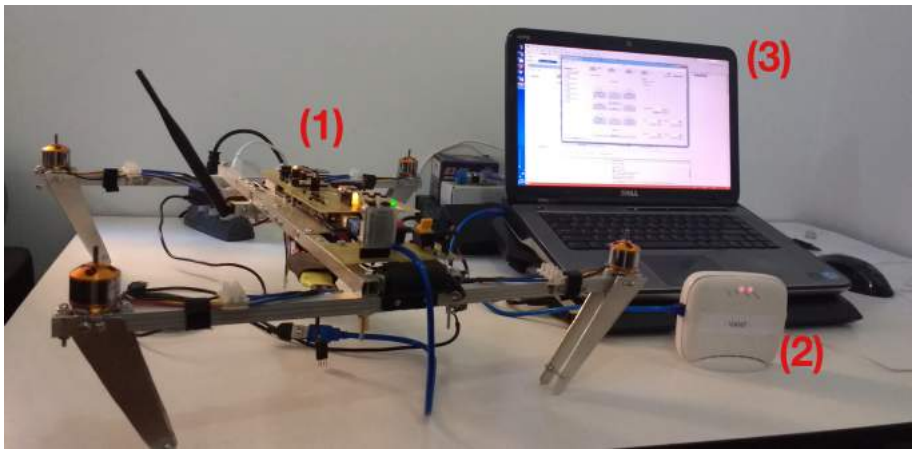

Figura 4.1: Visão geral do projeto final.

Pela figura 4.1 podemos analisar os itens em destaque:

1. VANT: Estrutura do quadricoptero completo.

2. Emissor: foi utilizado um textttcase de modem antigo para construção do módulo emissor.

3. Painel de controle: software que controla o VANT rodando em um PC.

4.2 Estrutura metálica

Nessa etapa será detalhado toda construção da estrutura metálica do VANT, como já mencionado o modelo base adotado segue o conceito "H-Cop"(mostrado na figura 4.2, modelada no creo parametric 3.0), esse design favorece a estabilidade, uma vez que podemos controlar a rotação das helices de maneira a minimizar um desvio acentuado para algum dos lados (isso é possível alternando entre rotação horária e anti-horária para cada um dos motores, tal como mostrado na figura 4.3, esse padrão segue por todo o desenvolvimento e os eixos y e x foram considerados para etapa de controle de estabilidade).

Na figura 4.3 cada motor é marcado com uma nomenclatura (M1,M2,M3 e M4), esses valores são importantes pois essa é a base para todo algoritmo desenvolvido, consequentemente qualquer menção a "motor 1"será referente ao "M1"que está em destaque na imagem; No item sobre motores será abordado com mais detalhes como é feito a orientação de giro (como se configura essa propriedade)

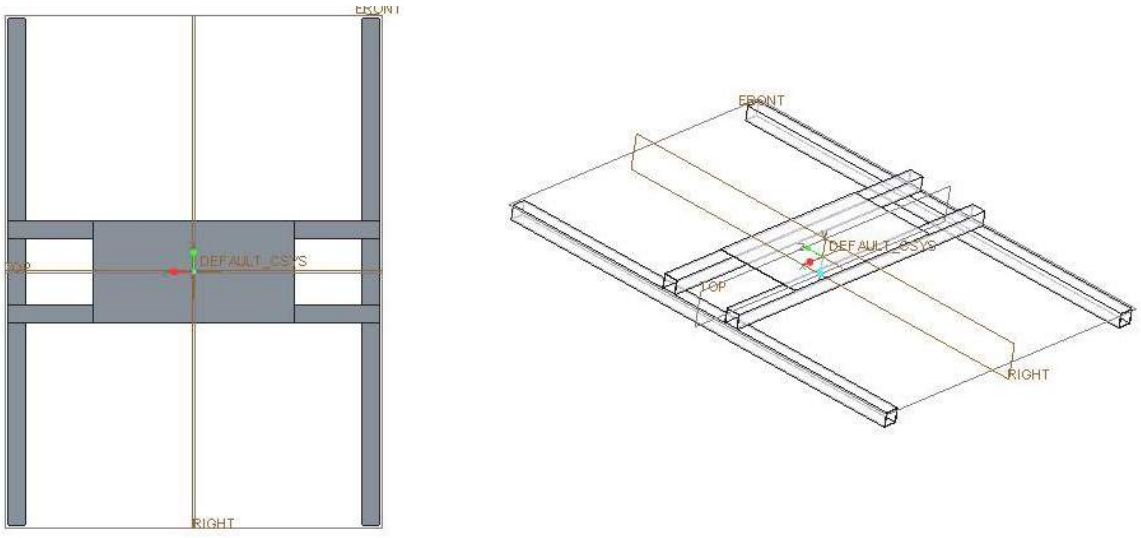

Figura 4.2: Modelagem do protótipo em 3D.

As medidas adotadas para a construção partiram de observações a modelos já existentes e são detalhadas na imagem 4.4.

Valores para as medidas destacadas na imagem 4.4 (os valores que divergem na imagem, como o caso do (2) e do (6), são os casos em que há a largura da própria barra de alumínio que mede 1,3 [cm]; Os pontos em vermelho destacam os locais de fixação dos parafusos):

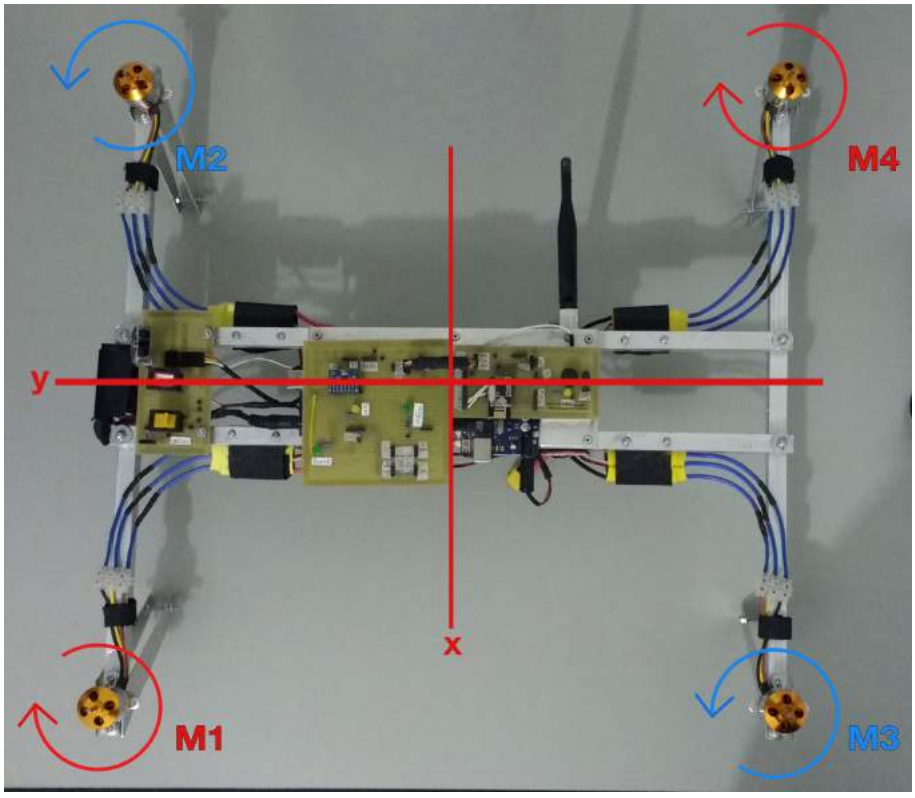

Figura 4.3: Eixos e sentidos adotados no projeto.

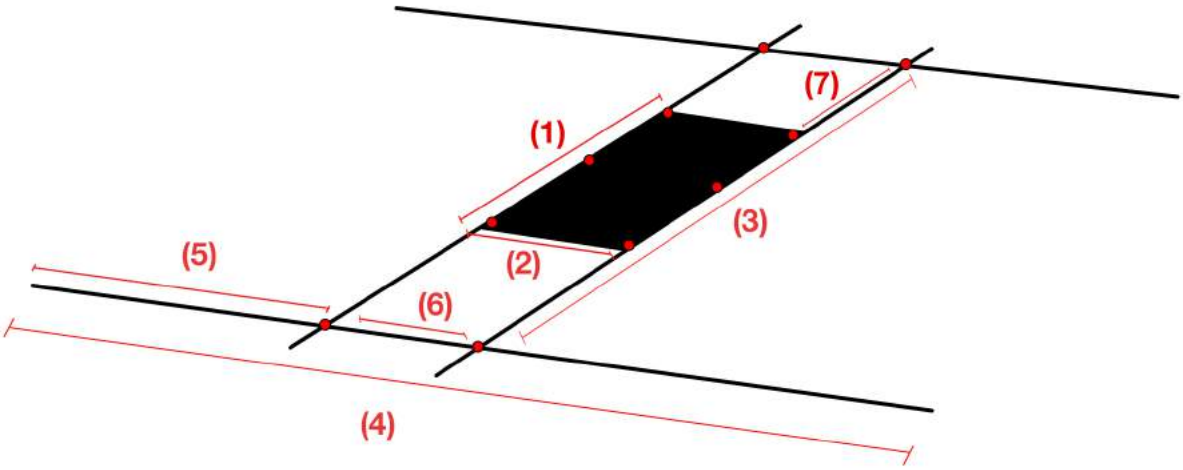

Figura 4.4: Detalhes de construção do modelo.

(1): 20,0 [cm]

(2): 8,0 [cm]

(3): 47,0 [cm]

(4): 47,0 [cm]

(5): 19,3 [cm]

(6): 5,9 [cm]

(7): 13,5 [cm]

4.3 Motores, hélices e bateria

4.3.1 Motores

Foram utilizados quatro motores brushless modelo 2212 outrunner [1] de 1000kV. São motores de corrente máxima em 12A (60s) com potência máxima de 150w e operando com bateria de 11v (3s). O valor de kV é uma notação para rotação por volt (rpm/v), quando utilizamos uma alimentação de 11.3v calculamos a rotação máxima do motor como: 11.3v * 1000kV = 11300 [rpm].

Figura 4.5: Motor brushless 2212.

Esse tipo de motor tem por característica não possuir escovas, ou seja, não há contato mecânico entre os dois módulos para passagem de corrente. Isso resulta em uma melhor eficiência (não há perda de calor nessa parte), menor consumo de energia, maior vida útil e um alto torque.

Através de testes de empuxo (utilizando um peso e uma balança) é possível obter os seguintes valores de força resultante para o motor (utilizando a hélice de 10', que será abordada em seguida). As forças geradas pelos motores se contrapõem à força peso aplicada no centro de gravidade (em sentido), o processo é ilustrado na figura 4.6.

Relacionamos as forças pela seguinte fórmula: 4*F - P = m*a.

[1] Maiores detalhes: http://www.flybrushless.com/motor/view/206/

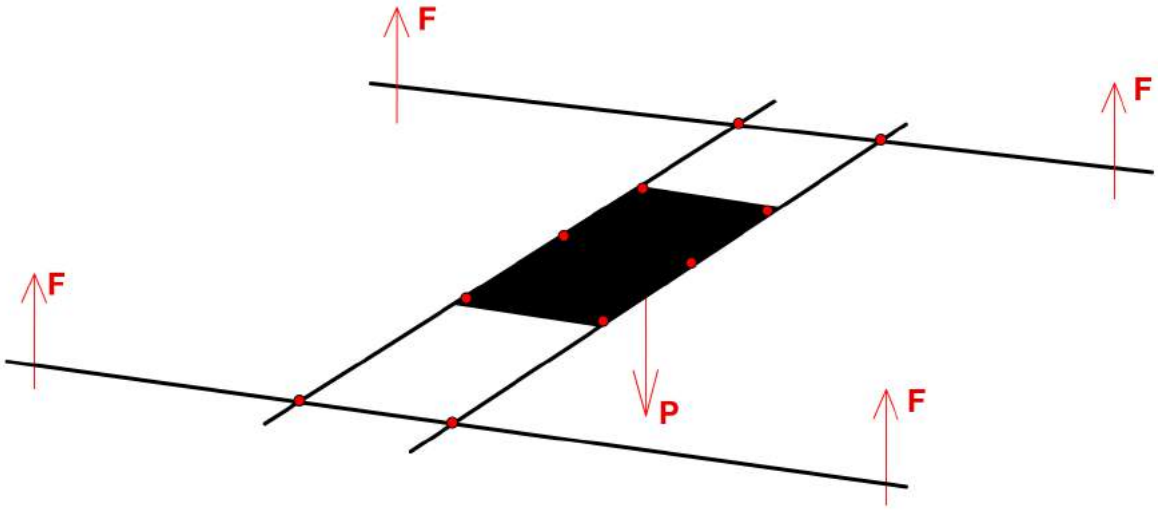

Figura 4.6: Forças exercidas no VANT.

Se houver uma aceleração isso implica em um movimento para cima ou para baixo, com isso conseguimos determinar a potência necessária para erguer o aeromodelo (dado que sabemos a massa e a força peso). Experimentalmente um motor gera uma força de 212[g] em 4000rpm, 620g em 6360rpm e 770g em 7200[rpm]. Portanto os motores adotados, pelas especificações, possuem potência necessária para levantar toda a estrutura (a massa final ficou em torno de 1100[g]).

O controle dos motores é abstraído por um ESC (`Eletronic Speed Control`) de modo que podemos utilizar a biblioteca `Servo.h` (o código 1 ilustra como efetuar o controle dos motores utilizando um potenciômetro).

4.3.2 Hélices

As hélices são importantes pois influenciam no modo de operação do VANT, para esse projeto foram utilizadas hélices de 10'x4,5 modelo 1045r, como já mostrado foi necessário adquirir duas que giram no sentido horário e duas no sentido anti-horário.

Uma observação sobre a fixação da hélice no motor: o conjunto das hélices acompanha alguns acessórios (algumas circunferências de plástico) que podem ser colocadas entre o motor e a hélice, foi necessário testar algumas combinações até que o conector prendesse de maneira satisfatória (onde a hélice não se desprendesse do motor).[2]

[2]Figura 4.7 reproduzida de: http://goo.gl/v5zN25/

Figura 4.7: Modelo de hélice utilizado no projeto.

4.3.3 Bateria e alimentação

Foi utilizado uma bateria de lipo 3s (3 cédulas de 3.3v resultando em uma bateria de 11v) com capacidade de 2200mah, de alta descarga (40/50c). Esse modelo foi utilizado para garantir que a corrente fornecida aos motores fosse suficiente, esse calculo pode ser feito multiplicando a capacidade pela saída: 2200mah * 40c = 88000mA = 88A, que é mais do que suficiente (na prática poderia ser uma bateria de descarga menor) para os quatro motores (no qual cada um tem uma corrente máxima de 12A).[3]

Foi adquirido um cabo `Xt60` para distribuir energia entre os motores, essa alimentação é separada da alimentação de outros componentes de hardware (figura 4.10).

Uma alternativa seria utilizar uma placa de distribuição, mas a proposta do cabo se mostrou melhor para organização dos elementos do VANT (pois o cabo fica na parte de baixo, junto com a bateria).[4]

[3] Figura 4.9 reproduzida de: http://goo.gl/yV3ra0/
[4] Figura 4.10 reproduzida de: http://goo.gl/Um6PzR/

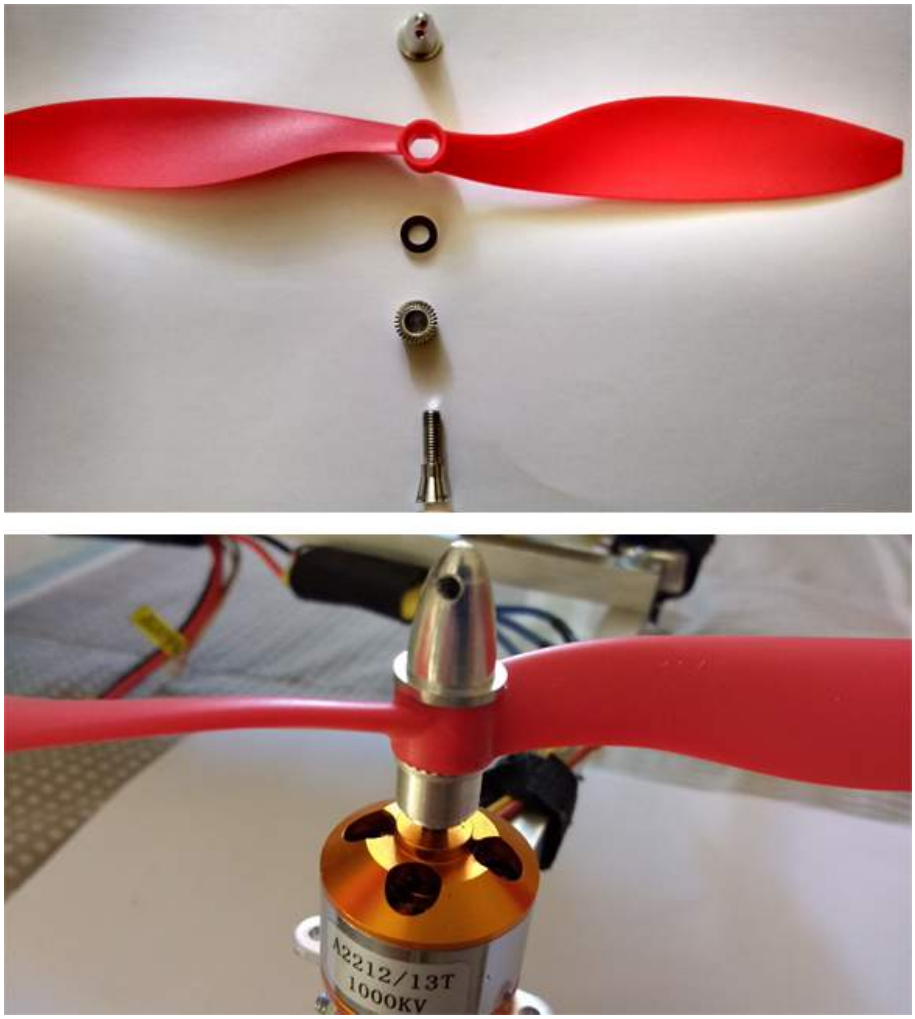

Figura 4.8: Fixação das hélices no motor.

Figura 4.9: Bateria de lipo 3s.

Figura 4.10: Cabo distribuição Xt60 para 4 X 3.5mm bullet.

4.3.4 ESC

O ESC é responsável pelo controle de velocidade do motor. Ele controla a velocidade por meio da modulação por largura de pulso [5], conhecida também como "PWM"(`Pulse-Width Modulation`), ele determina quanto de energia deve chegar ao motor de acordo com uma série de transistores, quanto mais os interruptores são conectados, maior será a potência que o motor é capaz de receber (por consequência terá uma rotação maior). O uso desse tipo de tecnologia vai de encontro a uma necessidade ao se construir um aeromodelo: o peso é um fator importante; Utilizando motores sem escova feitos de metal mais leve reduzem esse peso (estes que requerem o uso de um controlador eletrônico), o único contra fica por conta do custo ser maior devido ao uso dos componentes adicionais (um ESC para cada motor adicionado).

Foi utilizado um modelo de 30A mostrado na figura 4.11. A ligação entre o motor, esc e arduino é mostrada na figura 4.12. A ordem dos três fios do motor não importa para que ele funcione, mas dependendo da combinação ele vai girar no sentido horário ou anti-horário (no projeto foi determinado o sentido de cada motor e através de testes foram obtidos as ligações para obter essa configuração, como visto na figura 4.3).

[5] Saiba mais em: http://www.mecanicaindustrial.com.br/179-o-que-e-controle-eletronico-de-velocidade/

Figura 4.11: Controlador eletronico de velocidade (ESC).

4.3.5 Exemplo para utilização dos motores

Antes de começar a desenvolver o projeto propriamente dito foi necessário testar algumas partes isoladamente, no caso dos motores é possível fazer a ligação e testar o controle de velocidade com um potenciometro, a ligação é mostrada na figura 4.12, o pino VCC no cabo de controle do motor (no caso os cabos do ESC) não é utilizado (apenas o GND e o SIGNAL).

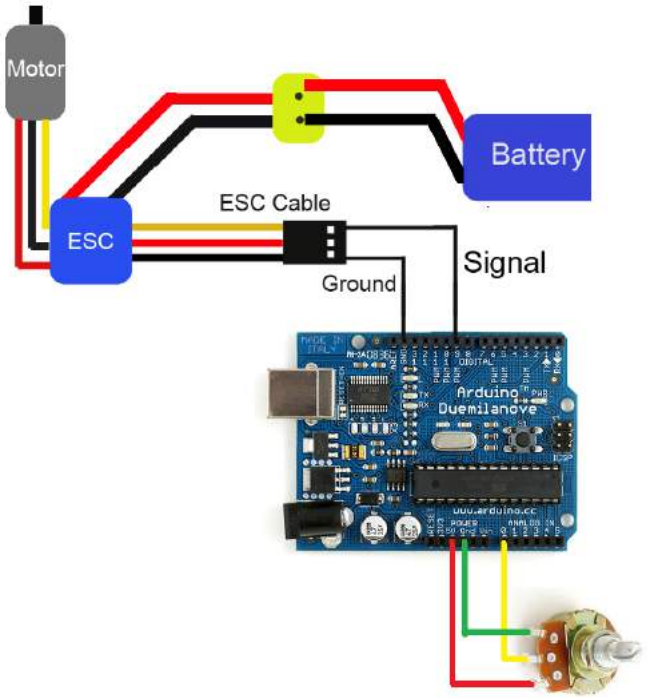

Figura 4.12: Ligação entre motor, esc e bateria.

O código exemplo para operar os motores é mostrado no código 1, apesar da figura 4.12

mostrar apenas um motor e utilizando o arduino Uno é possível utilizar outra placa (como no projeto é utilizado um arquino Mega e para testes também foi utilizado um arduino Due), é só alterar as portas PWM utilizadas e ligar o potênciometro em uma porta analógica.

```
#include <Servo.h>

Servo myservo1; //objeto para operar o motor

int potpin = 0; //pino analógico usado para o potenciômetro
int val; //valor de leitura do pino analógico

void setup()
   {
    myservo1.attach(7); //pino PWM do arduino

    Serial.begin(9600);

    //calibrar o motor.
    myservo1.writeMicroseconds(2100);
    delay(3);
    myservo1.writeMicroseconds(800);
    delay(2);
    myservo1.writeMicroseconds(1500);
    delay(2);
    myservo1.writeMicroseconds(2100);
   }

void loop()
   {
    val = analogRead(potpin); //(valor entre 0 e 1023)
    val = map(val, 0, 1023, 0, 179); //(valor mapeado entre 0 and 180)
    myservo1.write(val); //executa o comando para determinar a velocidade
    delay(15);
   }
```

Algoritmo 1: Controle do motor utilizando potênciometro.

Uma observação para o código, quando o motor é ligado na bateria e no arduino é comum a emissão de um apito em intervalos de um segundo (aproximadamente); Isso ocorre porque o ESC é um dispositivo programável e precisa de certos parâmetros para operar corretamente. O mesmo pulso utilizado para operar o motor também é utilizado para calibragem (no código 1 a calibragem ocorre na execução do `Setup`). Se tudo ocorrer bem devem soar apitos mais frequentes seguidos por um mais longo e em seguida o motor deve estar operacional, alterando a rotação de acordo com o valor do potenciômetro.

Algumas anomalias encontradas foram: em alguns casos o bipe produzido era em um intervalo de tempo bem menor (algo como um alarme de emergência), em outros ocorria de, mesmo

executando a operação de calibração, continuar o bipe inicial sem que fosse possível controlar o motor. Esses casos foram resolvidos revisando a ligação elétrica dos pinos de sinal e ground do motor e também modificando a parte elétrica (isolando a alimentação dos motores do fornecimento de energia para o hardware), então a bateria de lipo ficou exclusivamente para os motores.

4.4 Painel de controle

O painel de controle é o caminho pelo qual executamos o `input` dos dados, em projetos convencionais é normalmente utilizado um rádio controle mas nesse projeto foi adotado um sistema diferente: um software mapeia comandos que são enviados para um dispositivo e este envia novamente por uma conexão sem fio os dados para o receptor (que está no VANT). Essa abordagem foi seguida porque permite uma maior versatilidade do projeto, é possível personalizar e complementar as informações que são exibidas na tela do operador, esse tipo de programação ainda permite controlar outras aplicações sem fio de maneira genérica e prática.

Como se trata de uma produção científica, utilizar um rádio controle seria uma perda de oportunidade de pôr em prática conceitos de controle e automação, além de que seria perdido também a oportunidade de experimentar a interação e comunicação entre plataformas distintas

Focando no painel de controle: foi utilizado o `Visual Studio 2015` para o desenvolvimento (por consequencia o software é feito para uso em sistemas windows) em linguagem C. O projeto poderia ser desenvolvido em outra linguagem ou plataforma, mas foi adotado essa por conta de uma maior familiaridade com a tecnologia empregada.

Os elementos em destaque na figura 4.13 são partes que devem ser explicadas (algumas partes deixadas sem a marcação tratam de itens opcionais ou são replicações de outros implementados).

1. Botões de controle: são utilizados para acionar/desativar alguma função, todos enviam uma estrutura de dados que é utilizado pelo VANT (o detalhe do código e a lista de comandos estarão em a seguir).

2. Controle de velocidade: os comandos permitem o envio de um dado, nesse caso o controlador pode enviar um valor que será utilizado para determinar a velocidade de cada motor

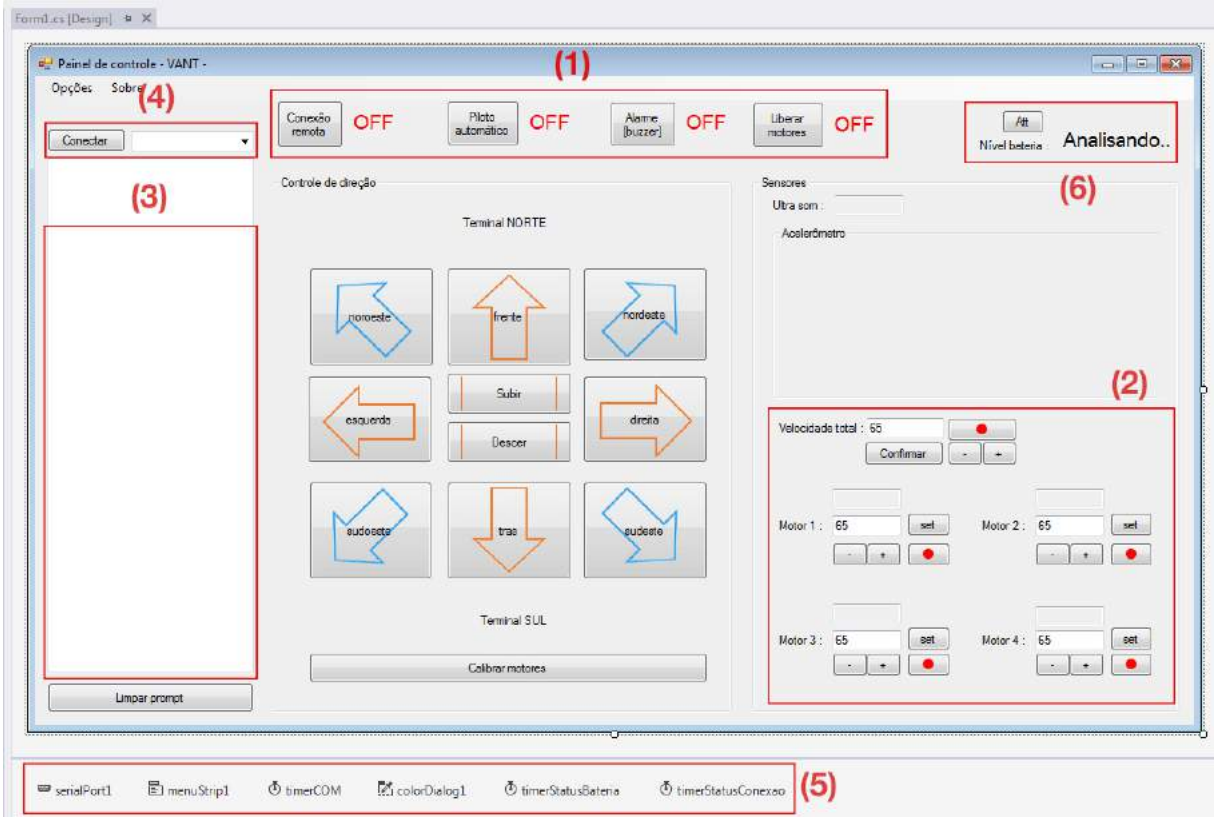

Figura 4.13: Painel de controle, editando pelo VS2015.

(o botão vermelho envia a velocidade mínima, é uma espécie de "parada de emergência").

3. Prompt: é uma espécie de prompt que exibe um histórico a cada comando enviado, quando recebe algum valor também é exibido nessa área.

4. Serial: o software se conecta ao emissor (um arduino ligado à USB) pela porta serial, da mesma forma que se utiliza essa porta para carregar um código na placa é possível conectar para comunicação de dados.

5. Componentes: os componentes servem para agilizar algumas configurações, o `SerialPort` serve para suporte a conexão serial (USB) para comunicação com o arduino Uno (emissor), os do tipo `Timer` são utilizados para programar algumas execuções automáticas a cada certo intervalo de tempo (no caso para checar as portas seriais disponíveis ou para enviar comandos periódicos ao VANT).

4.4.1 Porta serial

O painel de controle possui um funcionamento básico: ele conecta através de uma porta serial ao emissor para enviar e receber um pacote de dados, no caso da porta serial manipulamos as operações por um componente (mostrado na figura 4.14).

A conexão é feita por um botão e uma caixa de texto que lista todas as portas disponíveis (no caso o dispositivo conectado a entrada USB), o timer `timerCOM` é responsável por atualizar essa lista de portas através de uma rotina (código 2).

Com o dispositivo conectado (os detalhes vão ser tratados nos itens a seguir) a porta deve ser listada como disponível e basta clicar no botão *conectar* (figura 4.15). a conexão é explicada pelo código 3. Ao conectar podemos transferir e receber dados da entrada serial (da mesma maneira que é possível utilizando o serial da própria Arduino IDE).

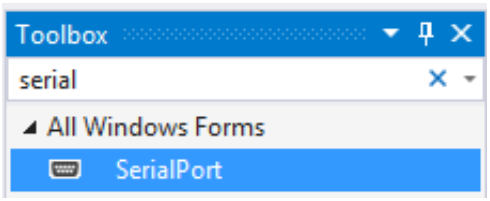

Figura 4.14: Porta serial na lista de componentes.

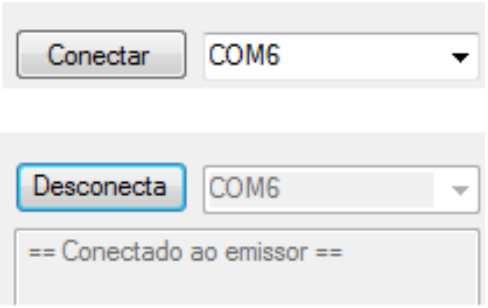

Figura 4.15: Mecanismo de conexão com a porta serial.

```
private void atualizaListaCOMs()
{
  int i;
  bool quantDiferente; //flag para sinalizar que a quantidade de portas mudou
    //considera que não existem portas diferentes
    i = 0;
    quantDiferente = false;
    //verifica se a quantidade de itens na comboBox1
    //é diferente das portas disponiveis:
    if (comboBox1.Items.Count == SerialPort.GetPortNames().Length)
    {
      //se for igual, pode acontecer de ter alguma que é diferente:
      //no caso de conectar outro dispositivo e tirar um que estava ligado.
      //a quantidade continua igual mas as portas mudaram
      foreach (string s in SerialPort.GetPortNames())
      {
        if (comboBox1.Items[i++].Equals(s) == false)
        {
          //se achou um item diferente, marca que há
          //portas que não estão listadas
          quantDiferente = true;
        }
      }
    }
    else
    {
      //se o comprimento for diferente já sabemos que há
      //portas não listadas
      quantDiferente = true;
    }

    //Se não foi detectado diferença
    if (quantDiferente == false)
    {
      return; //retorna sem alterações
    }
    //limpa comboBox
    comboBox1.Items.Clear();
    //adiciona todas as COM diponíveis na lista
    foreach (string s in SerialPort.GetPortNames())
    {
      comboBox1.Items.Add(s);
    }
    //seleciona a primeira posição da lista
    if(comboBox1.Items.Count > 0)
    {
      comboBox1.SelectedIndex = 0;
    }
    else
    {
      //isso é importante pois se você desconecta o
      //único dispositivo não terá nada pra listar e
      //vai dar um erro e encerrar o software
      comboBox1.Text = " ";
    }
  }
```

Algoritmo 2: Função para atualizar lista de portas disponíveis.

```
private void button9_Click(object sender, EventArgs e)
{
  //verifica se a serial port está aberta
  if (serialPort1.IsOpen == false)
  {
    //como não está, é feito uma tentativa de conexão
    //com a porta que está selecionada no componente
    //comboBox1:
    try
    {

    serialPort1.PortName = comboBox1.Items[comboBox1.SelectedIndex]
                                .ToString();
    serialPort1.Open();
    }
    catch
    {
      return;
    }
    if (serialPort1.IsOpen)
    {
      //quando a conexão é feita com sucesso
      //altera o texto do botão para desconectar
      //e desabilita a comboBox1
      btConectar.Text = "Desconectar";
      comboBox1.Enabled = false;
      //Esse trecho escreve uma linha no componente utilizado
      //como prompt, ele é utilizado para listar um histórico
      //de comandos executados, tal como dados recebidos
      textBoxReceber.AppendText("\n\n == Conectado ao emissor == \n");
    }
  }
  else //O click foi feito para efetuar a desconexão
  {
    try
    {
      serialPort1.Close();
      comboBox1.Enabled = true;
      btConectar.Text = "Conectar";
      textBoxReceber.AppendText("\n\n == Desconectado do emissor == \n");
    }
    catch
    {
      return;
    }

  }
}
```

Algoritmo 3: Função para efetuar a conexão com o dispositivo serial.

4.4.2 Lista de comandos

A única função "diferente"foi apresentada anteriormente (a conexão USB), todas as demais ações são baseadas no envio e recebimento de pacotes (contendo comandos e dados).

Foi construído a lista com os comandos criados e de que maneira vão ser tratados pelo painel de controle (os que forem de exclusiva execução no VANT serão detalhados em outros capítulos)

Havia a necessidade de se transmitir comandos que eram acompanhados de determinados valores, outras vezes comandos apenas (sem um dado associado) e em alguns casos comandos para requisitar alguma informação.

Dessa forma o mecanismo desenvolvido para a comunicação acabou composto por dois bytes: um byte para o comando (`cmd`) e outro byte para o dado (`valor`). Os dois bytes são enviados juntos (como em uma variável do tipo *int* de 8bits: 11110000, os 4 primeiros bits são correspondente ao comando e os 4 bits mais a direita correspondente ao dado).

É um método bem simples de transferir um pacote de dados entre dispositivos mas em um futuro uma melhoria como criptografia seria ideal para proteção dos dados transmitidos.

Também foi adotado um intervalo de valores para os quais os comandos seriam do tipo "simples"ou "composto"; Comandos simples são uma via de mão única: eles são transferidos via USB para o emissor e então enviados ao VANT, já comandos compostos precisam de um retorno.

Esse tratamento se fez necessário devido ao funcionamento do componente de rádio frequência que, apesar de operar como transmissor e receptor, precisa que o modo de operação seja alterado manualmente, então quando um comando composto é acionado o "anteriormente"emissor passa a operar como receptor até receber a informação do VANT.

O intervalo define que comandos entre 0000 até 0124 são tratados como simples, a partir do 0125 até 9999 são comandos compostos.

É possível ver os exemplos de envio de comandos via serial (no software do painel de controle) no código 5 e código 6.

Tabela 4.1: Tabela de comandos

cmd	dado	retorno?	descrição
1	velocidade para o motor1	não	Controle de velocidade do motor1 contendo a velocidade (dinâmica) 2 bytes: 0001XXXX.
2	velocidade para o motor2.	não	Controle de velocidade do motor2.
3	velocidade para o motor3.	não	Controle de velocidade do motor3.
4	velocidade para o motor4.	não	Controle de velocidade do motor4.
6	velocidade para todos os motores.	não	Controle de velocidade de todos os motores (simultaneamente).
7	não	não	Calibração dos motores, envia o comando para executar a rotina de calibração dos motores (envia alguns sinais PWM por determinados intervalos de tempo).
8	não	não	Liberação dos motores: altera uma flag no VANT que habilita o controle remoto dos motores.
9	não	não	Trava dos motores: altera a flag para desabilitar o controle remoto dos motores (o algoritmo passa a não executar o loop responsável pelo comando enviado ao ESC).
14	não	não	Acionamento do buzzer: o VANT é equipado com um alarme que irá apitar em determinados intervalos de tempo até que seja desativado.
15	não	não	Desligamento do buzzer: desabilita a execução do bipe.
125	não	Sim, retorna o cmd 125 e valor 1 ou 0 2 bytes: 01250001 ou 01250000.	Conexão inicial: (essa é a conexão entre o emissor e o VANT), é utilizada para verificar se tudo está OK, envia um valor que aciona um LED no VANT e retorna a confirmação.
126	não	Sim, retorna o cmd 126 e um valor dinâmico 2 bytes: 0126XXXX ou 0126XXXX.	Leitura da bateria: o receptor do VANT realiza uma leitura analógica e retorna o valor, que é comparado com um máximo ideal para determinar a % de bateria (no caso a de lipo).

```
using System.IO.Ports;  // necessário para ter acesso as portas

namespace VantPainelDeControle
{
    public partial class Form1 : Form
    {

        string RxString;
        int pacote_recebido;
        int status_buzzer = 0;
        int status_motores = 0;

        public Form1()
        {
            InitializeComponent();
            timerCOM.Enabled = true;
            //timerStatusBateria.Enabled = true;
            //timerStatusConexao.Enabled = true;
        }


  ....
```

Algoritmo 4: Variáveis de controles utilizadas no painel.

```
  private void btnM1_Click(object sender, EventArgs e)
  {
    // cmd:
    // 0001 # controle da velocidade do motor 1
    // retorno: nao
    // valor: sim

    byte temp = byte.Parse(lblM1.Text);
    //                          cmd  valor
    byte[] data = new byte[] { 0001, temp };

    if (serialPort1.IsOpen == true)
    {//porta está aberta
      //escreve o vetor de 2 bytes na saida serial
      serialPort1.Write(data, 0, 2);

    }
    //exibe o texto do comando executado no painel de controle
    //como uma especie de prompt:
    textBoxReceber.AppendText("\n\n == Comando enviado >> \n");
    textBoxReceber.AppendText("cmd: 0001 \n");
    textBoxReceber.AppendText("valor: " + lblM1.Text + "\n\n");
  }
```

Algoritmo 5: Exemplo de comando enviado pelo software de controle.

```
1   private void trataDadoRecebido(object sender, EventArgs e)
2   {
3     //Um dado recebido é composto por um cmd (comando) e um valor associado.
4     //dessa forma é possível examinar qual ação tomar sem ter salvo
5     //o comando enviado anteriormente
6     //textBoxReceber.AppendText(RxString + "\n");
7
8     //realiza o parsing do comando e do valor recebidos pela serial
9     int cmd = pacote_recebido / 100000;
10    int valor = pacote_recebido - cmd*100000;
11
12    //Console.WriteLine("pct_recebido: " + pacote_recebido);
13    //Console.WriteLine(cmd);
14    //Console.WriteLine(valor);
15
16    textBoxReceber.AppendText("\n\n == Recebido resposta << \n");
17    textBoxReceber.AppendText("cmd: " + cmd.ToString() + "\n");
18    textBoxReceber.AppendText("valor: " + valor.ToString() + "\n\n\n");
19
20    //tratar a descrição e ação do comando recebido:
21    switch (cmd){
22
23      case 125:
24        //conexao inicial (recebido a confirmação)
25        //porem, pode ter sucesso (Valor=1) ou falhado (valor=0);
26
27        if (valor == 1)
28        {
29          //conexao bem sucedida
30          lblStatusConexao.Text = "ON";
31          lblStatusConexao.ForeColor = System.Drawing.Color.Green;
32        }
33        else if(valor == 0)
34        {
35          lblStatusConexao.Text = "OFF";
36          lblStatusConexao.ForeColor = System.Drawing.Color.Red;
37        }
38        break;
39
40      case 126:
41        //Recebido valor da bateria
42        //leitura cedula unica: 4.13v = 100%
43        //                      2.13v = 0% (nivel altamente critico)
44
45        double porcent_bat = ((double)(valor-270) / (double)(413-270))*100;
46        int bat_int = (int)porcent_bat;
47        lblNivelBateria.Text = bat_int.ToString() + "%";
48        //Console.WriteLine(porcent_bat);
49        break;
50    }
51
52  }
```

Algoritmo 6: Função que trata dados recebidos pela entrada serial (para comandos compostos).

4.4.3 LEDs de controle

Antes de prosseguir para a análise dos demais controladores temos a abordagem sobre os LEDs utilizados para controle, no caso eles funcionam como um `feedback` visual para alguma mudança de estado ou execução de algum trecho chave do código. Esses sinais são muito importantes pois é possível concluir que algo deu errado mais rapidamente, diferentemente de uma aplicação 100

Dessa maneira os LEDs foram equipados como mecanismos de confirmação, eles serão mencionados de acordo com a explicação de determinada parte mais especifica do projeto então deixaremos na figura 4.16 a localização e uma breve nota sobre cada um desses indicadores.

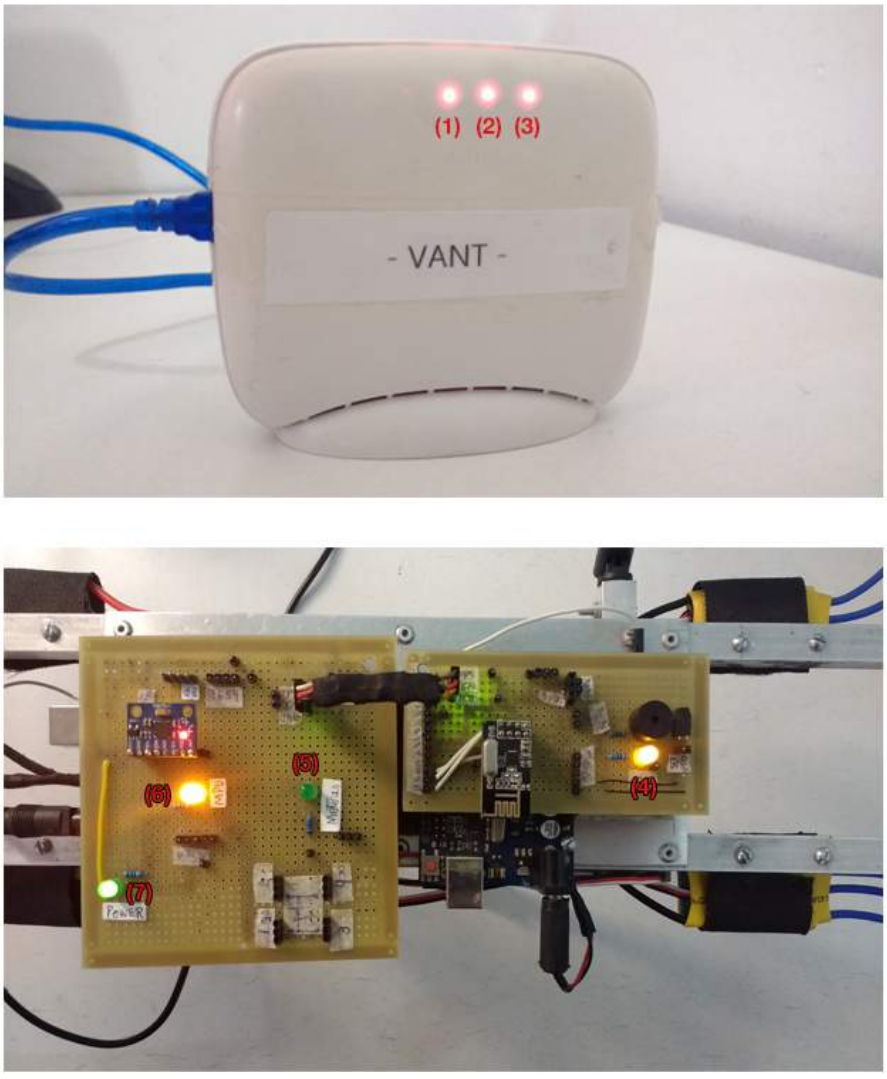

Figura 4.16: Conjunto de leds indicadores.

1. Power emissor: é ligado automaticamente quando ligamos o emissor a entrada USB.

2. Conexão ao VANT: quando é executado o comando de conexão inicial, caso tenha sucesso (o vant envia uma resposta confirmando a comunicação) acende o led de conexão.

3. Comunicação: o led pisca (acende por um intervalo de tempo e desliga) quando algum comando é enviado, uma maneira de visualizar a comunicação (principalmente em casos de comandos autônomos).

4. Conexão no receptor: também é acionado na conexão inicial, mas nesse caso funciona quando o VANT recebe a solicitação de comunicação, antes de enviar a resposta ele aciona o led indicando sucesso na operação.

5. Status dos motores: quando acesso indica que foi enviado o sinal para destravar o controle dos motores do VANT, analogamente se apagado indica que os motores estão desabilitados a qualquer tipo de operação.

6. Status do acelerômetro: o componente MPU precisa ser inicializado e se tudo ocorrer de acordo aciona esse led de controle.

7. PID: o led indica que houve uma atividade de controle na estabilização (é ligado e desligado em seguida), é operado diretamente pelo algoritmo do PID (será abordado no item sobre o receptor secundário).

4.4.4 Emissor primário

O emissor primário recebe os bytes da mensagem via serial e encaminha para o receptor no VANT. A estrutura interna é mostrada na figura 4.17, é composto por um conjunto de três LEDs e um componente de rádio frequência (modelo NRF24L01+).

O esquemático das ligações é mostrado na figura 4.18. A rotina do emissor é receber a mensagem e armazenar em um inteiro (de 2 bytes) e em seguida enviar para o VANT.[6]

Existe uma referência ao arquivo *pacote.h*, o código 7 foi criado para abstrair a composição da mensagem que é enviada/recebida (os 2 bytes de mensagem), como sempre é nesse formato para todos os dispositivos usar um `include` evita a repetição de código.

[6] Para mais informações sobre o RF24: http://br-arduino.org/2015/06/arduino-sem-fio-nrf24l01.html

```
#ifndef Pacote_h
#define Pacote_h

#include "Arduino.h"

typedef struct {
  int cmd;
  int valor;
} pacote;

#endif
```

Algoritmo 7: Código da estrutura de dados que é enviada/recebida

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

#include "pacote.h"
pacote dados;

//inicialização do radio controle,
//caso fosse um arduino mega os pinos
//seriam: radio(48, 49);
RF24 radio(7, 8);

//Declaração dos canais (uma espécie de pipe) para leitura e escrita.
//OBS: no receptor os endereços serão invertidos
//para formar a dupla de leitura e escrita)
const byte rxAddr[6] = "00001";
const byte wxAddr[6] = "00002";

//um buffer temporário que recebe um inteiro via USB
int buff;
int done;

//declaração dos LEDs
#define PIN_STATUS 1 //LED 3
#define PIN_CON 2    //LED 2
#define PIN_MSG 3    //LED 1

void setup() {

  //Comunicação serial com timeout (para ajustar a velocidade de açãoo
  Serial.begin(9600);
  Serial.setTimeout(50);

  //inicialização da comunicação sem fio
  radio.begin();
  radio.setRetries(15, 15);
  //configura os canais de comunicação wireless
  radio.openWritingPipe(wxAddr);
  radio.openReadingPipe(0, rxAddr);
  radio.stopListening();

  //Leds de controle
  pinMode(PIN_STATUS, OUTPUT);//luz esq: ON/OFF
  pinMode(PIN_CON, OUTPUT);//meio: Conexao com drone
  pinMode(PIN_MSG, OUTPUT);//dir: enviar/receber comando

  //Liga o LED de Power
  digitalWrite(PIN_STATUS, HIGH);
}
```

Algoritmo 8: Primeira parte do código do emissor primário

```
void loop() {

  //Descrição do loop:
  //O emissor fica checando continuamente a porta serial na espera de um
  //comando enviado pelo operador C#, quando recebe um comando
  //(inteiro, de 2 bytes) ele envia para o receptor primário
  //e também uma resposta para o cliente c# (para casos em que
  //é necessário receber algum valor do vant).

  if (Serial.available() > 0) {
    //Inicio: leitura do valor enviado pelo pc via usb
  //liga o LED de comunicação
    digitalWrite(PIN_MSG, HIGH);

  //faz a leitura dos dados enviados pela porta serial
    byte buff[2];
    Serial.readBytes(buff, 2);

  //extrai o comando e o valor associado a mensagem enviada
    dados.cmd = (int)buff[0];
    dados.valor = (int)buff[1];

    //Serial.print("Comando enviado: ");
    //Serial.println(dados.cmd);
    //Serial.print("Valor associado: ");
    //Serial.println(dados.valor);

    //envia os dados via conexão sem fio para o receptor primário
    radio.write(&dados, sizeof(dados));
  //desliga o LED de comunicação
    digitalWrite(PIN_MSG, LOW);
    //Fim: leitura do valor enviado pelo pc via usb

  //avaliar tipo de comando: simples ou composto
    //simples: um comando de uma via, apenas é enviado e pronto
    //composto: é enviado mas precisa de uma resposta
  //(como obter o nível da bateria atual)

  //por questões técnicas e de otimização foi considerado o intervalo
    //númerico para determinar se um comando é simples ou composto (uma
  //faixa de valores para cada). dessa forma não seria necessário
    //adicionar mais um componente na estrutura da mensagem permanecendo
  //assim nos 2 bytes (ou no caso 1 INT).
```

Algoritmo 9: Segunda parte do código do emissor primário

```
  //se o valor for um comando composto, aguarda uma resposta do vant:
    if (dados.cmd >= 125) {
      //Serial.println("");
      //Serial.println("Aguardando resposta \n");
      radio.startListening();
      for (int i = 0; i < 500; i++) {
        done = radio.available();
        if (done) { //multiplas tentativas de receber a mensagem
          break;
        }
        delay(4);
      }

      if (done) {
    //recebeu uma resposta do vant, agora precisa tratar de acordo com o
  //comando original
  //se entrou nesse if quer dizer que recebeu uma resposta do VANT
  //portanto a conexão está estabelecida, assim ligamos o LED de conexao
        digitalWrite(PIN_CON, HIGH);
        radio.read(&dados, sizeof(dados));

        switch (dados.cmd) {
          case 125: //comando para estabelecer conexao
            //Serial.println("");
            //Serial.println("== Retorno ==");
            //Serial.println("Conexao estabelecida!, 125, 1");
            //Serial.println(125);
            //Serial.println(1);

            dados.valor = 1;
      //essa escrita escreve na porta serial (USB) uma mensagem
      //de 2 bytes, ela será interpretada e exibida na lista de
      //comandos executados do painel de controle
            Serial.println(dados.cmd * 100000 + dados.valor);
            break;

          case 126: //leitura valor da bateria
      //aqui o valor de leitura foi devolvido pelo VANT e agora
      //transferido para o painel de controle.
            Serial.println(dados.cmd * 100000 + dados.valor);
            break;
        }
      }
      else {
        //não foi recebido uma resposta durante a analise
        //Serial.println("");
        //Serial.println("== Retorno ==");
        //Serial.println("Nenhum dado recebido, verificar alcance!, 125, 0");

        dados.valor = 0;
        Serial.println(dados.cmd * 100000 + dados.valor);

        digitalWrite(PIN_CON, LOW); //conexao deverá ser reestabelecida
      }
      radio.stopListening();
    }
  }
}
```

Algoritmo 10: Terceira parte do código do emissor primário

Figura 4.17: Detalhes da estrutura do emissor primário.

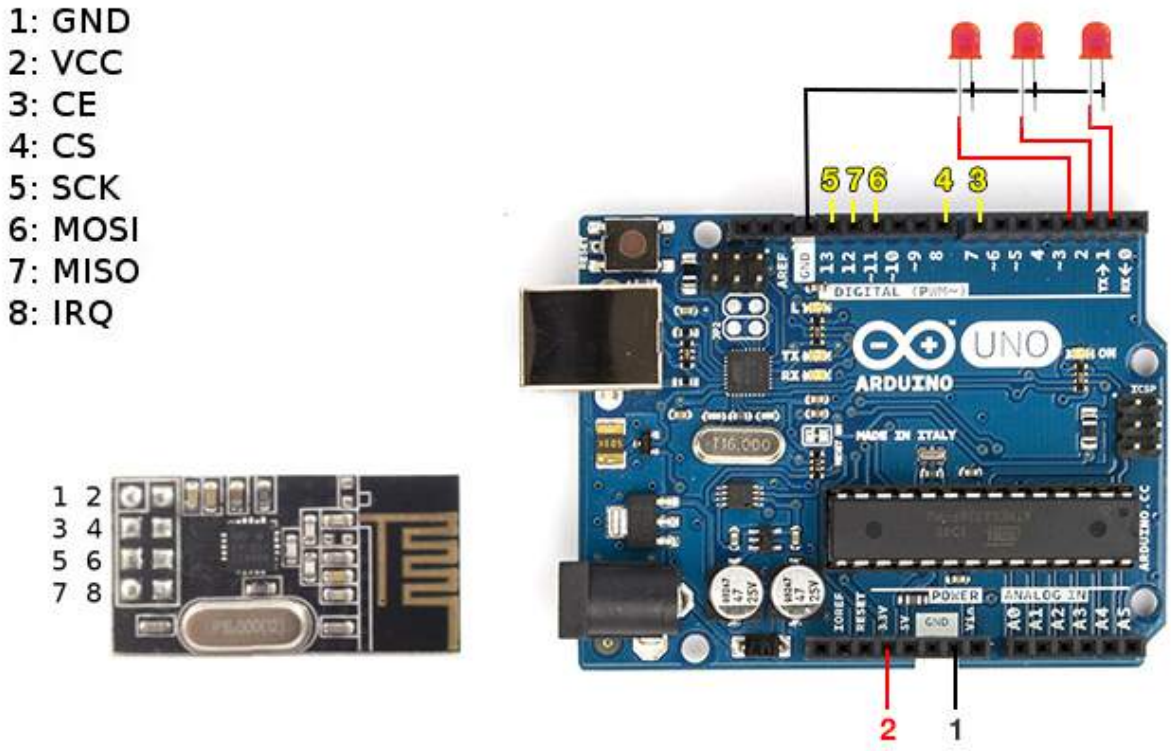

Figura 4.18: Ligação dos pinos no emissor primário.

4.4.5 Receptor primário

O receptor primário tem como prioridade receber os dados pelo sensor de rádio frequência e então repassar o comando para o controlador dos motores (no caso o Arduino MEGA, chamado de receptor secundário), executar uma ação por ele mesmo e/ou enviar uma mensagem de volta para o emissor primário (o arduino ligado na USB do PC).

Ele é equipado com um LED indicador, um buzzer (para emitir um aviso sonoro), um sensor de rádio frequência. Utilizamos a conexão IC2 para comunicação entre os dois arduinos e adicionalmente foi utilizado uma porta analógica para leitura da voltagem da bateria de lipo (que alimenta os motores).

A placa construída pode ser vista na figura 4.19, a visão logo abaixo não é espelhada, segue a mesma sequência de pinos (já foi invertida), na figura 4.20 temos o esquema de ligações realizado na placa.

O mecanismo de comunicação entre os dois Arduinos é chamado de IC2 (`Inter-Integrated Circuit`) e funciona com uma hierarquia mestre-escravo, o receptor primário é o mestre e o

receptor secundário o escravo, quando um comando precisa ser enviado ao Arduino Mega o Arduino Uno inicia a transferência que gera uma interrupção de execução no destinatário (para que essa recepção seja tratada no mesmo instante). Isso permite que múltiplos dispositivos se comuniquem por fio (inclusive o módulo do acelerômetro é operado por esse mesmo mecanismo, então a estrutura completa pode ser vista na figura 4.21). Cada módulo é acessado por um endereço local pré-definido (no código de cada um), o acelerômetro já possui um endereçamento fixo (há a opção para mudar para outro valor acionando uma entrada do sensor, permitindo o uso de dois módulos simultaneamentes).[7]

O código tem uma chave de execução: trabalhar com uma variável globalizada (`pacote dados`) que é utilizada para receber os dados, para enviar os valores para o receptor secundário e também para enviar de volta ao emissor, caso necessário (por isso nas chamadas da função *enviaIC2* não há parâmetros, ela simplesmente considera que o que estiver nessa variável é o que deve ser enviado).

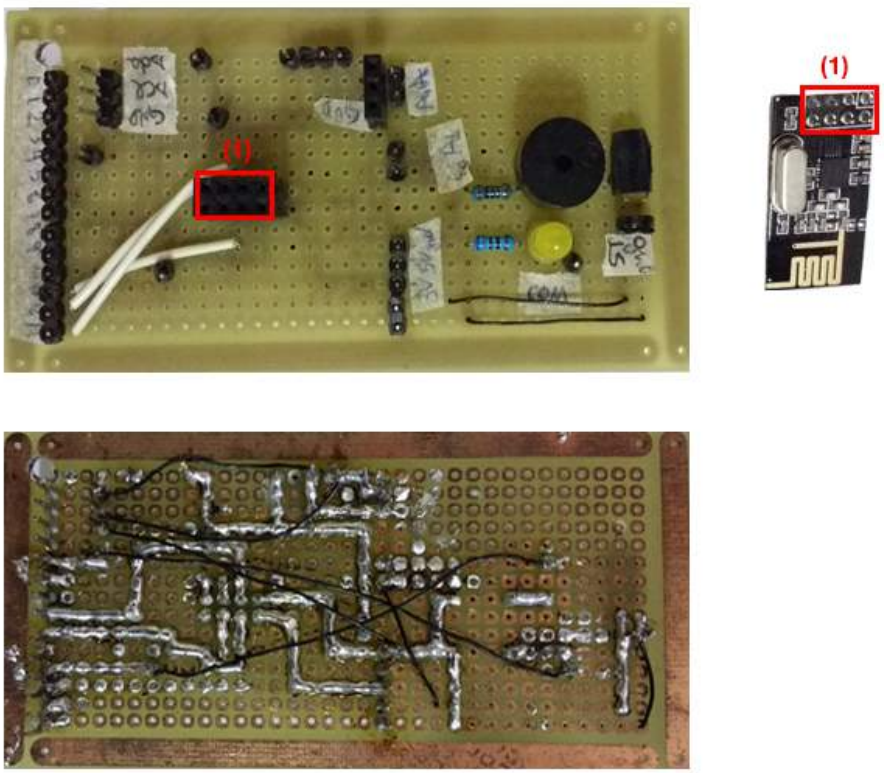

Figura 4.19: Detalhe da construção do receptor primário.

[7] Para mais informações sobre comunicação IC2: http://www.arduinobr.com/arduino/i2c-protocolo-de-comunicacao/

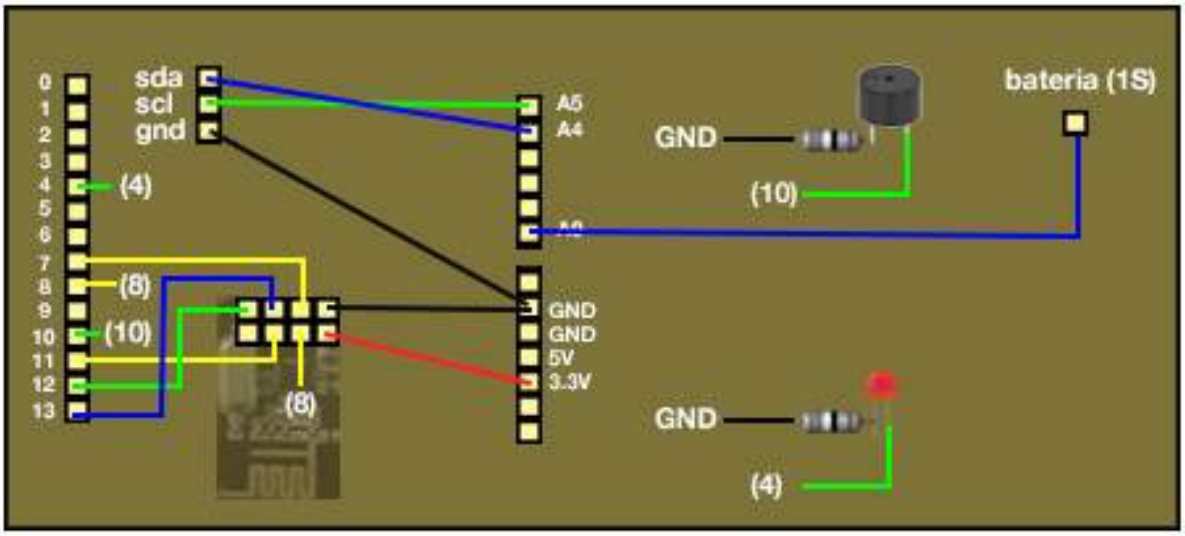

Figura 4.20: Esquema de ligação na placa do receptor primário.

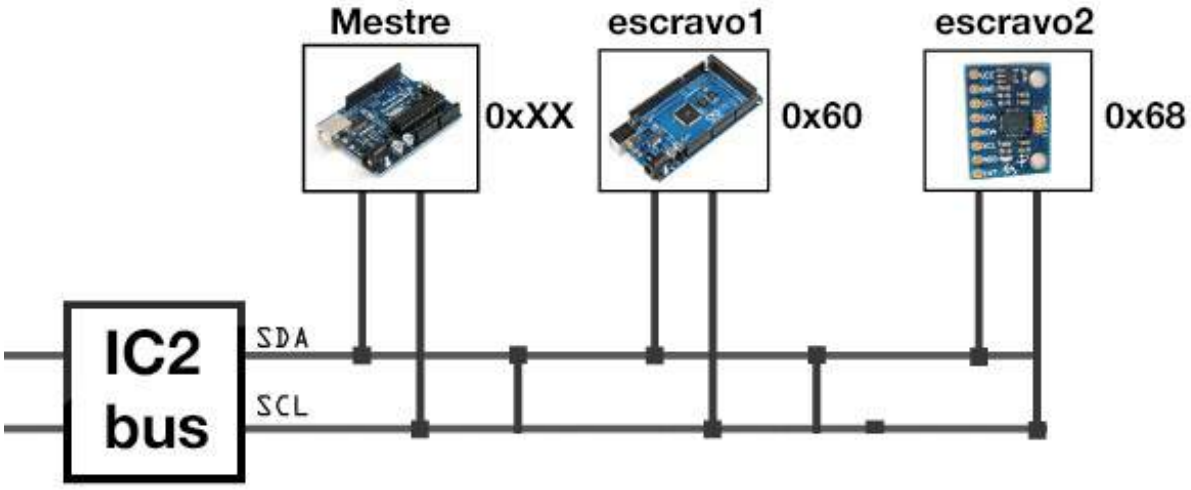

Figura 4.21: Estrutura de comunicação local entre os dispositivos.

```
//Receptor primário
//Modelo: Arduino UNO
//Objetivo: primeiro receptor, está conectado ao receptor wireless
//que é capaz de receber e enviar comandos, tem uma rotina de
//checagem por instruções do emissor, em caso positivo, envia
//as mesmas via IC2 para o receptor secundário.

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

//canais de comunicação sem fio
//como ele pode alternar entre emissor
//e receptor temos que indicar o endereço
//para leitura/escrita
//no emissor primario os valores estao invertidos
//pois sempre um for emissor o outro eh o receptor
RF24 radio(7, 8);
const byte rxAddr[6] = "00002";
const byte wxAddr[6] = "00001";

//Configuração para comunicação IC2 com o arduino mega
#include <Wire.h>
#define SLAVE_ADDRESS 0x60

pacote dados;
byte buff[2];

//Variáveis de hardware
int pin_buzzer = 10;
int status_buzzer = 0;//inicialmente desligado
unsigned long previousMillis = 0;
const long interval = 700;

//leitura da bateria
int sensorValue;
int led_conexao = 4;

....
```

Algoritmo 11: Primeira parte do código do receptor primário

```
void setup()
{
  //IC2
  Wire.begin();

  //Serial
  while (!Serial);
  Serial.begin(9600);

  //Wireless
  //inicia as configurações para funcionar como
  //um receptor
  radio.begin();
  radio.openWritingPipe(wxAddr);
  radio.openReadingPipe(0, rxAddr);
  radio.startListening();

  //Hardware
  pinMode(led_conexao, OUTPUT);
  pinMode(pin_buzzer, OUTPUT);
  pinMode(A0, INPUT);
}

//Função para enviar uma resposta para o emissor primário
//eh feito a troca para função de "emissor", monta-se
//o pacote de dados e eh feito algumas tentativas de
//envio, dado que o emissor original também precisa
//de um tempo para trocar de contexto e tornar um receptor.
void enviaMsg(int cmd, int valor) {
  //Serial.println("Enviando resposta");
  radio.stopListening();
  dados.cmd = cmd;
  dados.valor = valor;
  //tentativas de enviar ao emissor original
  for (int i = 0; i < 10; i++) {
    radio.write(&dados, sizeof(dados));
  }
  //delay(1);
  radio.startListening();
}

void buzzer() {
  tone(pin_buzzer, 4000, 300); //freq = 4000hz
}

....
```

Algoritmo 12: Segunda parte do código do receptor primário

```
//Função para enviar uma mensagem para o arduino mega
//(responsável pelo controle dos motores)
void enviaIC2() {
  Serial.println("enviando");
  //constroi-se o pacote de dados montando um inteiro
  //de 2 bytes, sendo cada byte formado pelo comando
  //e por um valor associado:
  int valor_pacote = dados.cmd * 1000 + dados.valor;
  //Serial.println(dados.cmd);
  //Serial.println(dados.valor);

  String pacote_ic2 = String(valor_pacote);
  Wire.beginTransmission(SLAVE_ADDRESS);
  for (int x = 0; x < 4; x++) {
    Wire.write(pacote_ic2.charAt(x));
  }
  //Serial.println(pacote_ic2);
  Wire.endTransmission();
}


void loop()
{
  //trata do estado em que o buzzer está como "ligado"
  //para nao prender a execução em um delay é utilizado
  //o conceito de tempo passado, se for verificado que um intervalo
  //de tempo de execucao ja foi atingido entao executa-se a ação,
  //uma operação portanto não bloqueante.
  if (status_buzzer == 1) {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
      previousMillis = currentMillis;
      buzzer();
    }
  }

  ....
```

Algoritmo 13: Terceira parte do código do receptor primário

```
//Descrição do loop: Mantém uma checagem contínua pelo sinal
  //wireless vindo do emissor primário,ao receber um comando
  //checa o tipo,caso seja simples executa a tarefa ou transmite
  //para o receptor secundário, caso seja composto,
  //produz os dados necessários e faz o envio de retorno ao
  //emissor primário.
  if (radio.available()) //requisição de mensagem recebida
  {
    radio.read(&dados, sizeof(dados));
    //int tmp = dados.cmd;
    //dados.cmd = dados.valor;
    //dados.valor = tmp;
    //Serial.print("\n cmd: ");
    //Serial.println(dados.cmd);
    //Serial.print("\n valor: ");
    //Serial.println(dados.valor);
    //Serial.println("Rprimário recebido: " + dados.cmd);
    //Serial.println("valor: " + dados.valor);
    //Serial.println(dados.cmd);

    //avalia o tipo do comando
  if (dados.cmd <= 124) { //comandos simples

      switch (dados.cmd) {
          case 14: //Acionamento do buzzer
      status_buzzer = 1;
          break;
          case 15: //Desligamento do buzzer
      status_buzzer = 0;
          break;
          case 1: //Aciona motor1
      enviaIC2();
          break;
          case 2: //Aciona motor2
      enviaIC2();
          break;
          case 3: //Aciona motor3
      enviaIC2();
          break;
      case 4: //Aciona motor4
      enviaIC2();
          break;
          case 6: //Acionar todos os motores em uma velocidade
      enviaIC2();
          break;
          case 7: //Calibrar motores
      enviaIC2();
          break;
          case 8: //Liberar motores
      enviaIC2();
          break;
          case 9: //trava motores
      enviaIC2();
          break;
      }
    }
  ....
```

Algoritmo 14: Quarta parte do código do receptor primário

```
else { // >=125 comandos compostos
//precisa gerar uma resposta e enviar de volta
//ao emissor primário.

      switch (dados.cmd) {
        case 125: //Conexao inicial
          digitalWrite(led_conexao, HIGH);
          enviaMsg(125, 0);
          break;

        case 126: //Ler carga da bateria
          sensorValue = analogRead(A0);
          float voltage = sensorValue * (5.0 / 1023.0);
      //ajustar casas decimais para emular um inteiro
          voltage = voltage * 100;
          int volt_tmp = voltage;
          enviaMsg(126, volt_tmp);
          break;
      }
    }
    /*
        if (dados.valor == 10) {
          enviaMsg(); //envia uma mensagem de volta ao controlador
        }
        else if (dados.valor == 15) {
          enviaIC2(); //envaminha o comando ao receptor secundário
        }
    */

  }
}
```

Algoritmo 15: Quinta parte do código do receptor primário

4.4.6 Receptor secundário

O receptor secundário executa rotinas de controle automatizado dos motores e recebe comandos pela comunicação IC2. Quando uma mensagem é recebida ocorre uma interrupção e essa chamada é atendida, depois retorna a execução do `loop`.

O controle de estabilidade é feito por leituras do acelerômetro e por uma estratégia chamada de PID (`proportional integral derivative`). O controle consiste em avaliar o erro, no caso o desvio identificado em cada eixo e com isso gerar uma saída que define uma ação corretiva, no caso essa ação consiste em aumentar ou diminuir a velocidade de um ou mais motores. O controle é feito por eixos, no caso temos um controle para o eixo y e outro para o eixo x, a partir da leitura definimos o erro (diferença entre o valor lido e o valor de base -estado considerado ideal-), que é o valor proporcional; Salvamos a leitura do estado anterior e

comparamos com a leitura atual, esse é o valor derivativo; Por fim mantemos um somatório dos valores lidos para gerar o valor de integração. A interação desses três fatores resulta no PIDy e PIDx, é feito uso de algumas constantes (K) para "pesar"cada uma dessas variáveis na conta final.

Nesse projeto a estratégia também foi levemente modificada para atender as necessidades da aplicação real, no caso o controle PID gera uma saída que incrementa/decrementa um valor de correção, após um intervalo de tempo é verificado se o erro diminuiu ou se é zero, se ainda estiver com desvio novamente o fator de correção é aumentado (positivamente ou negativamente); a partir do momento que não é gerado mais um valor de erro nenhuma ação é tomada, ou seja, cada motor deve estar operando com uma velocidade nominal diferente, pois a soma gerada não é perdida, isso ocorre porque em uma condição de equilíbrio não é apenas garantir que todos os motores estejam a uma mesma velocidade, algumas vezes podemos ter diferenças de performance no harware e principalmente uma desproporção na distribuição de peso da estrutura (essa é ideia, de que o VANT seja capaz de manter uma estabilidade de maneira adaptável).[8]

Os eixos adotados no código são os mesmos do mostrado na figura 4.3. Uma ilustração de como o desvio é identificado para um eixo está na figura 4.22.

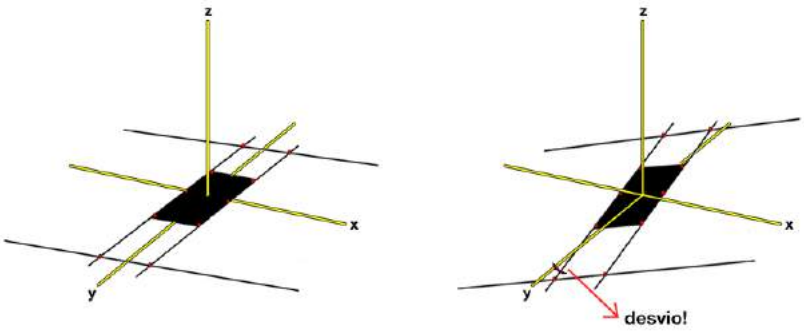

Figura 4.22: Ilustração de um desvio detectado pelo acelerômetro.

Um problema encontrado foi na operação para obter o ângulo de desvio, a princípio esse seria um item fácil a ser resolvido, mas demandou mais esforço do que o esperado; o módulo MPU 6050 conta com um acelerômetro e um giroscópio (este não utilizado no projeto), foi possível extrair o valor do erro (no ângulo do VANT) filtrando valores que são considerados

[8]Uma boa explicação para iniciantes sobre o conceito e aplicação do PID pode ser visto em: https://www.youtube.com/watch?v=txftR4TqKYA/

"válidos"pela função: `map(gy, -17000, 17000, 0, 179)`, o intervalo `[-17000, 17000]` compreende valores que se aproximam aos 180 graus (no qual são mapeados). O código desenvolvido teve como base uma versão para controle de servomotores.[9]. A versão de testes para os dois eixos do VANT pode ser vista em código 16.

[9] O código base para leitura do angulo de desvio pode ser visto em: http://lukagabric.com/arduino-mpu6050/

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 mpu;
int16_t ax, ay, az;
int16_t gx, gy, gz;

int valX;
int prevValX;
int valY;
int prevValY;
int incomingByte = 0;

int valX2;
int valY2;

int ledMpu = 8;
int statusCon = 0;

void setup()
{
  Wire.begin();
  Serial.begin(38400);
  pinMode(ledMpu, OUTPUT);

  Serial.println("Initialize MPU");
  mpu.initialize();
  if (mpu.testConnection()) {
    Serial.println("Connected");
    digitalWrite(ledMpu, HIGH);
  }
  else {
    Serial.println("Connection failed");
  }
}

void loop()
{
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  valX = map(ax, 17000, -17000, 0, 179);
  valY = map(ay, -17000, 17000, 0, 179);

  Serial.print("aX: ");
  Serial.print(valX);
  Serial.print(", ");
  Serial.print("aY: ");
  Serial.println(valY);
}
```

Algoritmo 16: Código para operação do acelerômetro (MPU)

Na figura 4.23 vemos em detalhe o receptor secundário (que é ligado ao Arduino MEGA), a imagem do verso não está espelhada (ela já foi invertida via software para mostrar como

se fosse a mesma visão nas duas imagens). Na figura 4.24 temos as ligações feitas na placa, note que o módulo MPU (acelerômetro) apesar de estar nesse circuito é ligado a hierarquia da comunicação, logo ele também é acessível pelo arduino uno que opera como `master` (na comunicação IC2 é necessário conectar os pinos *SDA* e *SCL* assim como o *GND* de todos os dispositivos envolvidos). Nos motores o pino do *VCC* não é utilizado (o cabo do ESC possui três entradas), apenas ligamos o pino de sinal (na porta lógica correspondente) e o gnd.

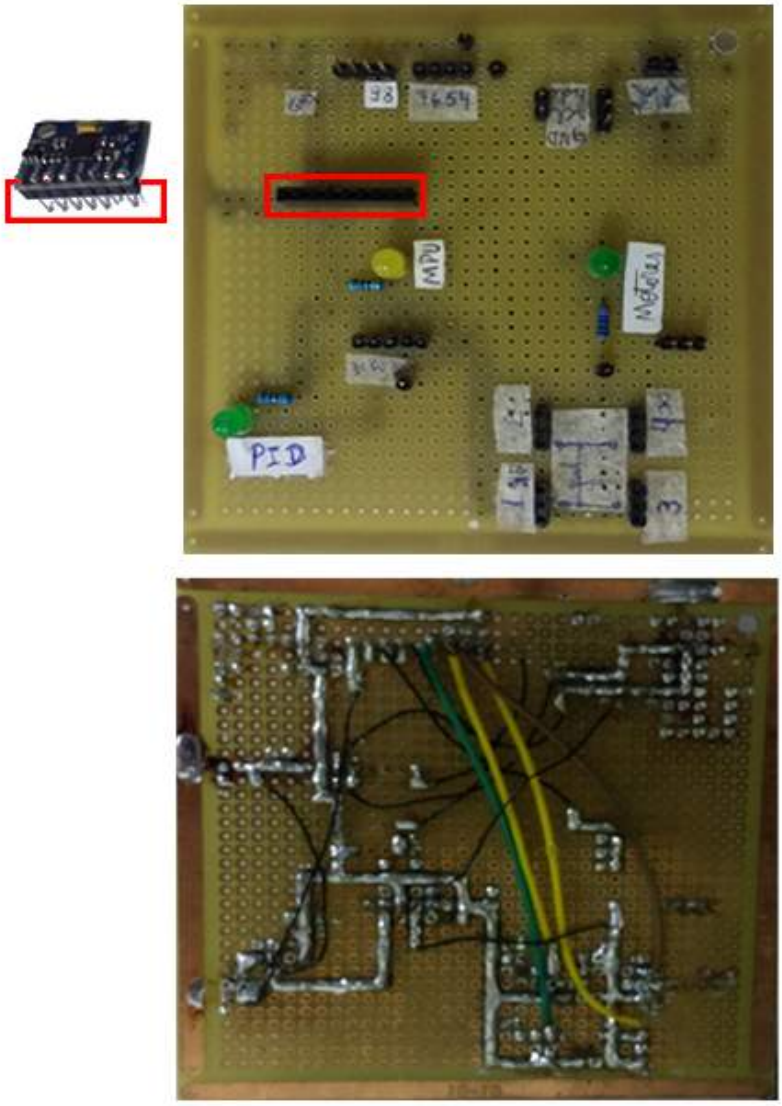

Figura 4.23: Detalhes da ligação dos componentes no receptor secundário.

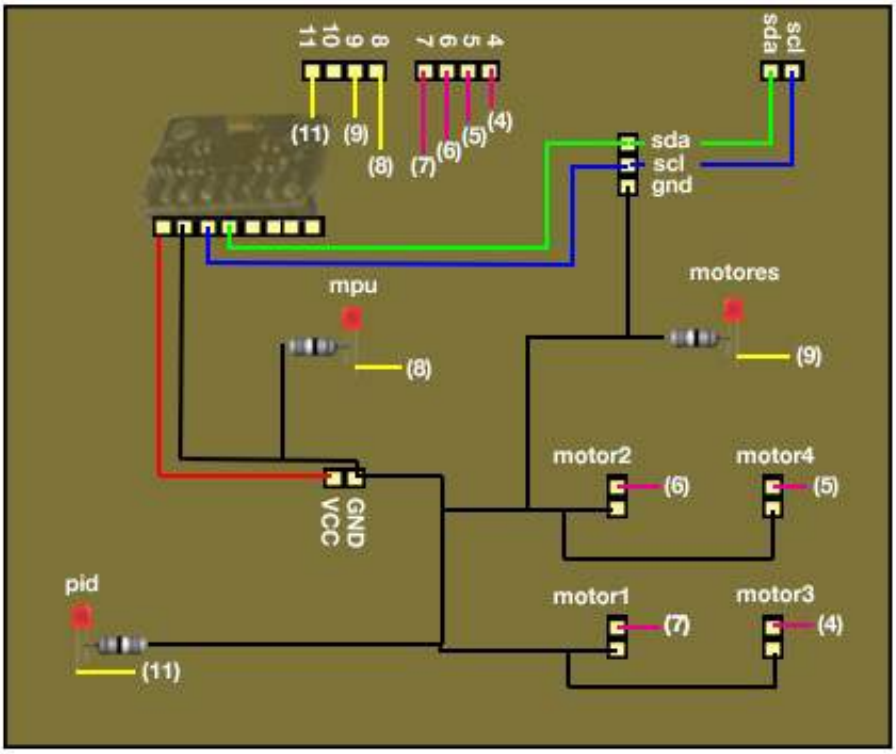

Figura 4.24: Esquema de ligações no receptor secundário.

```
#include <Wire.h>
#include <Servo.h>

#include "I2Cdev.h"
#include "pacote.h"
#include "MPU6050.h"

#define SLAVE_ADDRESS 0x60

pacote dados;
String data = "";

//Variáveis de controle dos motores
Servo motor1; //frente esquerda
Servo motor2; //frente direita
Servo motor3; //tras esquerda
Servo motor4; //tras direita

int flag_calibrar = 0;
int flag_operacao = 0;
int flag_ok = 0; //controle anti-rotação involuntária

int led_Motores = 9; //LED ligado: motores liberados

//PID: Controle de estabilização automático

//Velocidades: A velocidade de cada motor é definida por dois fatores:
//I: O emissor envia um valor base para o motor, no qual podemos
//controlar a altura do drone, já que é possível aumentar ou
//diminuir todas as intensidades.
//II: O valor calculado pelo PID, que define um valor de
//acrescimo ou decrescimo para cada motor afim de controlar
//a estabilidade

int vBase[4]; //velocidade base
int vPid[4]; //fator de acrescimo/decrescimo para cada motor

//MPU (acelerômetro/giroscópio)
MPU6050 mpu;

int16_t ax, ay, az;
int16_t gx, gy, gz;

int valX;
int prevValX;

int valY;
int prevValY;

int valX2;
int valY2;

int ledMpu = 8;
int led_pid = 11;

....
```

Algoritmo 17: Primeira parte do código de operação no receptor secundário.

```
//PID EIXO X
double setX = 0;
double prevX = 0;//leitura passada
double errorX = 0;

//K: "peso" de cada coeficiente
double kpX = 5.0,
       kiX = 0.0,
       kdX = 10.0;

//P: proporcional
//I: integral
//D: derivada
double Px = 0,
       Ix = 0,
       Dx = 0;

double PIDx = 0;

//Controle de tempo para Iy e Dy:
long prevProcX = 0;

//PID EIXO Y
double setY = 0;
double prevY = 0;//leitura passada
double errorY = 0;

//K: "peso" de cada coeficiente
double kpY = 7.0,
       kiY = 0.0,
       kdY = 10.0;

//P: proporcional
//I: integral
//D: derivada
double Py = 0,
       Iy = 0,
       Dy = 0;

double PIDy = 0;

//Controle de tempo para Iy e Dy:
long prevProcY = 0;

//Offset do MPU
int offX = 0;
int offY = 0;

int indiceMy = 1;
int indiceMx = 1;

float varTempo;
float reguladorAtt = 0;
float reguladorPrev = 0;
//FIM VARIÁVEIS DO PID

int tempOff=1;

....
```

Algoritmo 18: Segunda parte do código de operação no receptor secundário.

```
void setup() {

  //MPU
  pinMode(ledMpu, OUTPUT);
  pinMode(led_pid, OUTPUT);

  Serial.println("Initialize MPU");
  mpu.initialize();
  if (mpu.testConnection()) {
    Serial.println("Connected");
    digitalWrite(ledMpu, HIGH);
  }
  else {
  //se nao ligar o LED do MPU é sinal de que
  //a inicialização falhou, então é possível
  //detectar essa falha visualmente antes de
  //operar os motores.
    Serial.println("Connection failed");
  }

  //Motores
  for (int i = 0; i < 4; i++) {
    vBase[i] = 65;
    vPid[i] = 0;
  }

  pinMode(led_Motores, OUTPUT);

  motor1.attach(7);
  motor2.attach(6);
  motor3.attach(4);
  motor4.attach(5);

  //Configuração comunicação IC2:
  Wire.begin(SLAVE_ADDRESS);
  Wire.onReceive(receiveEvent);

  Serial.begin(9600);
  Serial.setTimeout(50);

  //Variaveis de controle inicialização:
  dados.valor = 0;

  ....
```

Algoritmo 19: Terceira parte do código de operação no receptor secundário.

```
//Calibração dos motores
  motor1.writeMicroseconds(2100);
  motor2.writeMicroseconds(2100);
  motor3.writeMicroseconds(2100);
  motor4.writeMicroseconds(2100);
  delay(3);

  motor1.writeMicroseconds(800);
  motor2.writeMicroseconds(800);
  motor3.writeMicroseconds(800);
  motor4.writeMicroseconds(800);
  delay(2);

  motor1.writeMicroseconds(1500);
  motor2.writeMicroseconds(1500);
  motor3.writeMicroseconds(1500);
  motor4.writeMicroseconds(1500);
  delay(2);

  motor1.writeMicroseconds(2100);
  motor2.writeMicroseconds(2100);
  motor3.writeMicroseconds(2100);
  motor4.writeMicroseconds(2100);
  delay(2);

  motor1.write(65);
  motor2.write(65);
  motor3.write(65);
  motor4.write(65);
  delay(2);
}
//fim do setup
```

Algoritmo 20: Quarta parte do código de operação no receptor secundário.

```
void loop() {

  if (tempOff == 1) {

    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    offX = map(ax, 17000, -17000, -50, 50);
    offY = map(ay, -17000, 17000, 50, -50);

    offX = (-1)*offX;
    offY = (-1)*offY;

    Serial.print("Offx:");
    Serial.println(offX);

    Serial.print("Offy:");
    Serial.println(offY);
    tempOff = 0;
  }

  //Loop: realiza operações de calibração nos motores (constitui uma
  //das ações críticas de segurança para evitar casos em que
  //       os motores hajam de maneira inapropriada (operando em
  //velocidade maxima sem controle por exemplo).

  //Antes de destravar os motores o loop fica executando a calibração
  /dos motores, quando se libera os motores essa operação será
  //executada apenas quando enviado um comando pelo painel de controle.
  //Ao liberar os motores a operação padrão será o controle de estabilidade
  //realizado pelo PID utilizando os valores do acelerômetro.
  //intercalado a essa operação seram executados operações de controle
  //manual (controlado pelo painel de controle).

  //flag de calibrar: indica a prioridade da operação de calibação (mantendo
  //os motores na velocidade mínima/desligados)
  //flag de ok: se for 0 indica que nao houve comandos que utilizam os
  //motores do drone.
  if (flag_calibrar == 1 || flag_ok == 0) {
    //calibrar motores

    //Serial.println("calibrando");
    flag_calibrar = 0;

    motor1.write(65);
    motor2.write(65);
    motor3.write(65);
    motor4.write(65);
    delay(2);

  }

  ....
```

Algoritmo 21: Quinta parte do código de operação no receptor secundário.

```
  //flag de operação: indica execução de algum comando que
2 //utiliza algum dos motores
  //flag de ok: indica que os motores estao liberados por software
4 else if (flag_operacao == 1 && flag_ok == 1) {
    //nao precisa calibrar, entao vai processar a operação dos motores
6   flag_operacao = 0; //a flag eh valida para uma operação

8   if (dados.valor < 180 && dados.valor > 60) {
      //faixa de operação dos motores
10
      switch (dados.cmd) {
12
        case 1:
14        //Serial.println("acionando motor 1");
          motor1.write(dados.valor);
16        break;
        case 2:
18        //Serial.println("acionando motor 2");
          motor2.write(dados.valor);
20        break;
        case 3:
22        //Serial.println("acionando motor 3");
          motor3.write(dados.valor);
24        break;
        case 4:
26        //Serial.println("acionando motor 4");
          motor4.write(dados.valor);
28        break;

30      case 6:
          //Serial.println("acionando todos os motores");
32        /* motor1.write(dados.valor);
            motor2.write(dados.valor);
34          motor3.write(dados.valor);
            motor4.write(dados.valor);
36        */
          //Salva o valor de base no vetor de velocidades
38        for (int i = 0; i < 4; i++) {
            vBase[i] = dados.valor;
40        }

42        //Serial.println(dados.valor);
          delay(15);
44        break;
      }
46  }
}
48

....
```

Algoritmo 22: Sexta parte do código de operação no receptor secundário.

```
//Operação de estabilização do VANT
  else {

    reguladorAtt = millis() / 1000.0;

    //Serial.println(reguladorAtt);
    //Serial.println(reguladorPrev);

    if ((reguladorAtt - reguladorPrev) > 1) {
      digitalWrite(led_pid, HIGH);
      //O loop de correcaoe executa em intervalos
      //iguais ou maiores a 1segundo
      reguladorPrev = reguladorAtt;

      //Serial.println("Estabilizando PID!");
      //o calculo e a correção são feitos em intervalos
      //de tempo: isso porque dado um desvio é aplicado
      //um critério de correção somando e diminuindo
      //a velocidade de certos motores,
      //após o período pode ser que o ajuste não seja
      //mais necessário (está estável), caso não
      //então um ajuste em cima do ajuste anterior é feito.

      mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

      valX = map(ax, 17000, -17000, -50, 50);
      valX += offX; //correção de hardware,

      valY = map(ay, -17000, 17000, 50, -50);
      valY += offY;

  /*
            Serial.print("aX: ");
            Serial.print(valX);
            Serial.print(", ");

            Serial.print("Offx: ");
            Serial.print(offX);
            Serial.print(", ");

            Serial.print("aY: ");
            Serial.print(valY);
            Serial.print(", ");

            Serial.print("Offy: ");
            Serial.println(offY);

     */

   ....
```

Algoritmo 23: Sétima parte do código de operação no receptor secundário.

```
//======= calcular PID para eixo Y  =======//
      errorY = setY - valY;

      //quanto tempo levou desde a ultima vez que executou o PID?
      //tempo atual - ultimo tempo
      varTempo = (millis() - prevProcY) / 1000.0;
      prevProcY = millis();

      varTempo = 1;

      //Py
      Py = errorY * kpY;

      //Iy
      Iy += (errorY * kiY) * (varTempo);
      //conversao do tempo para segundos

      //Dy
      Dy = (prevY - valY) * kdY / (varTempo);

      //salva o valor da "leitura passada" para o proximo loop
      prevY = valY;

      PIDy = Py + Iy + Dy; // PIDy = 0 => Equilíbrio atingido!

      /*
                      Serial.print(" Py: ");
                      Serial.print(Py);

                      Serial.print(", Iy: ");
                      Serial.print(Iy);

                      Serial.print(", Dy: ");
                      Serial.print(Dy);

                      Serial.print(" => Valor PIDy: ");
                      Serial.println(PIDy);

      */

      //calcular o efeito do PID no eixo Y

      //PIDy >0 : Inclinação do eixo dos motores 1 e 2 pra cima
      //Ação: diminuir intensidade nos motores 1 e 2,
      //acrescentar nos motores 3 e 4
      int factor = 10;

      PIDy = PIDy / factor;
      //PIDy = 0;
      if (PIDy > 0) {

        vPid[2] += PIDy;
        vPid[3] += PIDy;

        vPid[0] += (-1) * PIDy;
        vPid[1] += (-1) * PIDy;

      }

    ....
```

Algoritmo 24: Oitava parte do código de operação no receptor secundário.

```
//PIDy <0 : Inclinação do eixo dos motores 3 e 4 pra cima
      //Ação: diminuir intensidade nos motores 3 e 4,
      //acrescentar nos motores 1 e 2
      else if (PIDy < 0) {

        vPid[0] += (-1) * PIDy;
        vPid[1] += (-1) * PIDy;

        vPid[2] += PIDy;
        vPid[3] += PIDy;

      }

      //======= calcular PID para eixo X  =======//
      errorX = setX - valX;

      //quanto tempo levou desde a ultima vez que executou o PID?
      //tempo atual - ultimo tempo
      varTempo = (millis() - prevProcX) / 1000.0;
      prevProcX = millis();

      varTempo = 1;

      //Px
      Px = errorX * kpX;

      //Ix
      Ix += (errorX * kiX) * (varTempo); //conversao do tempo para segundos

      //Dx
      Dx = (prevX - valX) * kdX / (varTempo);

      //salva o valor da "leitura passada" para o proximo loop
      prevX = valX;

      PIDx = Px + Ix + Dx; // PIDx = 0 => Equilíbrio atingido!
/*
      Serial.print(" Px: ");
      Serial.print(Px);

      Serial.print(", Ix: ");
      Serial.print(Ix);

      Serial.print(", Dx: ");
      Serial.print(Dx);

      Serial.print(" => Valor PIDx: ");
      Serial.println(PIDx);
*/

....
```

Algoritmo 25: Nona parte do código de operação no receptor secundário.

```
//calcular o efeito do PID no eixo X
      //PIDx <0 : Inclinação do eixo dos motores 1 e 3 pra cima
      //Ação: diminuir intensidade nos motores 1 e 3,
      //acrescentar nos motores 2 e 4

      PIDx = PIDx / factor;
      //PIDx =0;
      if (PIDx < 0) {
        vPid[0] += PIDx;
        vPid[2] += PIDx;
        vPid[1] += (-1) * PIDx;
        vPid[3] += (-1) * PIDx;

      }
      //PIDx >0 : Inclinação do eixo dos motores 4 e 2 pra cima
      //Ação: diminuir intensidade nos motores 4 e 2,
      //acrescentar nos motores 1 e 3
      else if (PIDx > 0) {
        vPid[1] += (-1) * PIDx;
        vPid[3] += (-1) * PIDx;
        vPid[0] += PIDx;
        vPid[2] += PIDx;
      }
            for (int j = 0; j < 4; j++) {
              Serial.print("M");
              Serial.print(j);
              Serial.print(": ");
              //trava do intervalo de operação
              if((vBase[j] + vPid[j])<70){
                vPid[j] = (-1)*(vBase[j]-70);
              }
              else if((vBase[j] + vPid[j])>180){
                vPid[j] = 180 - vBase[j];
              }
              Serial.println(vBase[j] + vPid[j]);
            }
            Serial.println("<========>");

    //Atualiza o valor dos motores
      motor1.write(vBase[0] + vPid[0]);
      motor2.write(vBase[1] + vPid[1]);
      motor3.write(vBase[2] + vPid[2]);
      motor4.write(vBase[3] + vPid[3]);

      //FIM DA ESTABILIZAÇÃO
      digitalWrite(led_pid, LOW);
    }
    delay(100);

  } //fim do if para calculo do intervalo de tempo
}
  ....
```

Algoritmo 26: Décima parte do código de operação no receptor secundário.

```
//A função trata de eventos recebidos do mestre (pela hierarquia IC2)
//quando há um envio por parte dele o escravo (esse receptor)
//desvia para tratamento do dado recebido, que consiste em
//ativar um flag (como se fosse uma senha de banco) para o loop
//executar uma operação unicamente.
//e tb para salvar em uma variavel o valor do dado associado.
//a função foi previamente configurada no setup para esse tipo
//de tratamento.
void receiveEvent(int howMany) {
  data = "";
  while (Wire.available()) { //enqnto estiver disponivel bytes
    data += (char)Wire.read();
  }
  int valor_data = data.toInt();
  //Serial.println(valor_data);
  dados.cmd = valor_data / 1000;
  dados.valor = valor_data - dados.cmd * 1000;

  if (dados.cmd < 7) { //conjunto de comandos que vai
    //acionar um ou mais motores
    flag_operacao = 1;
  }
  else if (dados.cmd == 7) {
    flag_calibrar = 1;
  }
  else if (dados.cmd == 8) {
    //liberar flag para operar os motores
    flag_ok = 1;
    digitalWrite(led_Motores, HIGH);
  }
  else if (dados.cmd == 9) {
    //travar flag para operar os motores
    flag_ok = 0;
    digitalWrite(led_Motores, LOW);
  }
  //Serial.println(dados.cmd);
  //Serial.println(dados.valor);
}
//fim do código do receptor secundário e controle PID
```

Algoritmo 27: Décima primeira parte do código de operação no receptor secundário.

4.4.7 Alimentação do circuito

O circuito é alimentado por uma bateria de 9v, não foi utilizado o pino `Vin` mas sim a entrada `jack` nos dois arduinos. Foi colocado uma chave de *On/Off* (chave amarela na figura 4.25) para controle. Alternativamente o circuito oferece suporte a uma alimentação por fonte (no projeto: uma fonte comum de carregador com um regulador de tensão Lm2596s para 9v) que é utilizada para testes e depuração.

A bateria de lipo oferece saídas que são ligadas a um alarme que mostra visualmente a

carga de cada cédula e exibe um sinal sonoro quando atinge níveis críticos. Uma dessas saídas (a 1s de até 3.3v aproximadamente) é ligada a entrada analógica do receptor primário, a leitura analógica exibe no painel de controle um nível de porcentagem restante de bateria (essa conexão é controlada pela chave vermelha).

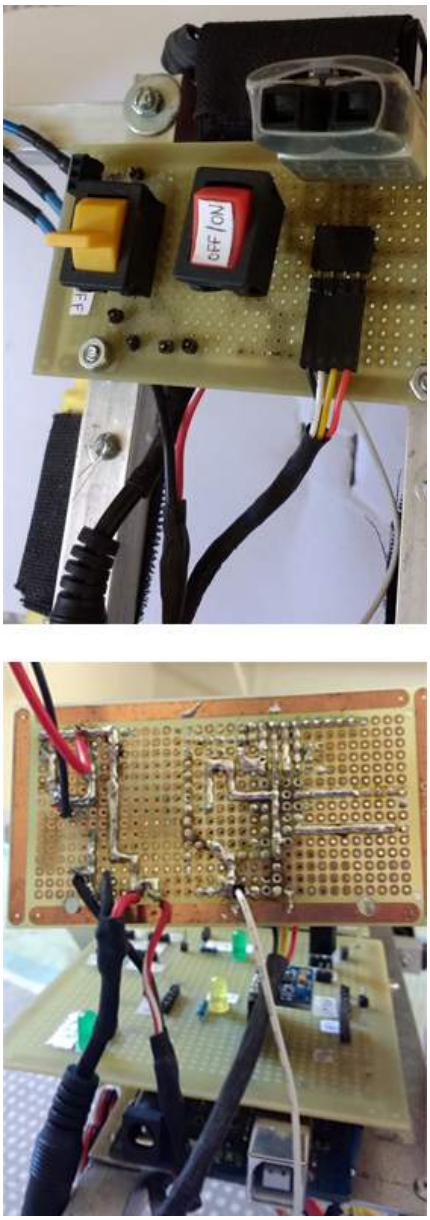

Figura 4.25: Alimentação do circuito.

5

Conclusão

A construção do projeto resultou do estudo e análise de diversas técnologias e práticas, foram diferentes softwares e hardwares, diferentes arquiteturas e modelos de prototipagem formando algo bem diversificado.

A contribuição pessoal e profissional do projeto não se resume ao nicho das aeronaves não tripuladas, mas abrange toda a área de computação embarcada e interação humano-computador; o modelo de comunicação sem fio e transferência dos dados pode ser usado para as mais diversas aplicações, assim como a hierarquia do hardware pode ser expandida, montando uma verdadeira rede de micro controladores em paralelo.

O desenvolvimento da estrutura física envolveu muitas considerações que puderam ser comprovadas posteriormente, com relação ao design na composição e peso dos materiais empregados, o uso do alumínio deu resultados satisfatórios (a estrutura final completa do VANT pesa aproximadamente 1100,00[kg]) mas futuramente pode ser substituída por uma construção em plástico (modelado por uma impressora 3D).

Os motores e a bateria corresponderam aos cálculos realizados e foram capazes de atender as necessidades básicas do VANT (decolar e permanecer no ar por um período considerável, apesar de que a adição de outra bateria aumentaria consideravelmente o tempo total mesmo sem causar grandes alterações no peso final).

A interface de controle garante um maior nível de personalização (poderia construir um banco de dados para coleta de dados específicos) mas também pode ter certas complementações como desenvolvimento de um controle analógico ou uso de um `smartphone` para um melhor controle aéreo.

Os micro controladores certamente foram suficientes para o processamento dos dados, apesar de ter sido utilizados duas unidades no VANT ainda sim podem ser considerados como uma opção viável, outras possibilidades como uso do `raspberry` podem ser estudadas e avaliadas.

O projeto como um todo foi um grande desafio, muitos testes e alterações até atingir o nível atual, a maior parte do esforço em desenvolver a arquitetura para interação entre todas os módulos garantindo que exista um processamento em tempo real -o mais fiel possível-, sem que algo tenha que ser propriamente sacrificado para isso. O VANT ainda será aprimorado com itens adicionais (um suporte para camera por exemplo, principalmente com um hardware para armazenamento de imagens) e também aprimorações no sistema de controle de estabilidade (no código é possível ver que há pesos para os parâmetros do PID, assim como uma variação de tempo definida para que seja feito os cálculos, esses valores devem ser ajustados com objetivo de atingir os melhores resultados possíveis, uma ação seria a substituição do módulo do acelerômetro, o utilizado funciona bem para um item de baixo custo mas em uma aplicação mais complexa seria ideal utilizar um que produzisse leituras mais corretas e precisas); A ideia é incrementar o projeto mantendo a arquitetura desenvolvida inicialmente e aprimorar pontos que resultem em melhoras significativas.

6

Produção ténico-cientfícica

Durante a preparação do projeto de desenvolvimento que foi executado no período de vingência houve a participação na 67ª reunião anual da SBPC em São Carlos, no qual foi apresentado um painel com o título: "Estudo de aplicações semiautônomas utilizando arduino". Posteriormente os dados levantados por esse estudo foram aplicados no projeto do VANT.

Figura 6.1: Certificado de participação na reunião da SBPC 2015.

7

Avaliação final

7.1 Auto avaliação

Com relação ao desenvolvimento e execução do projeto eu avalio como positiva, na medida em que reconheço o maior projeto já executado por mim, atualmente vejo crescimento de conceitos diretamente na minha área de conhecimendo (computação, comunicação, hardware e software em geral). O ciclo de formação de hipótese, concepção de idéias, construção, aplicação e testes serviu para agregar experiencia na condução e execução de um projeto completo e que, em alguns casos, não se havia uma referência a ser seguida mas sim a necessidade de desenvolver uma solução própria.

7.2 Avaliação do orientador

texto da avaliação final.exemplo .

São Carlos, ____ de _________________ de 2016.

Assinaturas:

____

Nome do(a) Bolsista: Caio Cesar Almeida Pegoraro

____

Nome do(a) Orientador(a): Edilson Reis Rodrigues Kato

Figura 7.1

Referências Bibliográficas

[1] MARGOLIS, Michael. Arduino Cookbook .ed. O'Reilly Media, 2011.

[2] McROBERTS, Michel, Arduino Básico, Ed. Novatec, 2011.

[3] MONK, Simon. Programação com Arduino: Começando com Sketches, Ed. Bookan, 2012.

[4] Arduino, <http://www.arduino.cc/>, acesso em acesso em 25 de Abril de 2015.

[5] "Veículos aéreos não tripulados prometem revolucionar mercado de geotecnologia", Massa Cinzenta, <http://bit.ly/2c1ptEK>, acesso em 26 de Abril de 2015.

[6] VANTs e RPA, <http://bit.ly/2bvq4Sd>, acesso em 26 de Abril de 2015.

[7] Visual Studio, <https://www.visualstudio.com/>, acessado em 26 de Abril de 2015.

[8] Bluetooth, < http://bit.ly/1Pj9caw>, acessado em 26 de Abril de 2015.

[9] FPV, <http://fpvbrasil.com.br/page/o-que-e>, acessado em 26 de Abril de 2015.

[10] ANAC, <http://www.anac.gov.br/>, acessado em 27 de Abril de 2015.

[11] WIRELESS, <http://bit.ly/2bT6XRs>, acessado em 27 de Abril de 2015.

[12] "Quadcopter Design",< http://bit.ly/2btvlIx>, acessado em 10 de março de 2016.

[13] "RF24 Driver release", Maniacal Bits, <http://bit.ly/2c7EfwH>, acessado em 10 de março de 2016.

[14] “Giroscópio GY-521”, O mundo da programação, <http://bit.ly/2bvqUJT>, acessado em 10 de março de 2016.

[15] “Controle PID em sistemas embarcados”, Embarcados, <http://bit.ly/2c1Dx0E>, acessado em 10 de março de 2016.

[16] “Quadcopter PID Explained and Tuning”, OscarLiang, <http://bit.ly/1QJgfun>, acessado em 10 de março de 2016.

[17] “Estabilizador de voo”, Drones personalizados, <http://bit.ly/2bKZ9hd>, acessado em 10 de março de 2016.