MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
(Real Academia de Artilharia Fortificação e Desenho - 1792)

CONTROLE DE DRONE COM GESTOS

Leonardo Gomes Gonçalves
Felipe **Reyel** Feitosa de Sousa

Prof. Maj Menezes

RIO DE JANEIRO
31 DE MAIO DE 2021

Sumário

1. INTRODUÇÃO

É inegável atualmente a versatilidade no que diz respeito à aplicabilidade relacionada a drones. Algumas das áreas de aplicação são: fotografias aéreas, delivery, mapeamento geográfico, agricultura de precisão, procura e resgate em acidentes, previsão do tempo, monitoramento de vida selvagem, policiamento remoto e como forma de entretenimento [1].

Prevê-se um grande crescimento no cenário da indústria de drones, sendo cada vez mais acessíveis ao público comum. Ainda é de se esperar que no Brasil, o desenvolvimento científico e tecnológico avance de modo expandir as aplicações do uso de drones da maneira que já é feita em outras grandes potências como Estados Unidos, Rússia e China [2].

Com essa rápida expansão do mercado, entra-se em foco o modo de controle do drone. Dependendo do objetivo que se pretende alcançar, esse controle pode ser feito de maneira autônoma ou manual, que será o foco deste projeto.

Outra área tecnológica que vem tomando importância crescente é a de Realidade Virtual, onde um dos fatores mais importantes é o grau de imersão do usuário em um ambiente virtual. Com o uso de reconhecimento de pose para as mãos do usuário, mistura-se os estímulos sensoriais de visão e de tato. Com as tecnologias disponíveis no estado da arte para hand-tracking, o ferramental necessário já encontra-se disponível para incluir o hand-tracking em ambientes de VR [3], aumentando consideravelmente o grau de imersão do usuário.

O presente projeto tem por finalidade mesclar os temas acima e fornecer um sistema embarcado que permita o controle total de um drone a partir de gestos, com status de bateria e informações sobre o drone localizados em uma interface e botões para alterar o modo de voo e auxiliar em pousos e decolagens.

2. CONCEITOS BÁSICOS

2.1. Movimentos de um drone

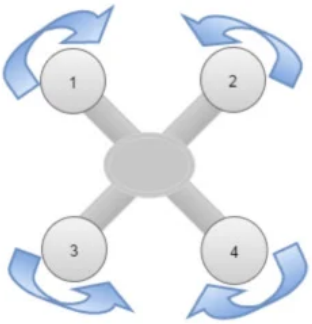

Figura 2.1.1: Diagrama dos motores de um drone

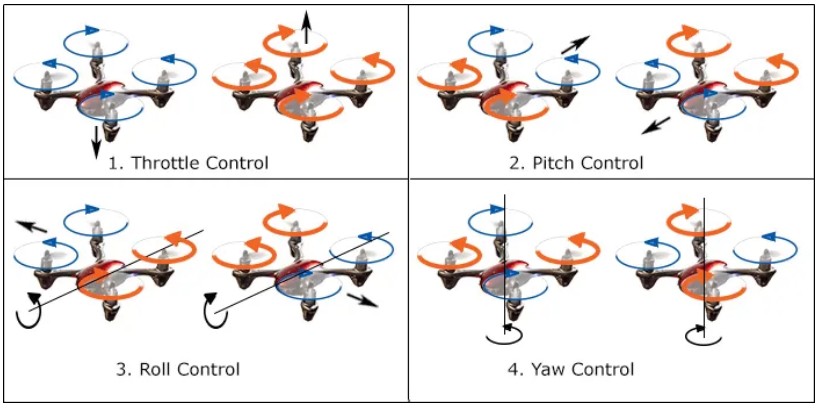

Figura 2.1.2: Movimentos de um drone

Existem quatro movimentos principais que um drone emprega e eles são controlados por cada uma das quatro hélices. As hélices 1 e 4 movem-se no sentido horário, enquanto as hélices 2 e 3 movem-se no sentido anti-horário, conforme figura 2.1.1.

Yaw é o giro de um drone no sentido horário ou anti-horário. Para que o drone use o *yaw* para girar para a esquerda, as hélices 1 e 4 se movem em velocidade normal, enquanto as hélices 2 e 3 se movem em alta velocidade. Para girar para a direita, as hélices 1 e 4 se movem em alta velocidade e as hélices 2 e 3 se movem em velocidade normal.

Pitch descreve o movimento para frente e para trás de um drone. Para ir para a frente, as hélices 1 e 2 movem-se em velocidade normal, enquanto a hélice 3 e 4 se movem em alta velocidade. Para mover para trás, as hélices 1 e 2 se movem em alta velocidade, enquanto as hélices 3 e 4 se movem em velocidade normal.

Roll é a rotação do drone para virar para a esquerda ou para a direita. Para virar para a esquerda, as hélices 1 e 3 se movem em velocidade normal, enquanto as hélices 2 e 4 se

movem em alta velocidade. Para rolar para a direita, as hélices 1 e 3 se movem em alta velocidade e as hélices 2 e 4 se movem em velocidade normal.

Subida / descida são os atos de mover o drone para cima e para baixo, aumentando ou diminuindo a elevação. Para subir, todas as hélices se movem em alta velocidade e, para descer, todas as hélices se movem em velocidade normal.

A maioria dos drones restringe os movimentos da aeronave nesses 4 movimentos, abstraindo o controle individual das velocidades dos motores ao operador. O drone DJI Tello cria uma camada a mais sobre esses movimentos e fornece uma API via websocket para controlá-los:

Movimento	Comando	Movimento	Comando
Subir x centímetros	up x	Direita x centímetros	right x
Descer x centímetros	down x	Rodar x graus no sentido horário	cw x
Trás x centímetros	back x	Rodar x graus no sentido anti horário	ccw x
Frente x centímetros	forward x	Decolar	takeoff
Esquerda x centímetros	left x	Pousar	land

2.2. Acelerômetro

Um acelerômetro é um dispositivo compacto desenhado para medir a aceleração. Quando é integrado em um objeto que tem mudanças no vetor da velocidade ele responde às vibrações associadas a tal movimento. Geralmente contêm cristais microscópicos que respondem a tais vibrações gerando uma pequena voltagem que é então lida pelo dispositivo e a aceleração pode ser calculada nos três eixos desse sensor [4].

Hoje podem ser encontrados em circuitos integrados de milímetros de tamanho e excelente precisão, conforme figura abaixo.

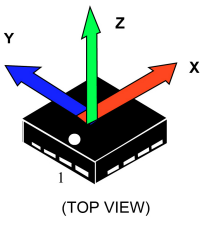

Figura 2.2.1: Esquema de acelerômetro

3. REVISÃO DE LITERATURA

O presente projeto se inspira em dois trabalhos japoneses [5] e [6] com objetivo similar: controlar o drone DJI Tello com os sensores de uma placa M5. Tais projetos têm placas e mecânica de controle levemente diferentes em relação ao comportamento dos botões pretendidos no escopo desse projeto.

No modo de voo livre o piloto controla a posição horizontal do veículo com o acelerômetro da placa selecionada. Mas para controlar o *yaw*, pouso e decolagem é necessário permanecer pressionando um botão enquanto faz movimentos com a mão para serem detectados pelo acelerômetro da placa.

Essa abordagem apresenta uma dificuldade devido a rigidez e posicionamento do botão, tornando ela desconfortável para o operador, possivelmente diminuindo a utilidade e aplicabilidade do projeto.

Na solução proposta no capítulo seguinte usaremos o conceito de estados/modos de voo para facilitar a operação do dispositivo e aumentar conforto e utilidade do mesmo.

4. EXPERIMENTOS REALIZADOS E RESULTADOS

4.1. Material

Para o desenvolvimento da solução, foi utilizado o Drone DJI Tello (Figura 4.1.1a) e um M5StickC (Figura 4.1.1b) como controlador do sistema embarcado.

(a) Drone DJI Tello (b) M5StickC

Figura 4.1.1: Componentes utilizados para o projeto

4.2. Metodologia de desenvolvimento

Seguindo desenvolvimento por módulos, o projeto foi desenvolvido em duas etapas: codificação do algoritmo de reconhecimento de gestos na placa controladora (simulando o drone com um servidor websocket servido no computador) e codificação do sistema de feedback de comandos (dessa vez usando o drone para testes). Foi feito assim para isolar variáveis e facilitar o desenvolvimento e depuração.

Na etapa de codificação do algoritmo de reconhecimento usou-se a heurística de zona morta: até um valor máximo do seno do ângulo da aceleração nenhum comando era interpretado. Depois desse valor interpretava-se a angulação como booleana, ou seja, não fez-se distinção entre ângulos grandes e ângulos pequenos, interpretando só se está inclinado ou não. Com os gestos interpretados traduziu-os para comandos do SDK do drone e estes foram enviados via websocket para o servidor de testes.

Na etapa do feedback de comandos apenas foi adicionado ao código previamente gerado a lógica de captura de respostas do drone para os comandos enviados. Essas respostas foram capturadas e codificadas para serem exibidas na tela do controlador.

4.3. Testes planejados

Para fazer uma validação se o drone estava com fácil controle, um teste foi elaborado um circuito consistindo de:

- ➔ O drone deve sair de uma posição estacionária no chão e decolar;
- ➔ Deverá seguir reto por 2 metros e virar à esquerda;
- ➔ Em seguida deverá descer 50 centímetros e passar por um alvo, virando a direita após;
- ➔ Depois o drone deverá subir 1 metro e andar 2 metros para a frente;
- ➔ Por fim, o drone deve ficar estacionário no ar, permitindo que o operador se desloque até embaixo dele e em seguida pouse na sua mão.

4.4. Resultados obtidos

Figura 4.4.1: Thumbnail do vídeo de demonstração - link do vídeo

O vídeo da figura 4.4.1 mostra o resultado da demonstração. Pode-se perceber que, embora tenham tido dificuldades como a velocidade e o controle fino da posição da aeronave, o drone conseguiu cumprir com facilidade as etapas do percurso.

É notável também que os feedbacks dos status do drone funcionaram muito bem, mostrando ao operador dados cruciais para a operação, como bateria e tempo de voo.

5. CONCLUSÃO

De acordo com os resultados obtidos dos testes realizados com o drone no percurso planejado, podemos concluir que o controle fornecido pelo M5Stickc é bom, permitindo boa capacidade de manobra para o operador e ainda deixá-lo estacionário no ar para livrar as mãos do operador.

Como possíveis trabalhos futuros é possível utilizar uma controladora com uma câmera ao invés de acelerômetro para identificar gestos, assim diminuindo mais ainda o acoplamento entre operador e veículo.

Um passo mais avançado, e possivelmente a melhor opção, seria utilizar a própria câmera do drone para reconhecer os gestos e executá-los, completamente eliminando a necessidade de controlador externo, tornando a relação operador-drone mais simples e dando mais autonomia para a aeronave.

6. REFERÊNCIAS

[1] Stunning applications of drone technology, Naveen Joshi. 2018. Disponível em <https://www.allerin.com/blog/10-stunning-applications-of-drone-technology>. Acesso em 29 de abril de 2021.

[2] CONTROLE DE DRONES ATRAVÉS DO RECONHECIMENTO DE IMAGENS DO TERRENO, Cocenza e Germano, 2020 Disponível em <http://www.comp.ime.eb.br/graduacao/pfc/repositorio-pfc/2020/PFC-CarlaGermano.pdf>. Acesso em 29 de abril de 2021.

[3] INTERAÇÃO POR HAND-TRACKING EM AMBIENTE DE REALIDADE VIRTUAL, Castro e Xavier, 2020 Disponivel em <http://www.comp.ime.eb.br/graduacao/pfc/repositorio-pfc/2020/PFC_DeCastroXavier.pdf>. Acesso em 29 de abril de 2021.

[4] Accelerometer vs. Gyroscope: What's the Difference, Ryan Goodrich, 2017. Disponível em <https://www.livescience.com/40103-accelerometer-vs-gyroscope.html>. Acesso em 29 de abril de 2021.

[5] TelloをM5Stack Grayでコントロールするスケッチを作りました, Mitsu Murakita, 2019. Disponível em <https://qiita.com/Mitsu-Murakita/items/b86ad79d3590adb3b5b9>. Acesso em 29 de abril de 2021.

[6] TelloをM5StickCでコントロールするスケッチを作りました, Mitsu Murakita, 2019. Disponível em <https://qiita.com/Mitsu-Murakita/items/c078752a570bf1295782>. Acesso em 29 de abril de 2021.

[7] Documentação do M5StickC, M5Stack, 2019. Disponível em <https://docs.m5stack.com/#/en/core/m5stickc>. Acesso em 29 de abril de 2021.

[8] SDK DJI Tello, Ryze Robotics, 2018. Disponível em <https://www.ryzerobotics.com/tello/downloads>. Acesso em 29 de abril de 2021.

[9] Projeto próprio no GitLab: Controlling a DJI Tello Drone with hand movement detected by an M5StickC board, Felipe Reyel, 2021. Disponível em <https://gitlab.com/ime-projects/embedded-systems-drone-control>. Acesso em 29 de abril de 2021.

7. ANEXO - SISTEMA DESENVOLVIDO

Nessa seção será comentado todo o código do controlador. A versão contínua do código pode ser encontrada em [9].

7.1. Inclusões, constantes e variáveis globais.

```
#include <M5StickC.h>
#include <WiFi.h>
#include <WiFiUdp.h>

#define max_sine 0.4
#define pool_time 200
#define takeoff_time 3000
#define land_time 10000

const char *ssid = "TELLO-C470F1";
const char *password = "";

const char* TELLO_IP = "192.168.10.1";
const int PORT = 8889;

WiFiUDP Udp;
char packetBuffer[255];
String message = "";

float accX = 0.0F;
float accY = 0.0F;
float accZ = 0.0F;

bool isOnAir = false;
bool conectado = false;

int movementState = 0;
int readState = 0;
unsigned long msg_time;
```

O bloco de código acima inclui as bibliotecas usadas, a API do M5StickC, da WiFi e do websocket UDP. Também define constantes usadas ao longo do projeto como o **seno máximo** que o sistema usa como região que nao ativa a aeronave, o **tempo de polling** entre as leituras do sensor e atualizações do controlador, tempo de **decolagem** e **pouso**, nome da **WiFi**, **IP** da aeronave e **porta** onde ela escuta o websocket. Inicia também variáveis usadas ao longo do projeto: instância **Udp** utilizada para comunicação websocket, **packetBuffer** e **message** usadas para receber mensagens do drone, **acc** usadas para ler a aceleração do drone nos eixos. As demais variáveis definem estados: **isOnAir** define se a aeronave está no ar ou no chão, **conectado** define se esta conectado na WiFi, **movementState** define o estado de voo, **readState** define qual propriedade o controlador está lendo do drone e **msg_time** guarda o tempo em milissegundos da última mensagem enviada para o drone.

7.2. Funções que mostram estado e comandos

```
void printMsg(int x, int y, int w, int h, uint16_t color, String msg) {
  M5.Lcd.fillRect(x, y, w, h, color);
  M5.Lcd.setCursor(x, y+3);
  M5.Lcd.println(msg);
}

void print_status(String status_msg, uint16_t color){
  printMsg(0, 14, 80, 15, BLACK, status_msg);
}

void print_movement_state(){
  if (movementState == 0) {
    print_status("Horizontal", BLUE);
  } else if (movementState == 1) {
    print_status("Vertical", CYAN);
  } else {
    print_status("Steady", NAVY);
  }
}

void print_air_state(){
  printMsg(0, 0, 80, 15, isOnAir ? RED : GREEN, isOnAir ? "On Air" : "On Land");
}

void print_command(String status_msg){
  printMsg(0, 79, 80, 40, BLACK, status_msg);
}

void print_message(String cmd){
  printMsg(0, 139, 80, 40, PINK, cmd + "\n> " + message);
}
```

Essas funções são helpers para impressão na tela do controlador os estados do drone: **printMsg** e o helper genérico que limpa um lugar da tela e imprime nele uma mensagem, **print_air_state** imprime no topo da tela o estado **isOnAir**, **print_status** imprime abaixo do topo uma mensagem de status (e um helper utilizado pela função **print_movement_state** e pela função que conecta a WiFi, mais adiante), **print_movement_state** imprime o estado de voo, **print_command** imprime no meio da tela o comando sendo enviado para o drone e **print_message** imprime a resposta de tal comando.

7.3. Funções de enviar comandos e receber comandos

```
void tello_command_exec(char* tello_command){
  print_command(tello_command);

  Udp.beginPacket(TELLO_IP, PORT);
  Udp.printf(tello_command);
  Udp.endPacket();
  listenMessage(tello_command);
  delay(100);
  msg_time = millis();
```

```
}

void listenMessage(String cmd) {
  int packetSize = Udp.parsePacket();
  if (packetSize) {
    IPAddress remoteIp = Udp.remoteIP();
    int len = Udp.read(packetBuffer, 255);
    if (len > 0) {
      packetBuffer[len] = 0;
    }
  }
  message = (char*) packetBuffer;
  print_message(cmd);
}
```

A função **listenMessage** escuta por um pacote UDP e se achar guarda o conteúdo da mensagem na variável message e chama a função **print_message** que imprime-a. Já a função **tello_command_exec** envia um comando para a aeronave, imprimindo o comando e esperando uma resposta, guardando também o tempo no qual enviou a mensagem.

7.4. Conectar a WiFi e iniciar o M5

```
void ConectarWifi() {
  print_status(" Conectando", YELLOW);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    M5.Lcd.print(".");
  }
  M5.Lcd.fillScreen(BLACK);

  print_status(" WiFi OK", GREEN);
  delay(1000);
}

void setup() {
  M5.begin();
  M5.IMU.Init();
  Wire.begin();

  ConectarWifi();

  Udp.begin(PORT);
  tello_command_exec("command");
  delay(2000);
  tello_command_exec("speed 40");
}
```

A função **ConectarWiFi** tenta por tempo indeterminado se conectar a WiFi, imprimindo na tela "*WiFi OK*" quando conecta.

Já a função **setup**, padrão dos projetos programados via Arduino IDE, é a função executada quando a placa está ligando, sendo nela onde devemos fazer todas as configurações iniciais do controlador e do drone. Nessa função iniciamos a API do M5, conectamos a WiFi via **ConectarWiFi** e conectamos o websocket. Após isso é necessário enviar o comando "*command*" para a aeronave para ela iniciar o modo SDK e escutar os comandos que enviaremos a seguir. Por fim, configuramos a velocidade do drone com o comando "*speed 40*" para *40 cm/s*.

7.5. Função *loop* principal

Nessa função devemos definir os comandos que rodam em loop durante toda a operação do drone. Essa é mais uma função padrão de projetos programados via Arduino IDE.

```
void loop() {
  print_air_state();
  M5.IMU.getAccelData(&accX, &accY, &accZ);
```

No começo dela imprimimos o estado **isOnAir** e lemos a aceleração nos 3 eixos do sensor.

7.5.1. Mudança de estados

```
  // land / take off
  if (M5.BtnB.wasPressed()) {
    if (isOnAir) {
      tello_command_exec("land");
      delay(land_time);
    } else {
      tello_command_exec("takeoff");
      delay(takeoff_time);
    }
    isOnAir = !isOnAir;
    print_air_state();
  }

  // change state
  if (M5.BtnA.wasPressed()) {
    movementState += 1;
    if (movementState == 3) movementState = 0;
  }
  print_movement_state();
```

Nessa parte da função **loop** lemos os botões **A** e **B** para decidirmos se mudamos de estado ou não.

O botão **B** comanda pouso e decolagem, assim, se ele foi pressionado, checamos o estado **isOnAir** para decidir se enviamos o comando "*land"* ou "takeoff", alteramos o estado e imprimimos o novo estado.

O botão **A** altera entre os estados de voo. Se ele foi pressionado, mudamos para o próximo estado, de forma circular. Ao fim imprimimos o estado de voo atual com **print_movement_state**.

7.5.2. Interpretando as acelerações

```
if (isOnAir) {
  if (movementState == 0) {
    if (accY > max_sine) {
      tello_command_exec("back 50");
    } else if (accY < -max_sine) {
      tello_command_exec("forward 50");
    }

    if (accX > max_sine) {
      tello_command_exec("left 50");
    } else if (accX < -max_sine){
      tello_command_exec("right 50");
    }
  } else if (movementState == 1) {
    if (accY > max_sine) {
      tello_command_exec("up 50");
    } else if (accY < -max_sine) {
      tello_command_exec("down 50");
    }

    if (accX > max_sine) {
      tello_command_exec("ccw 30");
    } else if (accX < -max_sine){
      tello_command_exec("cw 30");
    }
  }
}
```

Esse bloco de código é o principal do sistema. Ele decide com base nas acelerações de cada eixo e estado de voo quais comandos enviar para o drone. A **accY** define o *pitch* e a **accX** define o *roll*. Se o estado isOnAir for falso, esse bloco é pulado, pois o drone não está no ar para receber esses comandos.

O primeiro sub bloco lida com o estado de voo horizontal: se *pitch* for positivo ou negativo envia "*back 50*" ou "*forward 50*" e se *roll* for positivo ou negativo envia "*left 50*" ou "*right 50*".

O segundo sub bloco lida com o estado de voo vertical: se *pitch* for positivo ou negativo envia "*up 50*" ou "*down 50*" e se *roll* for positivo ou negativo envia "*ccw 50*" (rotação anti horária) ou "*cw 50*" (rotação horária).

7.5.3. Lidando com keep-alive

```
  if (millis() - msg_time > 3000) { // keep-alive
    if (readState == 0) {
      tello_command_exec("battery?");
      readState = 1;
    } else if (readState == 1) {
      tello_command_exec("wifi?");
      readState = 2;
    } else if (readState == 2) {
      tello_command_exec("time?");
      readState = 3;
    } else {
      tello_command_exec("temp?");
      readState = 0;
    }
  }

  delay(pool_time);
  M5.update();
}
```

Ao fim da função **loop** é necessário fazer o check do *keep-alive*: o drone tem um sistema de segurança que se nenhuma mensagem for recebida por 15 segundos ele pousa. Assim verificamos ao fim de cada execução da função **loop** se a última mensagem foi enviada a mais de 3 segundos. Em caso positivo, o controlador envia um comando de leitura de uma das quatro propriedades: bateria, sinal da WiFi, tempo de voo e temperatura da aeronave, que será impresso na tela do controlador.

Por fim o programa espera o **pool_time** e atualiza a placa.

8. asdasd