

Relatório Científico Final

Pesquisa e Desenvolvimento de Metodologia para Estimar a Distância Máxima Efetiva de Coleta de Informação de Inversão Térmica, Apontada por Estação Meteorológica com Drones, para Aplicação no Controle do Bicudo do Algodoero (*Anthonomus grandis*)”.

FAPESP - Treinamento Técnico Nível 2
Número do processo: 2019/15012-0

Instituição sede: Universidade Federal de São Carlos - UFSCar
Período de vigência: 01/08/2019 à 31/01/2020

Bolsista: Alcides Mignoso e Silva
Supervisores de treinamento: Regina Bernardina Johanna Hakvoort, Prof. Dr. Luciano de Oliveira Neris - UFSCar
(Colaborador)

São Carlos - SP
30/09/2019

Sumário

1	Introdução	3
1.1	Objetivos	3
2	Atividades realizadas	5
2.1	Raspberry, Sensores e Módulos	5
2.1.1	Raspberry	5
2.1.2	I2C e sensores HDC1050 e MS4525	5
2.1.3	Leitura dos dados	7
2.1.4	UART, GPRS e Wi-Fi	7
2.1.5	Pixhawk PX4	9
2.2	Ambiente de Desenvolvimento	10
2.2.1	ROS	10
2.2.2	MAVROS e o Sistema Operacional	10
2.3	QGroundControl	11
2.4	Gazebo e o HITL	12
2.5	Aplicação	13
2.5.1	Nós principais da aplicação	13
2.5.2	Inicialização do sistema no Linux	14
2.5.3	Checagem do sistema e interfaces com o usuário	15
2.5.4	Modularização dos sensores	15
2.5.5	Rotina de coleta de dados	16
2.5.6	Sensor LIDAR	18
2.5.7	Dados coletados	18
2.5.8	Transmissão de dados em tempo real e aplicativo para celular	19
2.5.9	Codificação	21
2.5.10	Organização do sistema dentro do Linux	21
2.5.11	Distribuição do sistema	22
2.5.12	Utilização do sistema	22
2.6	Tutoriais	23
3	Conclusão	24
4	Referências Bibliográficas	25

1 Introdução

Este relatório descreve as atividades desenvolvidas pelo bolsista no período compreendido entre agosto de 2019 e janeiro de 2020 relativas ao projeto “Pesquisa e Desenvolvimento de Metodologia para Estimar a Distância Máxima Efetiva de Coleta de Informação de Inversão Térmica, Apontada por Estação Meteorológica com Drones, para Aplicação no Controle do Bicudo do Algodoeiro (*Anthonomus grandis*) [1]”.

Para a implementação do sistema proposto foram adotadas algumas tecnologias consolidadas, de fácil acesso e de código aberto. Para a aplicação do sistema a ser embarcado foi selecionada a placa Raspberry Pi 3 A+[14] conectada ao módulo de piloto automático Pixhawk/PX4 [9]. Esta placa é responsável por controlar o drone e por realizar a obtenção de dados de diversos sensores conectados via barramento I2C[13] à placa: um sensor de temperatura e umidade (HDC1050[4]) de baixo custo, baixo consumo energético e de alta acurácia e um transdutor de pressão (MS4525[8]) para medir a velocidade do ar.

O sistema operacional adotado à primeira instância foi o Raspbian Jessie [15]. Por causa de complicações em relação a compatibilidade das tecnologias utilizadas com o sistema operacional, o Raspbian foi posteriormente substituído pelo Ubuntu Mate 18.04[21].

A partir da seleção do piloto automático do drone foi realizada a escolha de uma tecnologia que permitisse implementar as ações de controle do drone durante o voo em pontos estratégicos para coleta de dados. Para o desenvolvimento do software do sistema proposto foi adotado uma coleção de frameworks concebida especificamente para o desenvolvimento e controle de robôs denominado de ROS (Robot Operating System)[18][19]. Para a comunicação com o piloto automático o padrão disponível pela plataforma Pixhawk/PX4 é o protocolo de mensagens MAVLink[6].

Para facilitar a utilização do MAVLink o pacote MAVROS[7] para o ROS foi selecionado. O MAVROS é a peça fundamental no desenvolvimento do projeto por prover estruturas consolidadas para realizar a comunicação do ROS com o drone, através do protocolo de mensagens MAVLink.

1.1 Objetivos

O objetivo principal do projeto é desenvolver um sistema capaz de monitorar o fenômeno da inversão térmica por meio de um drone simulando uma torre meteorológica para medição em tempo real de dados meteorológicos georreferenciados. Para uma eficiente coleta de dados o drone deve realizar a seguinte sequência de manobras em cada “waypoint”:

- Descer o drone até obter uma altitude válida do altímetro a laser (LIDAR). Posicionar o drone a uma altitude em relação ao solo de 2 metros;
- Girar o drone em seu eixo vertical no sentido horário, e coletar a temperatura e a velocidade do vento a uma taxa de coleta adequada. A rotação completa do drone deve ser realizada em cinco segundos.
- Alterar a altitude do drone para 5 metros em relação ao solo sobre a mesma posição;

- Girar o drone em seu eixo vertical no sentido horário e coletar a temperatura e a velocidade do vento a uma taxa de coleta adequada. A rotação completa do drone deve ser realizada em cinco segundos.
- Alterar a altitude do drone para 10 metros em relação ao solo sobre a mesma posição;
- Girar o drone em seu eixo vertical no sentido horário e coletar a temperatura e a velocidade do vento a uma taxa de coleta adequada.

A rotação completa do drone deve ser realizada em cinco segundos, totalizando inicialmente 50 amostras em cada coleta - esse valor foi tornado dinâmico com base no tempo que o drone demora para realizar a coleta dos dados. Além disso, o deslocamento do drone entre os pontos deve ser realizado a uma altura de 30 metros em relação ao solo, para evitar obstáculos. A coleta da velocidade do vento deve ser atrelada aos dados de uma bússola magnética atrelada drone. Por fim, a direção e a velocidade do vento devem ser determinadas com base na diferença das velocidades medidas durante a coleta, podendo ser determinadas a partir da maior velocidade do vento medida durante a rotina.

Em cada ponto de coleta de dados ("waypoint") o drone deve realizar a rotina descrita acima e armazenar os dados coletados em um dispositivo externo (pendrive) conectado a placa. Os dados, no final de cada ponto, devem ser publicados via HTTP no endereço local da placa na rede Wi-Fi para serem acessados pelo aplicativo mobile citado anteriormente. Dessa forma, a previsão da inversão térmica e os dados coletados podem ser acessados em tempo real pelo usuário através de um dispositivo celular.

2 Atividades realizadas

Todas as atividades propostas no projeto submetido foram realizadas. Diversos empecilhos em relação a compatibilidade de sistemas e a impossibilidade de utilizar certas ferramentas para a realização de algumas tarefas foram encontradas. Dessa forma, foi demandado um grande tempo para realizar à configuração do setup de desenvolvimento e acertar todos os detalhes das bibliotecas, do sistema operacional e do hardware adotado.

2.1 Raspberry, Sensores e Módulos

2.1.1 Raspberry

A placa Raspberry Pi 3 A+, Figura 1, é um computador de baixo custo do tamanho de um cartão de crédito. Esse computador pode ser utilizado com periféricos normais (teclado, mouse e monitor) ou através de conexão SSH (Secure Shell) [20] por outro computador na mesma rede. Uma Raspberry Pi 3 A+ deve ser instalada no drone e a ela conectados sensores para coleta de dados e para envio de rotinas pré-programadas de comportamento ao drone.

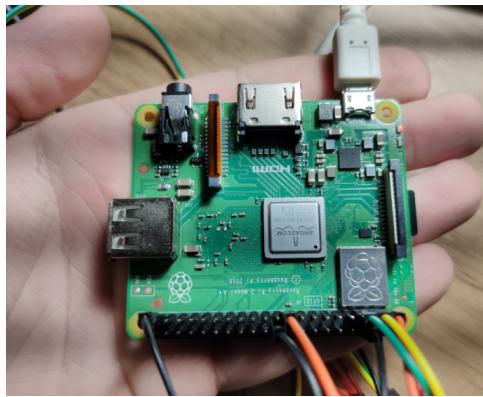


Figura 1: Raspberry Pi 3 A+

2.1.2 I2C e sensores HDC1050 e MS4525

A conexão dos sensores à placa Raspberry Pi 3 A+ é realizada através da tecnologia I2C (Inter-Integrated Circuit). O protocolo I2C é composto por dois tipos de dispositivos, um chamado de mestre e o outro de escravo. O mestre é a unidade de controle encarregada de controlar todos os escravos, dispositivos ligados ao barramento. A placa Raspberry Pi 3 A+ possui suporte à tecnologia I2C e atua como um dispositivo mestre. Os demais sensores conectados à placa atuam como dispositivos escravos.

Para a utilização da porta I2C da placa Raspberry Pi 3 A+ é necessário inicialmente habilitá-la. Dentro de um sistema compatível com a placa, é necessário executar o programa utilitário de ferramentas da Raspberry, chamado de raspi-config[12]. A depender do sistema operacional adotado, o programa vem pré-instalado e, caso não venha, basta instalá-lo. Ao chamar o mesmo pelo

terminal, com privilégios de administrador, uma tela de configuração é aberta. Então, basta habilitar a I2C, como mostrado nas Figuras 2 e 3.

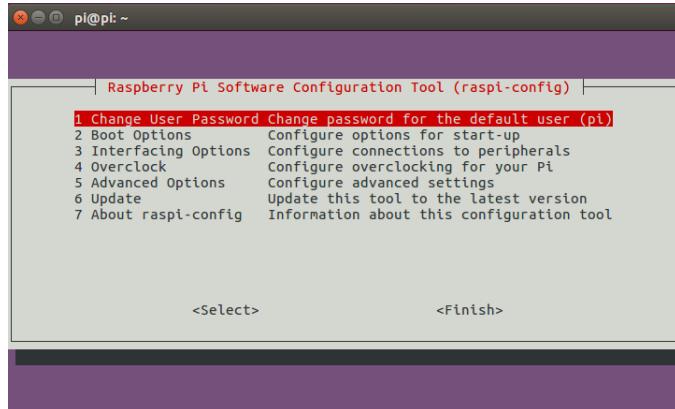


Figura 2: Habilitando a i2c - etapa 1

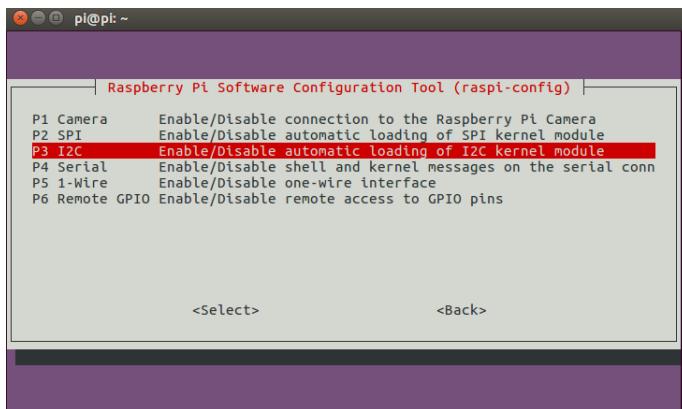


Figura 3: Habilitando a i2c - etapa 2

O dispositivo HDC1050 é um sensor de temperatura e umidade de baixo custo energético e alta precisão. O dispositivo MS4525 é um transdutor de pressão responsável por coletar dados relacionados a velocidade e direção do vento. Ambos os sensores estão conectados ao barramento serial I2C da Raspberry e podem ser acessados a qualquer momento para coleta de dados.

Em alguns casos, pode acontecer dos dois sensores I2C estarem mapeados no mesmo endereço da rede, o que torna necessário a disponibilização de outra rede ou a alteração do endereço. Contudo, varrendo todos os endereços da rede I2C com os sensores conectados, verificou-se a existência de dois dispositivos mapeados em endereços diferentes, o que confirma que não houve conflito. O sensor de temperatura foi mapeado para o endereço 0x40 e o sensor utilizado para medir a velocidade e direção do vento foi mapeado para o endereço 0x28, como pode-se observar na Figura 6. A conexão física dos sensores com a placa pode ser observada na Figura 7.

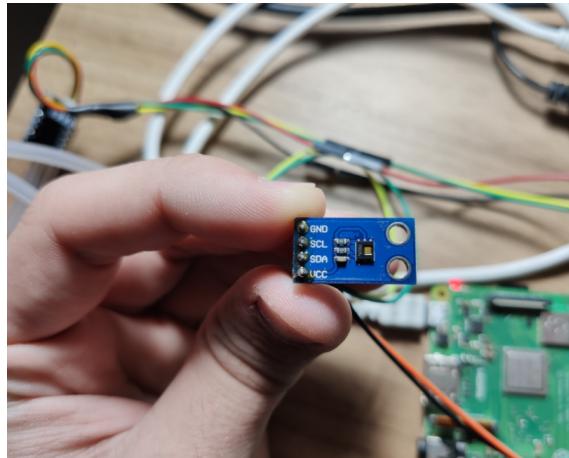


Figura 4: Sensor HDC1050

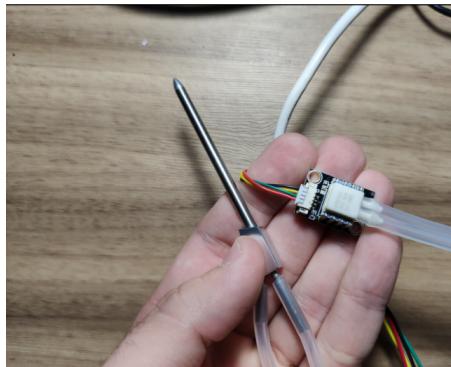


Figura 5: Sensor MS4525

2.1.3 Leitura dos dados

Diversos são os modos pelos quais é possível acessar os dados dos sensores. Neste projeto foi adotada a linguagem C por sua performance superior à de outras linguagens consideradas. Junto a ela, deve ser utilizado algumas bibliotecas externas da linguagem, sendo elas: a biblioteca i2c do pacote Linux (linux/i2c.h) e a biblioteca ioctl (sys/ioctl) . Por questões de facilidade, foram utilizados mecanismos diferentes dentro da linguagem para a leitura dos dados de ambos os sensores. É importante relatar que é necessário, após a coleta dos dados via acesso ao endereço I2C, um tratamento dos mesmos, conforme é detalhado no “datasheet” de cada sensor.

2.1.4 UART, GPRS e Wi-Fi

Além dos sensores de temperatura e velocidade do ar, a primeira versão do sistema necessitava de um GPRS conectado a placa para transmissão dos dados coletados para uma central. Para conexão do mesmo, era necessária uma porta para comunicação serial via UART.

```

pi@pi:~$ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- 28 -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- --

```

Figura 6: Scan na i2c - i2cdetect

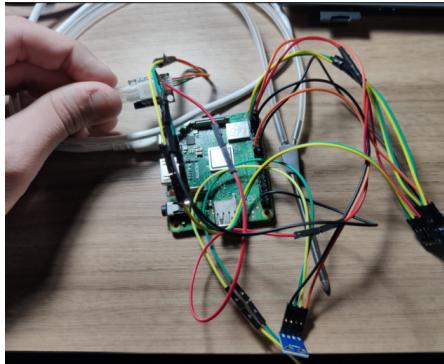


Figura 7: Conexão física dos sensores com a placa

A placa utilizada no projeto, a Raspberry 3 A+, possui duas portas UARTs. O plano inicial era utilizar as duas. A primeira, e mais robusta, por padrão é ligada ao módulo de Bluetooth, e a segunda e menos robusta, é chamada de mini UART. Essa porta vem previamente disponível para utilização, mas é possível alterar a porta utilizada pelo módulo Bluetooth fazendo com que ele utilize a mini UART, assim deixando a porta principal livre para ser utilizada. Pode-se, também, desabilitar o módulo Bluetooth.

Para utilizar a porta mini UART, é necessário alterar os arquivos do sistema operacional. Primeiramente, é necessário alterar o valor da variável “enable-uart” de 0 para 1 no arquivo */boot/config.txt* e desativar o console manualmente apagando o que se refere ao mesmo dentro do arquivo */boot/cmdline.txt*. Para transferir o módulo Bluetooth da UART para a mini UART, é necessário alterar o parâmetro “dtoverlay”, também em */boot/config.txt*, para “*pi3-miniuart-br*”. É importante ressaltar que essas configurações devem ser feitas independentemente do sistema operacional adotado.

A porta UART principal foi utilizada para realizar a conexão da placa com a Pixhawk para enviar e receber dados ao PX4 via Mavlink. A segunda porta UART seria utilizada para transmissão de dados via GPRS.

Entretanto, após diversas tentativas e pesquisa em fóruns com pessoas especializadas, foi constatado que infelizmente não é possível utilizar as duas UARTs simultaneamente, uma vez que elas compartilham dos mesmos pinos físicos GPIO (14 e 15). Por esse motivo, optamos por alterar a proposta da transferência de dados em tempo real para utilizar Wi-Fi ao invés do GPRS. Essa proposta será melhor discutida em um tópico posterior.

```
enable_uart=1  
dtoverlay=pi3-miniuart-bt
```

Figura 8: Parâmetros no Arquivo /boot/config.txt

2.1.5 Pixhawk PX4

Um módulo de controlador de voo Pixhawk PX4 deve ser conectada à placa sendo o responsável por prover e gerenciar o voo do drone.

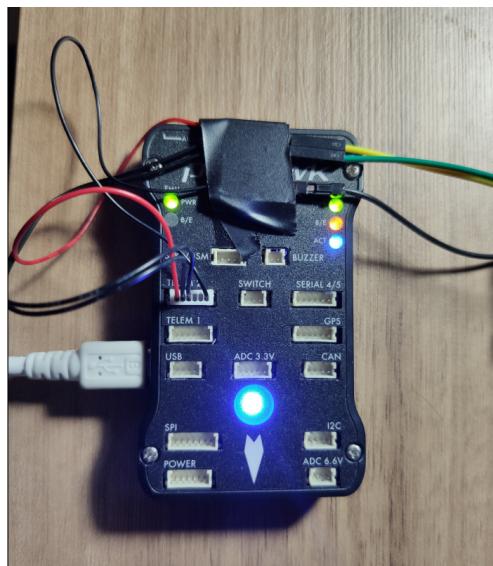


Figura 9: Pixhawk PX4

2.2 Ambiente de Desenvolvimento

Uma vez que todas as configurações em relação aos sensores e módulos tiveram sido realizadas, foi iniciada a configuração do ambiente de trabalho.

2.2.1 ROS

Primeiramente, é necessário realizar a instalação do ROS no sistema. Como citado anteriormente, foi utilizado como ponto de partida o sistema operacional Raspbian. Entretanto, na página oficial do ROS o sistema não consta como compatível por conta da placa utilizada no projeto possuir somente 512mb de memória RAM, sendo um dos modelos menos robusto da terceira geração. Alterar de sistema operacional seria algo complicado. Sendo assim, foi realizada uma tentativa falha de instalar o ROS no Raspbian de diversas formas diferentes e com ajuda dos fóruns relacionados ao tema. Depois de longas pesquisas, foi encontrado como solução viável instalar na placa uma imagem pública do Raspbian com o ROS pré-instalado, que foi encontrada em um dos fóruns anteriores.

2.2.2 MAVROS e o Sistema Operacional

Após a instalação do ROS, o próximo passo foi realizar o download e a compilação do pacote MAVROS. As instruções para a realização desse passo fazem referência ao repositório oficial do MAVROS e são relativamente simples. Entretanto, surgiram mais uma vez problemas com o sistema operacional. Algumas dependências necessárias para a instalação do MAVROS não existiam para o Raspbian, e portanto impossibilitavam a instalação. As tentativas de resolução das dependências manualmente falharam, o que forçou um estudo para a troca do sistema operacional.

Trocar o sistema operacional, apesar de teoricamente simples, se tornou um desafio. O único sistema que atendia todos os critérios de compatibilidade com as tecnologias necessárias e que poderia ser utilizado na placa foi o Ubuntu MATE. A versão adotada foi a 18.04 que é compatível com a versão mais recente do ROS (ROS Melodic). O problema citado anteriormente se deve ao fato da placa Raspberry Pi 3 A+ possuir 512mb de RAM, o que não é suficiente para fazer o primeiro boot do sistema. Esse fato faz com que a placa não conste na lista das placas que suportam o mesmo. Contudo, uma vez que o primeiro boot seja realizado em outra placa, compatível com o sistema e com a arquitetura da placa utilizada no projeto, ele tenderia a funcionar na placa A+ posteriormente. Sendo assim, foi utilizada uma placa mais robusta do Departamento de Computação da UFSCar que atendesse os critérios necessários.

A placa citada foi uma Raspberry 3 B+, e foi utilizada para fazer o primeiro boot, que consiste na configuração e instalação do sistema operacional. Vale ressaltar que houve uma tentativa de emular a placa com o QEMU[10], um emulador genérico de código aberto, para que fosse possível realizar a instalação em uma máquina normal. Emular somente o terminal foi simples, mas para emular o ambiente gráfico, necessário para configuração no primeiro boot, foi obtida uma série de erros e diversas panes no kernel por conta da ausência de um firmware adequado. Sendo assim, foi escolhido usar outra placa, onde foi realizada a configuração do cartão SD para posterior uso na placa A+.

Com o sistema configurado e o primeiro boot realizado, foi colocado o cartão na placa e foi obtido sucesso no uso do mesmo. O passo seguinte foi realizada

a instalação do ROS, que aconteceu de forma natural, uma vez que o sistema é compatível com o mesmo. Da mesma forma, também foi resolvido o problema da instalação do MAVROS, que também aconteceu de forma natural.

Resolvidos os problemas de instalação, iniciou-se o processo de compilação do MAVROS no ambiente (chamado, por padrão, de catkin workspace - catkin_ws). Após quase 2 horas de compilação, pouco progresso havia acontecido e a placa havia sobreaquecido, estando extremamente travada. Então, percebeu-se a necessidade de utilizar do conhecimento sobre emuladores que havia sido adquirido anteriormente. Foi emulado um ambiente compatível com a Raspberry utilizando o QEMU e, com o cartão SD no computador, compilou-se o MAVROS em uma máquina mais potente. A compilação ocorreu conforme esperado. É importante ressaltar que caso fosse compilar muitos pacotes pesados diversas vezes, seria inviável utilizar a placa escolhida por conta de seu poder de processamento. Contudo, uma vez compilado, não é necessário recompilar o MAVROS, sendo os pacotes desenvolvidos para o projeto pequenos o suficiente para serem compilados direta e separadamente na Raspberry. Com o MAVROS compilado, pôde-se iniciar a codificação dos códigos necessários.

2.3 QGroundControl

O QGroundControl é um poderoso "ground control station", de código aberto, que é responsável por prover controle total do voo e o planejamento das missões. Dentro do software é possível planejar uma missão e carregá-la no piloto automático do drone de forma fácil e intuitiva, para posterior utilização no drone, conforme mostra a Figura 9. Na missão deve-se adicionar os "waypoints" nos locais em que a coleta dos dados deve ocorrer.



Figura 10: Planejamento de uma Missão no QGroundControl

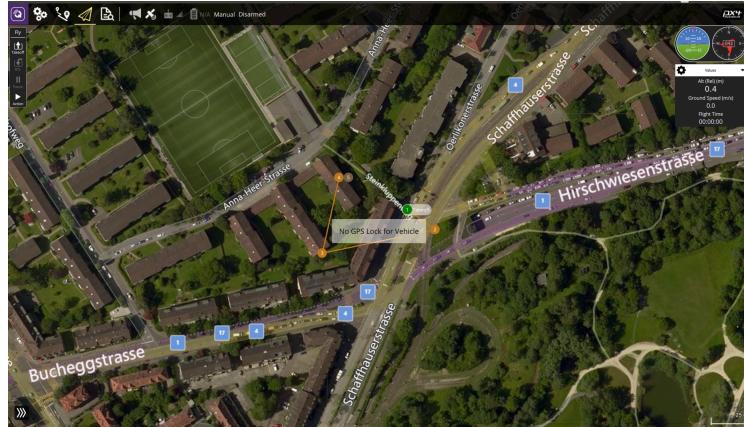


Figura 11: Interface Principal do QGroundControl

2.4 Gazebo e o HITL

Por questões de praticidade, torna-se muito complicado realizar os testes presencialmente com o drone a cada mudança no código. Dessa forma, buscou-se uma forma de simular o voo do drone para a realização de testes iniciais. Diversas são as opções quando se busca um simulador, porém o que mais se adequou a aplicação foi o Gazebo [3] versão 9. O Gazebo é um simulador de ambiente 3D para robôs que permite, através da estratégia HITL [5] (Hardware-in-the-loop), simular de forma eficiente e precisa o comportamento do drone. O Hardware-in-the-loop é um modo de simulação no qual o firmware do PX4 é executado no hardware do simulador de voo, o que permite testar a maior parte do código no hardware real em um ambiente simulado. Existem instruções para a realização da simulação HITL junto ao MAVROS nos manuais do PX4. É válido ressaltar que, por questões de compatibilidade, o sistema operacional adequado para realizar as simulações foi o Ubuntu 16.04, que é compatível tanto com o ROS Melodic, quanto com o MAVROS e o Gazebo.

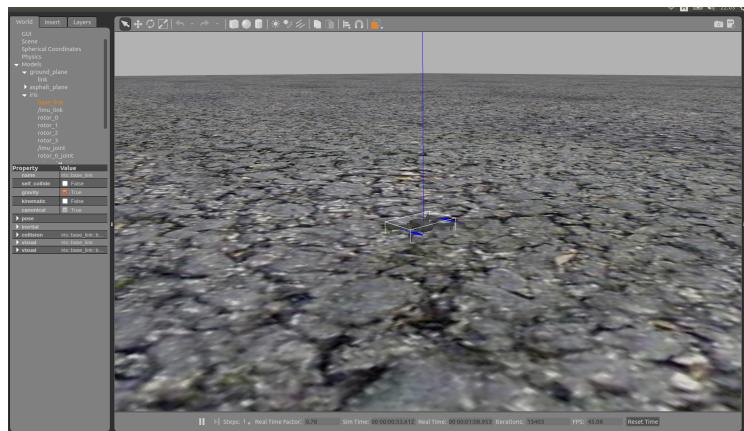


Figura 12: Interface Principal do Gazebo

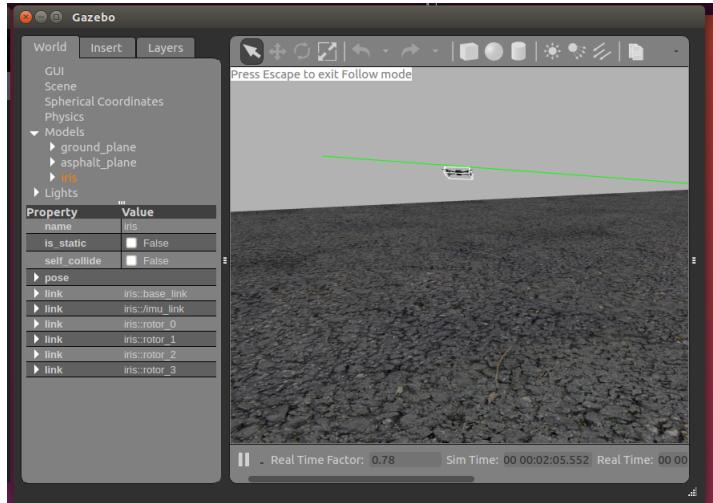


Figura 13: Drone em simulação no Gazebo

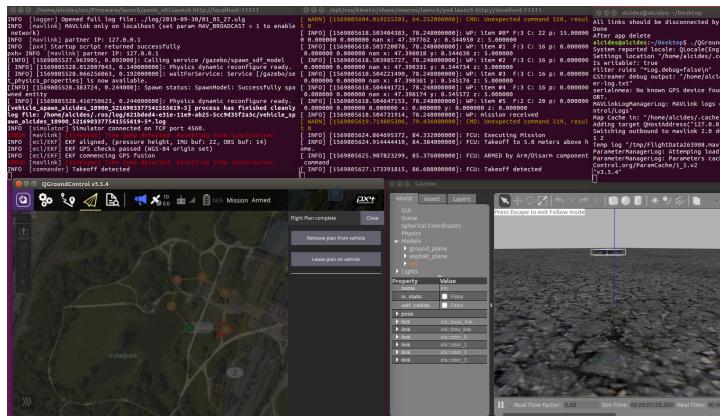


Figura 14: Simulação HITL

2.5 Aplicação

É importante ressaltar que o ROS é um sistema baseado em nós de comunicação, que conversam entre si utilizando os conceitos de tópicos (“publishers” e “subscribers”), serviços e clientes. Elaboramos, para a nossa aplicação, um modelo composto de cinco nós: o nó principal, responsável por “hospedar” os outros, dois nós que publicam constantemente o valor do sensor de temperatura e o de velocidade do ar, respectivamente, um nó da aplicação, responsável por enviar mensagens e controlar o drone durante as rotinas especiais, e um nó para fazer o teste do sistema durante a inicialização do mesmo.

2.5.1 Nós principais da aplicação

Para os nós dos sensores não foi encontrado material significativo na internet que pudesse cumprir os requisitos. Sendo assim, o código responsável por publicar as informações de cada sensor no tópico correspondente para ser aces-

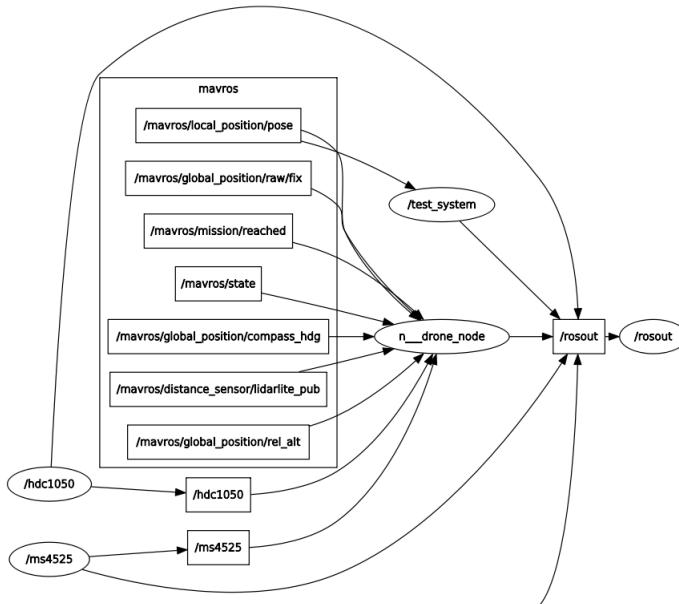


Figura 15: Conexão entre os nós do sistema e os tópicos utilizados

sado futuramente foi desenvolvido em sua totalidade. Ele consiste na união dos conceitos de leitura de dados da I2C com o ROS. Basicamente, existe para cada sensor um nó que publica constantemente os dados lidos dos sensores em um tópico, que pode ser acessado por qualquer outro nó.

Uma questão preocupante foi a possibilidade de haver conflitos na leitura dos dados dos sensores, uma vez que ambos estavam conectados ao mesmo barramento i2c. Entretanto, uma possível solução seria habilitar mais um canal i2c. Por sorte, após a realização de diversos testes, a leitura simultânea dos dados não foi um problema.

Por fim, foi desenvolvido o nó responsável pela rotina de coleta de dados. Esse nó tem o papel de identificar a chegada do drone em um ponto de parada e assumir o controle do mesmo, enviando mensagens para que ele execute minuciosamente a rotina que será apresentada.

2.5.2 Inicialização do sistema no Linux

Como mencionado anteriormente, a aplicação foi construída em sobreuma distribuição Linux (Ubuntu Mate) e, para que ela pudesse iniciar junto com o sistema operacional foi necessária a utilização de um recurso do Linux chamado de serviço.

O Linux funciona com base em serviços e, basicamente, um serviço é um programa executando em segundo plano. Os serviços podem ser habilitados e desabilitados, ligados e desligados a partir do gerenciador de serviços, que no caso é o "systemctl", e costumam iniciar junto com o sistema operacional. Dessa forma, foi criado um serviço que inicia junto com o sistema operacional chamando um *script* de inicialização da aplicação.

O *script* de inicialização da aplicação contém todas as chamadas necessárias para o sistema funcionar de forma correta. É feita a configuração das variáveis de ambientes necessárias para inicializar os nós do ROS, a montagem do pendrive como dispositivo externo que permite leitura e escrita por qualquer usuário, a conexão com a Pixhawk/PX4 e a inicialização do restante nós do sistema.

É válido ressaltar que o serviço criado chama-se "mavros", e está configurado dentro de um arquivo chamado "mavros.service" na pasta do Linux responsável pela manutenção dos serviços do sistema (/lib/systemd/system/mavros.service).

2.5.3 Checagem do sistema e interfaces com o usuário

Por ser um sistema embarcado, o sistema constituído pela placa acoplada no drone está suscetível a falhas de hardware. Como o sistema não possuía nenhuma interface direta com o usuário, foi necessária a adição de mecanismos que interagissem com o usuário para indicar possíveis erros.

Para evitar problemas na utilização da aplicação, foi desenvolvida uma rotina de checagem de erros que é executada anteriormente ao restante do sistema. A rotina consiste em checar a interoperabilidade dos sensores utilizados para medir a velocidade do vento e a temperatura e a umidade do ambiente, checar a existência de um pendrive conectado a placa com espaço disponível para a gravação dos dados, e checar se a placa está realmente conectada e trocando informações válidas com a controladora Pixhawk/PX4.

A rotina checagem do sistema indica o status da verificação ao usuário através de um LED de cor verde, que acende caso o sistema esteja funcionando corretamente e permanece apagado caso algum problema venha a ser detectado.

Um dos problemas detectados durante a etapa de testes está relacionado ao desligamento forçado da placa, seja por corte de energia ou travamento. Pelo sistema operacional utilizado trabalhar com *buffers*, a distribuição Linux utilizada deve fazer a sincronização dos dados em *buffers* para que possa ser desligado de forma correta. A sincronização é feita no Linux periodicamente com um intervalo de 30 segundos, porém ainda assim o sistema estava vulnerável a problemas. Dessa forma, foi adicionado junto ao LED de status do sistemas um botão de "poweroff" que quando pressionado os dados em *buffers* são sincronizados e o "shutdown" do sistema é realizado. É válido ressaltar que o sistema pode apresentar problemas caso seja desligado de forma incorreta, o que pode ser corrigido refazendo o setup do mesmo, como apresentado posteriormente.

2.5.4 Modularização dos sensores

Como citado anteriormente, o ROS trabalha com nós que comunicam-se entre si através de mensagens. Com fins de organização do código e seguir a proposta do framework, foi feita a modularização dos sensores criando um respectivo nó para cada um, conforme citado acima.

Dentro do nó de cada um dos dois sensores ocorre um laço de repetição (provisto pelo ROS, e chamado de "*timer*" que lê os dados do sensor através do barramento I2C a uma frequência de 100Hz. Os dados são validados no processo da leitura, e são enviados por um outro "*timer*" que publica os dados para que outros nós possam receber a uma frequência de 10Hz.

A mensagem de cada sensor possui todos os dados lidos da I2C e convertidos nas unidades de medidas adotadas. Uma variável indica a corretude dos dados. Caso a validação dos dados falhe, essa variável indica que os dados são inválidos e, portanto, não devem ser utilizados pelos outros nós.

2.5.5 Rotina de coleta de dados

Na aplicação, um trajeto deve ser definido para o drone utilizando o QGroundControl[11]. Esse trajeto deve ser elaborado e carregado na Pixhawk/PX4, e deve possuir pontos de parada em locais estratégicos para coleta de dados (“waypoints”).

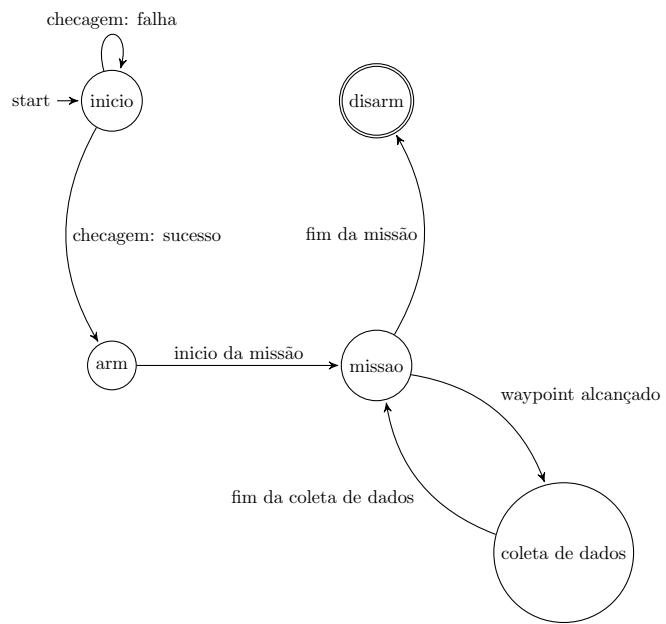
O sistema tem o objetivo coletar dados nos “waypoints” determinados. Quando um “waypoint” é atingido, uma mensagem de “waypoint reached” é enviada via Mavlink da Pixhawk para o QGroundControl, e ela pode ser acessada por meio de um tópico pré-definido contido no MAVROS chamado de “waypoint topic”. Dessa forma, o sistema aguarda um “waypoint topic” constantemente a uma taxa de 100Hz e, quando ele indica que o drone chegou em um “waypoint”, o sistema entra em ação e assume o controlar do drone.

Uma máquina de estado referente ao controle do drone foi projetada e implementada. O primeiro estado é o responsável por verificar a chegada no “waypoint”. Quando o drone atinge um “waypoint” o segundo estado é acionado. No segundo estado, primeiramente é realizada a troca do modo de controle do drone para “OFFBOARD”, permitindo que o drone seja controlado com mensagens enviadas pelo sistema desenvolvido. Após a alteração do modo é realizada a verificação do sensor lidar, abordada no próximo tópico, para então o drone parte para a coleta de dados.

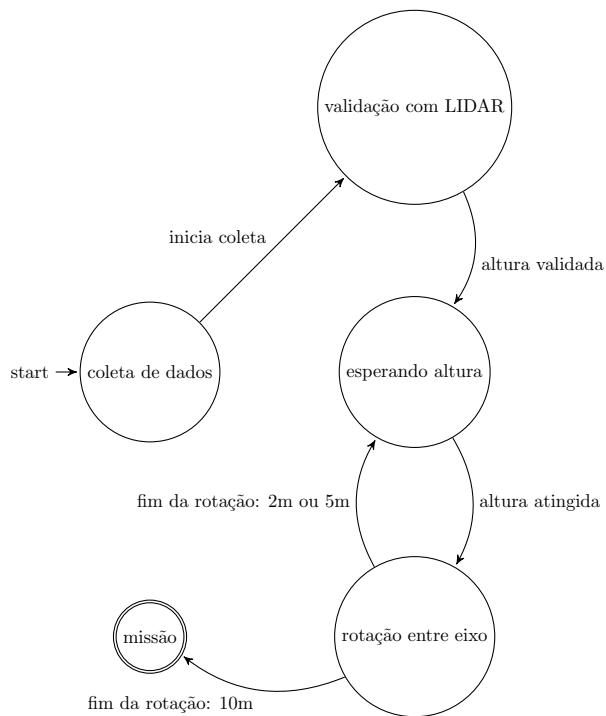
Ainda no segundo estado, o drone, estando ciente de sua altitude através dos dados recebidos do sensor LIDAR, parte para a altitude da primeira coleta. O sistema está programado para realizar a coleta dos dados nas alturas de 2, 5 e 10 metros acima do terreno. Quando o drone alcança a altura de 2 metros (com uma margem de erro pré-definida de 30 cm), o drone gira em torno do próprio eixo coletando e armazenando os dados dos sensores, a rotação completa dura 5 segundos e, após finalizada, o drone parte para a próxima altitude de coleta.

Após a coleta de dados na última altura definida (10 metros acima do nível do terreno), o controle do drone é devolvido ao piloto automático através da mudança do modo do drone para “AUTO.MISSION” (terceiro estado da máquina de estados), retornando a seguir o seu trajeto original, subindo para a altura determinada no QGroundControl e partindo para o próximo “waypoint” e máquina de estados volta para o primeiro estado. Quando todos os “waypoints” são atingidos, o drone parte para o pouso no ponto pré-determinado, que geralmente se encontra no mesmo local da decolagem.

Segue, abaixo, uma máquina de estados que representa superficialmente o funcionamento do sistema.



Segue, abaixo, uma máquina de estados que representa o funcionamento do sistema dentro da etapa de coleta de dados.



2.5.6 Sensor LIDAR

O piloto automático Pixhawk/PX4 é equipado com um sensor barométrico que permite determinar a altitude do drone em relação ao nível do mar. A altura aproximada do drone em relação ao solo é obtida pela diferença entre da altitude do ponto inicial armazenada durante a inicialização do sistema e a altitude do ponto atual. Caso existam irregularidades no terreno, ou até mesmo obstáculos (árvores, por exemplo), a altura do drone em relação ao solo calculada através dos dados do barômetro pode ser inválida e não pode ser utilizada para o posicionamento em relação ao solo em uma posição diferente da posição de decolagem. Para solucionar esse problema, foi acoplado ao drone um sensor LIDAR (*Light Detection And Ranging*), que calcula a distância do drone em relação ao solo através de um laser pulsado constantemente.

Apesar das especificações do fabricante indicarem a capacidade de medir distâncias de até 40 metros, o sensor LIDAR utilizado apresenta na prática valores válidos dentro do intervalo de altura em relação ao solo de 2 a 10 metros em qualquer tipo de terreno. Para solucionar esse problema foi adicionada à rotina de coleta de dados uma etapa de verificação da altura em relação ao solo utilizando os dados do sensor LIDAR. Quando o drone atinge um "waypoint", ele começa a descer até obter um valor válido para o LIDAR, e então atualiza a sua altura em relação ao solo com base nesse valor. Dessa forma, eventuais problemas em relação a altura do drone em relação ao ponto no terreno são evitados.

Vale ressaltar que essa verificação da altura só ocorre caso o sensor LIDAR tenha funcionado em algum momento do voo. Caso ele não tenha apresentado valores válidos em nenhum momento, a etapa deixa de ser realizada e o drone é guiado somente pelo barômetro.

2.5.7 Dados coletados

Durante a rotação do drone, na rotina de coleta de dados, os dados são armazenados em arquivo no pendrive conectado à Raspberry. Os seguintes dados são armazenados: a temperatura no local, a velocidade do ar (com e sem cálculos de compensação realizados com a densidade do ar e com e sem o offset apresentado pelo sensor na inicialização do sistema), a umidade do ar, a direção da bússola no respectivo instante, a latitude, longitude e altitude do drone, a altura do drone em relação ao solo utilizando o barômetro e o sensor LIDAR, e um "timestamp" para determinar o exato momento da coleta dos dados.

Todos os valores acima são gravados a uma frequência programada de 72Hz, que pode variar a depender da velocidade da placa para gravar os dados. Essa frequência é maior que a de leitura dos sensores de temperatura e velocidade do ar, entretanto esse é um arquivo com fins de depuração do sistema.

Durante a coleta dos dados, o maior valor de velocidade do vento é sempre armazenado junto com os outros valores acima apresentados. Conhecendo o maior valor da velocidade do vento durante a rotação, é possível determinada a direção do vento. Dessa forma, é armazenado no pendrive um arquivo com os dados filtrados, sendo esses somente os dados apresentados no momento em que o sensor responsável pela velocidade do ar captou o maior valor, implicando em ser exatamente o momento em que ele aponta na direção do vento.

Uma observação a ser levada em conta é que o valor gravado da bússola é o

módulo da soma do valor real com 180, para indicar a real direção do vento.

O arquivo com os dados de depuração é nomeado de "*data_X*", e o arquivo com os dados filtrados de "*regina_X*", onde "X" é o "*timestamp*" do momento em que o programa é iniciado. Vale ressaltar que esse "*timestamp*" se refere ao tempo marcado na placa e que não é equivalente a data corrente em questão, pois da placa não possui uma bateria CMOS para armazenar esse tipo de dado.

2.5.8 Transmissão de dados em tempo real e aplicativo para celular

A última etapa do projeto a ser implementado consiste na transmissão dos dados coletados em tempo real para a base. Como citado anteriormente, foi idealizada inicialmente a utilização de GPRS para transmissão dos dados, porém não pode ser realizada por limitações físicas da placa escolhida e pela rede GPRS cobrir totalmente as regiões agrícolas. Dessa forma, a alternativa encontrada, após diversas tentativas que serão abordadas abaixo, foi a utilização de uma rede Wi-Fi para transmitir os dados, uma vez que o operador estará próximo do drone durante a realização da missão.

A ideia inicial para a transmissão de dados em tempo real era a de utilizar a Pixhawk/PX4 para enviar uma mensagem via MAVlink para o QGroundControl com os dados coletados, através do link de telemetria à estação de solo. Entretanto, alguns problemas tornaram a solução inviável. Para transferir os dados coletados dos sensores via Pixhawk/PX4, torna-se necessária a criação e definição de uma nova mensagem no MAVlink (maiores detalhes podem ser encontrados em <https://dev.px4.io/v1.9.0/en/middleware/mavlink.html>). Entretanto, a adição de novas mensagens exige a recompilação do sistema PX4, o que limita a escalabilidade do sistema, visto que para cada nova versão do piloto automático torna-se necessário refazer a compilação do MAVlink. A recompilação pode ser uma tarefa exaustiva tendo em vista a quantidade de erros que podem surgir durante o processo e que devem ser corrigidos durante a mesma. Para a interpretação da nova mensagem pelo software QGroundControl através da ferramenta MAVlink Inspector, também é necessário recompila-lo.

É válido ressaltar que tanto utilizando o suporte do MAVROS para envio de mensagens MAVlink personalizadas (utilizando o tópico /mavros/mavlink/to) quanto fazendo a implementação do zero, os problemas eram os mesmos e, portanto, acabariam por continuar impossibilitando a utilização do sistema em larga escala.

Tendo em vista as complicações anteriores, a utilização de uma rede Wi-Fi foi a alternativa escolhida para a transferência de dados em tempo real. Foi feita uma configuração no gerenciador de rede da placa para que ela sempre buscasse se conectar automaticamente a qualquer rede com o nome de "*raspberry*", com senha "*raspberry*". Dessa forma, pode-se usar tanto um roteador, quanto a placa de rede de um computador ou a função de "hotspot móvel" do celular para criar uma rede que será utilizada para a transmissão dos dados.

Uma das limitações da rede Wi-Fi, entretanto, é em relação a área de alcance da rede e, consequentemente, a distância em que o drone estará disponibilizando os dados em tempo real. Estudos indicam que uma rede Wi-Fi de 2.4 GHz pode alcançar até 92 metros em ambientes externos, a depender do dispositivo utilizado. Esse problema pode ser facilmente contornado substituindo o Wi-Fi por um modem de longo alcance.

Os dados filtrados dos sensores são então publicados via HTTP (Hypertext

Transfer Protocol) no endereço local da placa na rede. Os dados são publicados constantemente enquanto a placa estiver conectada na rede Wi-Fi, e voltão a ser publicados após eventuais quedas ou saídas do alcance da rede.

A publicação dos dados ocorre via JSON (JavaScript Object Notation) e deve ser feita no endereço IP (Internet Protocol address) local da placa na rede. Para não ser necessário realizar um *scan* na rede em busca do IP da placa, foi utilizado um protocolo DNS chamado de mDNS (Multicast DNS), que basicamente atribui a um determinado "hostname" o IP da placa. Dessa forma, dentro de uma rede Wi-Fi, o endereço "*pi.local*" está configurado para apontar para o endereço local da placa, independente da rede em que ela estiver, e pode ser utilizado para acessar os dados publicados por ela em formato JSON.

Para apresentar os dados ao usuário, foi desenvolvido um aplicativo mobile para Android em Flutter, um conjunto de ferramentas desenvolvido pela Google para desenvolver, entre outras coisas, aplicações mobile.

O aplicativo é simples e intuitivo, basta abri-lo e conectar o celular na rede Wi-Fi com nome "*raspberry*" a qual a placa estará conectada e os dados serão mostrados em tempo real. A atualização dos valores ocorre de 5 em 5 segundos e pode ser feita arrastando o dedo para baixo na tela do aplicativo, ou apertando o botão indicado no mesmo. Vale ressaltar que, caso o drone saia do alcance da rede Wi-Fi, os dados serão atualizados a partir do momento em que o drone voltar para o alcance da mesma.

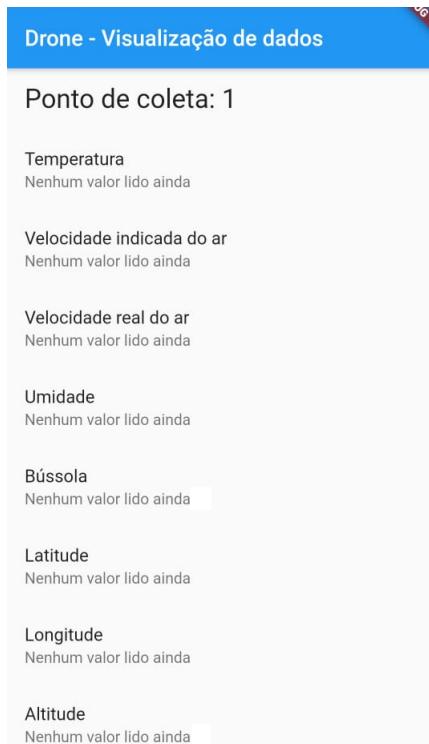


Figura 16: Tela principal do aplicativo

2.5.9 Codificação

Após a leitura do guia de início, dos tutoriais da página oficial do ROS[17] e a ambientação com o MAVROS, foi iniciada a codificação do sistema. O MAVROS possui uma grande quantidade de tópicos para trabalhar com drones, o que simplifica a utilização do ROS.

Conforme os requisitos da aplicação foram sendo cumpridos, pesquisas foram realizadas para encontrar tópicos do MAVROS que pudessem auxiliar no cumprimento dos novos requisitos. Vale ressaltar que existe uma comunidade consideravelmente grande em fóruns da área, fóruns esses que foram consultados em diversos momentos de dúvidas e trouxeram opiniões relevantes para a construção do sistema.

Não é cabível dentro desse relatório explicar linha a linha do código construído para a aplicação. Entretanto, o código pode ser consultado no GitHub. O GitHub foi a ferramente de versionamento de código utilizada para controlar o desenvolvimento do código, e cada alteração no código foi dividida em "commits" e pode ser visualizada no repositório em questão [16].

O repositório do GitHub possui todos os códigos de teste e "debug" que foram utilizados para o desenvolvimento do projeto, entretanto os códigos principais e as versões finais estão munidas de comentários com o intuito de facilitar o entendimento do código.

Dentro da pasta "Testes", no repositório, pode-se encontrar os códigos referentes a leitura de dados dos sensores. Existe um Makefile com o intuito de facilitar a compilação dos mesmos. A pasta "collect_data" é a pasta referente ao pacote do projeto criado. Dentro dessa pasta, pode-se encontrar os códigos desenvolvidos ao acessar a pasta "src". Os arquivos que foram utilizados na versão final do projeto foram: "final_version.cpp" (arquivo que contém o código do nó principal da aplicação), "test_system.cpp" arquivo que contém o código do nó responsável por fazer os testes de integridade do sistema), "ms4525.cpp" e "hdc1050.cpp" (arquivos que contém, respectivamente, os códigos dos sensores associados aos nomes dos arquivos). Dentro da pasta "msg", em "collect_data", pode-se ver os arquivos que foram utilizados para gerar as mensagens ROS personalizadas para modularizar a utilização dos sensores do sistema.

Na raiz do repositório, pode-se encontrar o *script* responsável por controlar o botão de "shutdown" do sistema (arquivo "shutdown.py3"), o ".bashrc", que é o arquivo de configuração do terminal do sistema e que possui alguns atalhos que foram criados para facilitar o desenvolvimento do sistema, e o arquivo "startup_launch.sh", que é chamado pelo serviço criado responsável pela inicialização do sistema, que está configurado no arquivo "mavros.service".

2.5.10 Organização do sistema dentro do Linux

Essa seção descreve a distribuição dos arquivos de configuração do sistema desenvolvido dentro do sistema de arquivos do sistema operacional utilizado.

O ambiente de desenvolvimento do ROS, chamado de *catkin_ws*, pode ser acessado no diretório /home/pi/catkin_ws, e todos os códigos do projeto estão dentro da pasta src/collect_data. Para realizar alterações no sistema de voo, pode-se alterar os códigos dentro de src/collect_data/src e recompilar os mesmos acessando a raspberry via ssh (vide seção de tutoriais) e utilizando do comando build (atalho para catkin build collect_data -j1). Pode ser necessário realizar o

setup das variáveis de ambiente, caso apareça algum erro na compilação, através do comando `source /home/pi/catkin_ws/devel/setup.bash`.

Caso problemas relacionados a lentidão na compilação dos arquivos apareçam, pode-se desligar o sistema de voo através do comando `sudo systemctl stop mavros`, o que fará com que a placa tenha mais poder de processamento disponível. É importante ressaltar que não se deve re-recompilar os projetos mavros e mavlink, caso seja necessário devido alguma atualização para suporte a novos recursos, deve-se fazer emulando o sistema da placa em um computador através do emulador QEMU no Linux seguindo algum dos diversos tutoriais existentes em fóruns disponíveis da internet.

O *script* de inicialização do sistema, chamado pelo serviço criado (*mavros.service*), pode ser encontrado e alterado em `/home/pi/scripts/startup_launch.sh`, enquanto os programas para o teste dos sensores de velocidade do ar e temperatura podem ser encontrados em `/home/pi/Testes`. O repositório do projeto do GitHub está clonado em `/home/pi/tt2-fapesp-ros` e possui a versão do momento atual do projeto.

Outras configurações quaisquer compatíveis com a versão do sistema operacional utilizado podem ser feitas através da linha de comando, e não é recomendável que a interface gráfica do sistema seja ativada, por conta da demanda exigida do processador e da memória RAM pelos sistemas rodando em segundo plano.

Vale ressaltar, finalmente, que caso qualquer alteração seja feita no sistema será necessário fazer a reinicialização do mesmo para que ela seja aplicada. Pode-se fazer essa tarefa através do comando `sudo reboot`. Caso alguma senha seja solicitada em algum momento por conta da utilização do sudo no Linux, deve-se utilizar "pi".

2.5.11 Distribuição do sistema

O sistema foi construído com o intuito de ser escalável, algo que pudesse ser utilizado por qualquer pessoa com um drone e disponibilidade de realizar o setup de instalação e montagem do ambiente.

Decidiu-se, portanto, gerar uma imagem do sistema de forma a ser obtida e gravada em um cartão SD por qualquer pessoa. A seção tutoriais possui mais informações sobre a gravação da imagem do sistema em um cartão SD.

Para o desenvolvimento do sistema foi utilizado o esquema da Figura 17 que relaciona os sensores, a Pixhawk/PX4 e Raspberry.

2.5.12 Utilização do sistema

Para utilizar o sistema, após ter realizado os passos relativos a distribuição e instalação do setup do mesmo, pode-se seguir os seguintes etapas para um bom funcionamento do sistema:

1. Criar uma missão no QGroundControl e realizar o upload na Pixhawk/PX4;
2. Posicionar o drone em um local plano e apropriado para a decolagem;
3. Proteger o tubo de pitot conectado ao sensor MS4525 para que o sistema registre o valor de zero do sensor de airspeed;
4. Aguardar o LED verde que indica a inicialização correta do sistema;

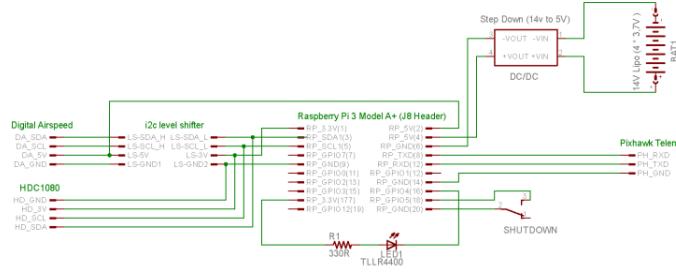


Figura 17: Diagrama elétrico do sistema

5. Aguardar cerca de 15 segundos;
6. Iniciar o voo através do QGroundStation;

Após o voo ser iniciado, pode-se acompanhar os valores coletados em tempo real através do aplicativo, ou aguardar o drone pousar e conferi-los nos arquivos armazenados no pendrive acoplado à placa raspberry. É válido ressaltar novamente a necessidade de realizar o *"shutdown"* do sistema de forma correta, utilizando o botão de *"shutdown"* desenvolvido propriamente para isso, antes de retirar o pendrive da placa ou desligar a energia do drone.

2.6 Tutoriais

Os tutoriais de utilização do sistema produzidos podem ser acessado através do link <https://github.com/alcidesmig/tt2-fapesp-ros/blob/master/tutoriais.pdf>.

3 Conclusão

Por meio da utilização de frameworks, bibliotecas e pacotes de código disponíveis na internet pôde-se desenvolver um sistema que ao ser integrado em um drone pode trabalhar conjuntamente com o mesmo de forma a coletar, gravar e disponibilizar dados de locais escolhidos pelo piloto e realizar o monitoramento de condições atmosféricas do local.

Assim, é possível utilizar os dados coletados para realizar estudos quanto a viabilidade de determinação da inversão térmica, possibilitando otimizar os processos de pulverização. O monitoramento do fenômeno da inversão térmica com drones permite atingir maior mobilidade e precisão nas determinações substituindo equipamentos terrestres e podendo alcançar modelos de decisão em tempo real e com menor número de aplicações possível.

Provou-se, então, que um sistema para realizar o monitoramento do fenômeno em questão utilizando drones é factível, podendo ser realizado de maneira funcional e a apresentar resultados válidos. Vale ressaltar que o fenômeno da inversão térmica é algo que não afeta somente a agricultura, o que permite que o projeto seja utilizado para outros fins por quaisquer outras pessoas interessadas.

A etapa final do projeto previa o desenvolvimento de um mecanismo para transferência em tempo real dos dados coletados. Diversas foram as soluções estudadas. A primeira tentativa foi a de transferir os dados utilizando a Pixhawk/PX4 para o software de controle do drone via link de telemetria. Também foi realizada uma tentativa de transferência por meios de um programa externo. Devido a complicações em relação a escalabilidade dos resultados obtidos, foi preferível utilizar a transferência de dados para um aplicativo Android/iOS desenvolvido em Flutter[2] conectado na placa via rede Wi-Fi.

Foi disponibilizada uma imagem de sistema que pode ser gravada por qualquer pessoa e utilizada para fins de estudo, continuação ou utilização dos resultados do projeto. Todos os códigos foram devidamente comentados e tutoriais foram escritos com o intuito de tornar o sistema o mais transparente o possível.

Têm-se, portanto, finalizadas todas as atividades propostas no plano de atividades do projeto.

4 Referências Bibliográficas

- [1] Bicudo do algodoeiro. https://www.agrolink.com.br/problemas/bicudo_29.html.
- [2] Flutter. <https://flutter.dev/>.
- [3] Gazebo Simulation. <http://dev.px4.io/v1.9.0/en/simulation/gazebo.html>.
- [4] HDC1050 Datasheet. <http://www.ti.com/lit/ds/symlink/hdc1050.pdf>.
- [5] HITL Simulation. <https://dev.px4.io/v1.9.0/en/simulation/hitl.html>.
- [6] MAVLink developer guide. <https://mavlink.io/en/>.
- [7] MAVROS Wiki. <http://wiki.ros.org/mavros>.
- [8] MS4525 Datasheet. https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FMS4525%7FB9%7Fpdf%7FEnglish%7FENG_DS_MS4525_B9.pdf%7F4425-005D.
- [9] Pixhawk Controller. https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html.
- [10] QEMU Website. <https://www.qemu.org/>.
- [11] QGroundControl User Guide. <https://docs.qgroundcontrol.com/en/>.
- [12] Raspberry - raspi-config. <https://www.raspberrypi.org/documentation/configuration/raspi-config.md>.
- [13] Raspberry Pi - I2C. <https://www.instructables.com/id/Raspberry-PI-Multiple-I2c-Devices/>.
- [14] Raspberry Pi 3 A+. <https://pi4j.com/1.2/pins/model-3a-plus-rev1.html>.
- [15] Raspbian Website. <https://www.raspbian.org/>.
- [16] Repositório do projeto. <https://github.com/alcidesmig/tt2-fapesp-ros/>.
- [17] ROS - Start Guide. <http://wiki.ros.org/ROS/StartGuide>.
- [18] ROS Website. <http://wiki.ros.org/Documentation>.
- [19] ROS Wiki. <http://wiki.ros.org/>.
- [20] Secure Shell. https://pt.wikipedia.org/wiki/Secure_Shell.
- [21] Ubuntu Mate Website. <https://ubuntu-mate.org/>.