Fabrícia Paola Bosman Barros

Desenvolvimento de um sistema de navegação do drone Tello

Campina Grande, Paraíba

Outubro de 2021

Fabrícia Paola Bosman Barros

Desenvolvimento de um sistema de navegação do drone Tello

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharela em Ciências no Domínio da Engenharia Elétrica.

Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática – CEEI

Departamento de Engenharia Elétrica – DEE

Orientador: George Acioli Júnior, D.Sc.

Campina Grande, Paraíba

Outubro de 2021

Fabrícia Paola Bosman Barros

Desenvolvimento de um sistema de navegação do drone Tello

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharela em Ciências no Domínio da Engenharia Elétrica.

Trabalho aprovado. Campina Grande, Paraíba, 18 de outubro de 2021:

George Acioli Júnior, D.Sc.
Orientador

Rafael Bezerra Correia Lima, D.Sc.
Convidado

Campina Grande, Paraíba
Outubro de 2021

Dedico a todos que ousam viver.

Agradecimentos

Os agradecimentos principais são direcionados aos meus pais, Péricles Barros e Heliante Bosman, bem como aos meus irmãos, Clarisse Pétua e Pedro Jacob, por todo o incentivo e apoio, seja profissional ou emocional, que sempre me deram. Não existem palavras que descrevam minha gratidão à vocês.

Agradeço aos mestres desta jornada, em especial: D.Sc. George Acioli, pela orientação no desenvolvimento deste e de outros trabalhos acadêmicos fundamentais na minha formação profissional; D.Sc. Rafael Lima, por ter aceitado compor a banca de avaliação deste trabalho; Ph.D. Péricles Barros, pela orientação profissional, pelos incentivos e inspirações.

Agradeço também a todos os outros professores e funcionários da Universidade Federal de Campina Grande pelos ensinamentos e experiências que vão além dos livros e por tornarem o ensino possível.

Ao meu companheiro, Caique Pessoa, e aos meus grandes amigos, Júlia Raposo, Olívia Gomes e Bruno Medeiros, sou imensamente grata por tê-los em minha vida e pela compreensão das ausências. Por fim, agradeço aos meus queridos amigos de curso e que levarei para a vida, em especial, Matheus Vilarim, Camila Machado, Matheus Rocha, Arthur Dimitri, Saulo Sobreira, Laís Souto e José Adeilmo. Sem vocês este curso não teria graça.

Resumo

Este projeto consiste no desenvolvimento de um sistema de navegação do drone Tello, da *startup* Ryze Technology. O sistema desenvolvido tem o objetivo de operação do drone, composto por uma Interface Gráfica do Usuário e pelo código responsável por enviar os comandos de deslocamento e de aquisição de dados ao drone. O Tello é uma Aeronave Remotamente Pilotada (RAP) caracterizada pela propulsão por quatro motores, classificando-o como um quadricóptero. O sistema foi desenvolvido como um aplicativo no *App Designer* do software MATLAB, em linguagem MATLAB e utilizando o MATLAB *Support Package for Ryze Tello Drones*. Neste trabalho são documentados os ensaios realizados, as ferramentas utilizadas e suas intruções de uso. Além disso, são apresentadas as etapas de construção de uma Interface Gráfica do Usuário, tomando referência conceitos do Design.

Palavras-chave: Tello, MATLAB, Navegação, App Designer, RAP, quadricóptero.

Abstract

This project consists of developing a navigation system for drone Tello, from startup Ryze Technology. The system developed has the objective of operating the drone, consisting of a Graphical User Interface and the code responsible for sending the displacement and data acquisition commands to the drone. The Tello is a Remotely-Piloted Aircraft (RAP) characterized by propulsion by four engines, classifying it as a quadricopter. The system was developed as an application in App Designer of MATLAB software, in MATLAB language and using MATLAB Support Package for Ryze Tello Drones. In this work, the tests performed, the tools used and their instructions for use are documented. In addition, the steps to build a Graphical User Interface are presented, taking into account Design concepts.

Keywords: Tello, MATLAB, Navigation, App Designer, RAP, quadricopter.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANAC	Agência Nacional de Aviação Civil
DJI	*Da-Jiang Innovations*
EPI	*Equipamento de Proteção Individual*
GUI	*Graphical User Interface*
HD	*Hard Disk*
IP	*Internet Protocol*
LED	*Light Emitting Diode*
MIT	*Massachusetts Institute of Technology*
RAP	*Remotely-Piloted Aircraft*
SDK	Software *Development Kit*
SI	Sistema Internacional de Unidades
UAV	*Unmanned Aerial Vehicle*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
VPS	*Visual Positioning System*
Wi-Fi	*Wireless Fidelity*

Lista de símbolos

Ψ	Ângulo de rotação no eixo Z - [°]
ϕ	Ângulo de rotação no eixo X - [°]
θ	Ângulo de rotação no eixo Y - [°]
cm/s	Centímetro por segundo
$\Delta\vec{r}$	Deslocamento vetorial - [m]
X	Eixo X
Y	Eixo Y
Z	Eixo Z
g	Grama
GHz	Giga Hertz
°C	Graus Celsius
Δt	Intervalo de tempo - [s]
MP	Megapixel
m	Metro
m/s	Metro por segundo
mAh	MiliAmpère-hora
km/h	Quilômetro por segundo
s	Segundo
$\vec{v}_{media}$	Velocidade vetorial média - [m/s]
$\vec{v}$	Velocidade vetorial instantânea - [m/s]
V	Volt
W	Watt

Sumário

1 Introdução

Neste capítulo serão apresentados os objetivos do projeto proposto, bem como a motivação para o seu desenvolvimento e a descrição da estrutura do documento.

1.1 Motivação

A Agência Nacional de Aviação Civil (ANAC) define as Aeronaves Remotamente Pilotadas (*Remotely-Piloted Aircraft* – RAP) como aeronaves não tripuladas pilotadas a partir de uma estação remota com finalidade diversa de recreação (ANAC, 2021). Apesar de que a regulamentação da ANAC não define formalmente o termo "drone", essa nomenclatura é utilizada e popularmente aceita ao se referir aos RAPs.

A origem do termo advém do inglês, cujo significado é zangão, zumbido, e os primeiros drones foram desenvolvidos com intuito bélico, conhecidos como *target drones*. Pertencentes a uma das classes de *Unmanned Aerial Vehicle* (UAV), eram aeronaves usadas para treinamentos e para testes dos sistemas militares. O seu financiamento e desenvolvimento foi intensificado a partir da Primeira Guerra Mundial. (KEANE et al., 2013)

Partindo disso, o avanço tecnológico no decorrer dos anos permitiu a evolução dos RAPs em termos da sua eficiência e da sua construção em diversas escalas. Como consequência, atualmente aeronaves não tripuladas de pequeno porte são regulamentas e ganham cada vez importância no uso civil. As áreas de aplicação dos drones são plurais, como o entretenimento, para fins científicos e para transporte de cargas (KARDASZ et al., 2016).

Em ambientes acadêmicos, a presença de RAPs auxilia o desenvolvimento de projetos, principalmente em locais de difícil acesso e quando evita-se a interferência humana. Um exemplo é o seu uso para pesquisa e monitoramento de peixe-boi-marinho e seu habitat (ALENCAR et al., 2020).

Dentre os drones de pequeno porte, o quadricóptero é o mais comum no mercado. Esse veículo aéreo não tripulado é caracterizado pela propulsão por quatro motores, cada um acoplado a asas rotativas (BRESCIANI, 2008). Normalmente equipado por sensores, como os de altura e de velocidade, o sistema de controle dos drones permitem com que sua trajetória seja controlada e verificada durante a execução de comandos. A partir disso, empresas do ramo educativo desenvolvem RAPs de pequeno porte, com preços mais acessíveis e que possibilitam o acesso e a manipulação das suas funcionalidades por meio de programação.

Dessa forma, o acesso aos dados obtidos por seus sensores permite com que sejam desenvolvidos sistemas de navegação, em que há a possibilidade de verificação das trajetórias percorridas, bem como da assertividade do comando programado. Conclui-se, assim, que os drones como objeto de estudo proporcionam avanços em áreas como eletrônica, desenvolvimento de sistemas e algoritmos de controle.

1.2 Objetivos

O Projeto de Conclusão de Curso proposto neste documento tem como objetivo geral o desenvolvimento de um sistema de navegação do drone Tello e a documentação do estudo. O sistema desenvolvido diz respeito à operação do drone, composto por uma Interface Gráfica do Usuário e pelo código responsável por enviar os comandos de deslocamento e de aquisição de dados ao drone. Assim, para sua conclusão, são definidos os seguintes objetivos específicos:

- estudar e apresentar as ferramentas e os conceitos utilizados;
- executar ensaios durante a construção do sistema;
- documentar e analisar o desenvolvimento;
- apresentar o sistema de navegação.

1.3 Estrutura do documento

O Capítulo 2 contém a fundamentação teórica. As temáticas discorridas são os princípios físicos e a dinâmica de um quadricóptero; o drone Tello e suas especificações; e as ferramentas utilizadas do software MATLAB.

O Capítulo 3 apresenta a preparação, os requisitos e os testes realizados no desenvolvimento do projeto.

O Capítulo 4 contém a apresentação e a descrição do sistema de navegação.

Por fim, o encaminhamento das conclusões e as propostas de melhorias do sistema apresentam-se no Capítulo 5.

2 Fundamentação Teórica

Neste capítulo serão descritos os conceitos utilizados da física, a dinâmica de um quadricóptero, as especificações do drone Tello e as ferramentas utilizadas no desenvolvimento do seu sistema de navegação.

2.1 Conceitos da Física

A seguir serão apresentados os conceitos da física relacionados ao movimento de um objeto, obtidos do livro "Fundamentos de física 1: mecânica 1" (HALLIDAY et al., 1996).

- Deslocamento

A localização de um objeto é realizada ao determinar sua posição em relação a um ponto de referência.

O deslocamento ($\Delta\vec{r}$) é uma grandeza vetorial, possuindo um módulo, direção e sentido, e é representado em metro (m) no S.I. (Sistema Internacional de Unidades). O módulo é o valor numérico e representa a distância entre as posições inicial e final. A direção e o sentido referem-se, respectivamente, ao eixo em que ocorre o deslocamento e ao sinal de mais ou de menos, indicando a posição em relação ao convencionado ponto de referência.

- Velocidade Média e Velocidade Instantânea

A velocidade é uma grandeza vetorial que indica o quão rápido um objeto se move. A velocidade média ($\vec{v}_{media}$) é a razão entre o deslocamento ($\Delta\vec{r}$) e o intervalo de tempo (Δt) em que esse deslocamento é realizado, como mostra a Equação 2.1. A sua unidade no S.I. é metro por segundo (m/s).

$$\vec{v}_{media} = \frac{\Delta\vec{r}}{\Delta t} \tag{2.1}$$

Considerando o caso em que o intervalo de tempo (Δt) tende a zero, obtém-se a variação do deslocamento em um instante de tempo, conhecida como velocidade instantânea. A representação é dada em função da derivada, taxa de variação instantânea, como pode ser observado na Equação 2.2.

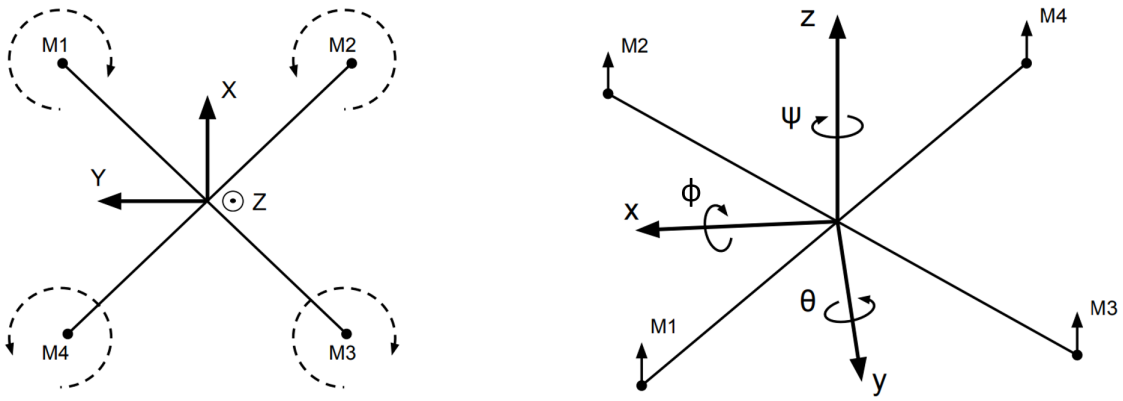
$$\vec{v} = \frac{d\vec{r}}{dt} \tag{2.2}$$

2.2 Dinâmica do Quadricóptero

O quadricóptero é uma das possíveis estruturas de um RAP e é caracterizado pela presença de quatro motores em uma configuração de cruz ou "x". Aos motores são acopladas hélices, que possuem eixos de rotação fixos e paralelos e são compostas por lâminas que distam de um passo fixo. O formato da lâmina é inclinado, fazendo com que o fluxo de ar aponte para baixo e o drone desloque para cima.

Em operação, para que o quadrator flutue, os pares de hélices laterais rodam em sentidos opostos, como pode ser observado no modelo simplificado de um drone com hélices em forma de "X" da Figura 1 (M1, M2, M3 e M4). Uma condição flutuante estacionária é caracterizada quando as quatro hélices rotacionam com a mesma velocidade. Nesse caso, a aceleração da gravidade é balanceada (BRESCIANI, 2008) e nenhuma força ou torque muda a posição do drone.

Figura 1 – Modelo simplificado de um quadricóptero



Fonte: Adaptação, (FERREIRA, 2015)

A movimentação do drone depende da diferença de velocidade dos motores. Assim, os movimentos básicos são (FERREIRA, 2015):

1. vertical (*throttle*), em que o drone desloca no eixo Z de acordo com o aumento ou a diminuição da velocidade dos propulsores de forma simultânea e em uma mesma quantidade;

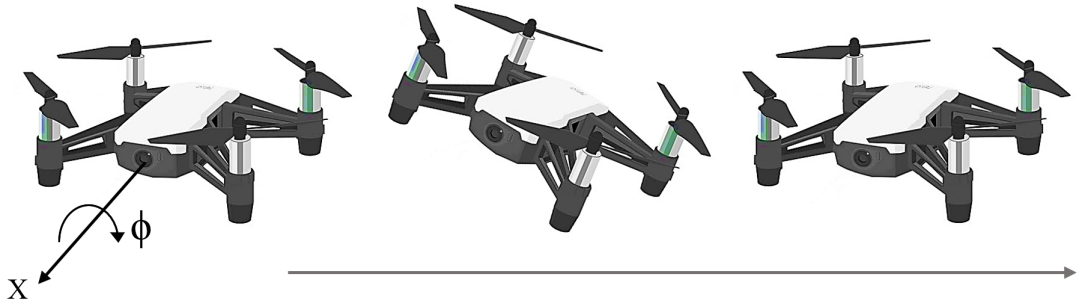
2. rotacional (ângulo de *yaw*, Ψ), em que o drone rotaciona no sentido horário ou anti-horário de acordo com o aumento da velocidade de um par de hélices transversais e a diminuição do par ortogonal;

3. horizontal (ângulos de *roll*, ϕ, e *pitch*, θ), em que o drone desloca nos eixos X ou Y a partir do aumento de velocidade de um par de hélices laterais e da diminuição da velocidade do par oposto.

A Figura 2 apresenta o exemplo de um deslocamento lateral, em que ocorre a variação no ângulo ϕ. No caso, o aumento na velocidade das hélices do lado esquerdo e a diminuição da velocidade do lado direito fazem com que o drone desloque para a direita.

Ademais, para que o drone pare, a partir do momento em que se encontra em movimento, o processo de variação da velocidade ocorre de forma análoga, mas oposta. No caso da Figura 2, as hélices do lado direito do quadricóptero aumentariam e do lado esquerdo diminuiriam.

Figura 2 – Exemplo do deslocamento do drone da esquerda para a direita



Fonte: Autoria própria, 2021, com a imagem do site (MATHWORKS, 2021)

2.3 Drone Tello

2.3.1 Ryze Technology

A Ryze Technology, presente no mercado desde 2017, é uma *startup* chinesa de tecnologia voltada para o desenvolvimento e venda de drones, possuindo o Tello e o Tello EDU como principais modelos. Em parceria com a DJI e a Intel, responsáveis respectivamente pelo sistema de controle de voo e pelo processador, a empresa apresenta drones para uso recreativo e educativo no âmbito da programação.

Nesse contexto, os drones da Ryze possuem duas possibilidades de controle. A primeira se dá por meio de um controle remoto compatível ou pelo aplicativo desenvolvido pela empresa (*Tello App*), que permitem o acesso às funcionalidades de deslocamento, manobras pré-definidas e visualização e captura de imagens em tempo real.

A segunda alternativa de controle é voltada para quem deseja aprender habilidades em programação. Os drones são programáveis em Scratch, sistema de programação desenvolvido pelo Media Lab do MIT, e é voltado para quem está começando na área, em especial as crianças. Além disso, é possível estabelecer uma conexão com o drone e enviar e receber dados via códigos mais avançados desenvolvidos com base nos SDKs (Software *Development Kits*).

2.3.2 Estrutura Física

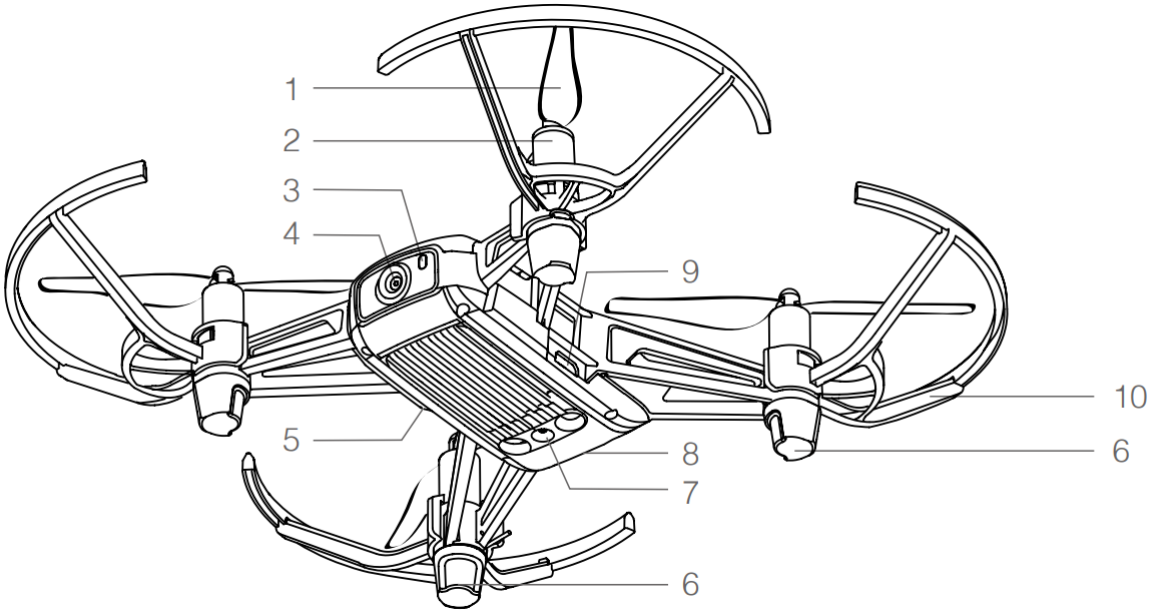
O presente trabalho estuda o drone Tello Boost Combo, modelo TLW004. O Tello é classificado como um quadricóptero de pequeno porte, com posição de hélices em forma de "X" e é voltado para voos em ambientes internos. A Tabela 1 contém os principais componentes do quadricóptero em estudo e a Figura 3 apresenta um diagrama indicando as respectivas localizações.

Tabela 1 – Principais componentes do drone Tello.

Número indicado na Figura 3	**Componente**
1	Hélices
2	Motores
3	Indicador do Estado da Aeronave
4	Câmera
5	Botão de Energia
6	Antenas
7	Sistema de Posicionamento de Visão
8	Baterias
9	Porta Micro USB
10	Protetores de Hélices

Fonte: Adaptação, (RYZE, 2018b)

Figura 3 – Diagrama da aeronave Tello



Fonte: Adaptação, (RYZE, 2018b)

Além disso, sua estrutura interna é composta pelo processador Intel Movidius Myriad e pelo sistema de controle de voo DJI. A documentação fornecida pela Ryze Tech (RYZE, 2020) aponta para a presença de um barômetro e, como será visto nas próximas seções, infere-se a presença de sensores de velocidade e de orientação (por ex., acelerômetro) a partir dos dados que o drone fornece.

2.3.3 Especificações

A Tabela 2 apresenta as principais especificações de operação, da câmera e da bateria do drone Tello. É válido lembrar que, apesar dos limites de altitude, é indicado manter o drone dentro dos limites de performance adequada do Sistema de Posicionamento de Visão.

Tabela 2 – Especificações do drone Tello.

Aeronave	
Peso	87g
Velocidade Padrão (modo *Slow*)	$10,8$km/h
Velocidade Máxima (modo *Fast*)	$28,8$km/h
Altitude Máxima	30m
Distância Máxima	100m
Deslocamento mínimo	$0,2$m em pelo menos um dos eixos
Tempo Máximo de Voo	13 minutos (0 vento em velocidade constante de 15km/h)
Faxia de Temperatura Operacional	0°C a 40°C
Faixa de Frequência Operacional	$2,4$GHz a $2,4835$GHz
Câmera	
Tamanho Máximo da Imagem	2592x1936 (5MP)
Formato da Foto	JPG
Modo de Gravação de Vídeo	HD: 1280x720 30p
Formato do Vídeo	MP4
Bateria	
Capacidade	1100mAh
Tensão	$3,8$V
Tipo de Bateria	LiPo
Energia	$4,18$Wh
Peso Líquido	25 ± 2g
Faxia de Temperatura de Carga	5°C a 45°C
Potência Máxima de Carga	10W

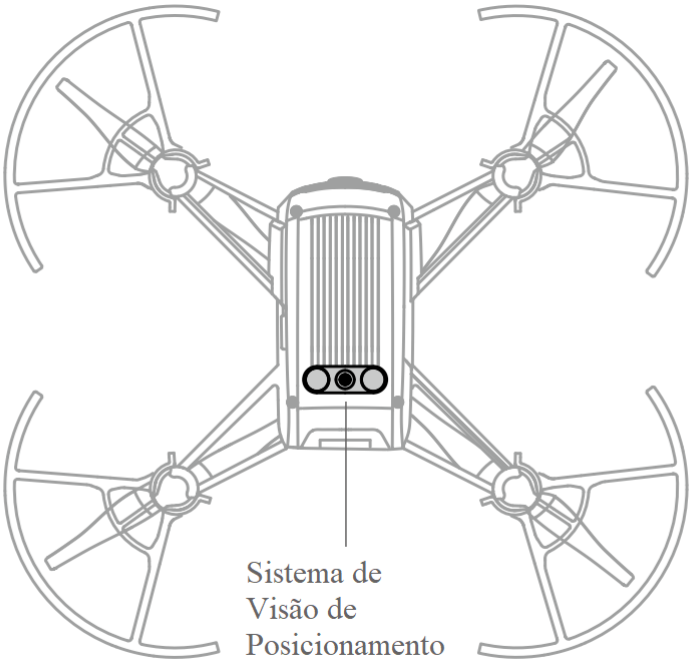
Fonte: Adaptação, (RYZE, 2018b)

A partir do momento em que o RAP decola, caso nenhum comando seja enviado no intervalo de 15 segundo, o Tello pousa.

2.3.4 Sistema de Posicionamento de Visão

O Sistema de Posicionamento de Visão (VPS - *Vision Positioning System*) do Tello é composto por uma câmera e um módulo infravermelho 3D localizados na parte inferior do drone, como pode ser observado na Figura 4. Ativado quando a aeronave é ligada, o sistema é responsável por ajudar na manutenção da posição atual no drone e é funcional quando o drone encontra-se entre $0,3$m a 30m de altitude, com máxima eficiência nas altitudes entre $0,3$m e 6m. Apesar de ser indicado no SDK, Software *Development Kit*, o material não fornece informações adicionais sobre o funcionamento dos componentes do VPS.

Figura 4 – Localização do Sistema de Visão de Posicionamento



Fonte: Adaptação, (RYZE, 2018b)

Localizada no meio do VPS, a câmera é utilizada na identificação dos padrões da superfície abaixo do drone. A partir do momento em que a câmera está ativa, o sistema interno do quadricóptero identifica nas imagens o número que representa a cor de cada pixel. O padrão é formado com a sequência de números identificados. Quando o Tello não está executando um comando de movimentação, mas o padrão é alterado, o sistema interno processa essa informação como um movimento (por exemplo, causado pelo vento) e executa o controle para que o drone volte à posição esperada, ou seja, ao mesmo padrão.

Os componentes laterais do VPS compõem o módulo infravermelho 3D, respectivamente na Figura 4 o receptor e o transmissor. O transmissor emite ondas infravermelhas que são refletidas pela superfície e identificadas pelo receptor. A partir da velocidade da onda e do tempo entre emissão e recepção, o sistema interno do quadricóptero calcula a distância até a superfície.

Assim, o módulo atua na prevenção de colisão e, juntamente com o barômetro interno, atua na identificação da altura em que o drone se encontra. De forma análoga às movimentações laterais, o sistema interno executa o controle que faz com que o drone retorne para o valor esperado de altura em caso de um deslocamento não desejado.

É válido ressaltar que determinadas características da superfície podem alterar a performance do Sistema de Posicionamento de Visão. Um exemplo é o caso da impossibilidade da câmera de identificar padrões em superfícies monocromáticas. Nesse contexto, caso o VPS seja afetado, o drone entra em modo de atitude (*Attitude/ATTI mode*), em que não consegue se posicionar e é facilmente afetado pelas condições do ambiente.

As principais situações que devem ser evitadas são:

1. voar abaixo de $0,5$m em alta velocidade;
2. voar em áreas com grande variação de iluminação;
3. voar em velocidades acima de 18km/h em alturas abaixo de 1m;
4. voar sobre superfícies: monocromáticas, altamente reflexivas, transparentes (por ex., água), muito escuras ($<$10 lux) ou claras ($>1000,000$ lux), que se movem, com padrões ou texturas sutis.

2.3.5 Kit de Desenvolvimento de Software (*SDK*)

O SDK, Software *Development Kit*, é um conjunto de ferramentas que permite com que programadores criem aplicações e funcionalidades para produtos digitais já existentes. No caso do quadricóptero em estudo, a Ryze Technology disponibiliza o SDK 2.0 *User Guide* (RYZE, 2018a) como o material de referência mais atualizado.

Assim, além de conter a arquitetura de comunicação do drone Tello, o SDK 2.0 apresenta os comandos dispoíveis de controle, de definição das configurações e de leitura. Cada comando é acompanhado da explicação correspondente e a possível resposta da aeronave.

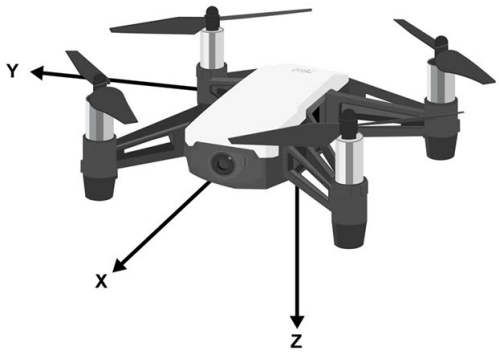
A Tabela 3 apresenta alguns dos comandos utilizados no desenvolvimento do projeto. É válido salientar que o drone realiza os deslocamentos de acordo com os seus eixos de coordenadas, apresentados na Figura 5.

Tabela 3 – Exemplo de comandos do SDK (RYZE, 2018a).

Comando	**Descrição**	**Possível Resposta**
Comando de Controle		
Command	Drone entra no modo SDK	*ok/erro*
takeoff	Decolagem automática	*ok/erro*
land	Pouso automático	*ok/erro*
streamon	Habilita transmissão de vídeo	*ok/erro*
streamoff	Desabilita transmissão de vídeo	*ok/erro*
up x	Move "x"cm para cima (x = [20, 500])	*ok/erro*
forward y	Move "y"cm para frente (y = [20, 500])	*ok/erro*
cw x	Rotaciona "x"graus - horário (x = [1, 360])	*ok/erro*
Comando de Definição		
speed x	Mude para a velocidade "x"cm/s	*ok/erro*
wifi ssid pass	Defina a senha "*pass*" do Wi-Fi "*ssid*"	*ok/erro*
Comando de Leitura		
speed?	Obter velocidade atual (cm/s)	"x"= [10, 100]
time?	Obter o tempo de voo (s)	"*time*"

Fonte: Adaptação, (RYZE, 2018a)

Figura 5 – Eixos de coordenadas do drone Tello



Fonte: (MATHWORKS, 2021)

Acerca da comunicação com o Tello, ela é realizada sem fio e via UDP, como será explanado a seguir. Existe a possibilidade de comunicação com o drone por protocolo *Bluetooth* aos aplicativos desenvolvidos pela Ryze Technology, contudo o SDK explicita apenas a conexão Wi-Fi.

Outra diferença é que, enquanto as especificações do drone indicam a velocidade padrão e máxima respectivamente como $10,8$km/h (300cm/s) e $28,8$km/h (800cm/s), o SDK apresenta que o programador pode variar a velocidade entre 10cm/s a 100cm/s.

Além da velocidade e do tempo de voo fornecidos pelo Tello, é possível obter dados

como a altura em relação ao ponto de decolagem, a aceleração angular, a porcentagem de bateria e as configurações do Wi-Fi.

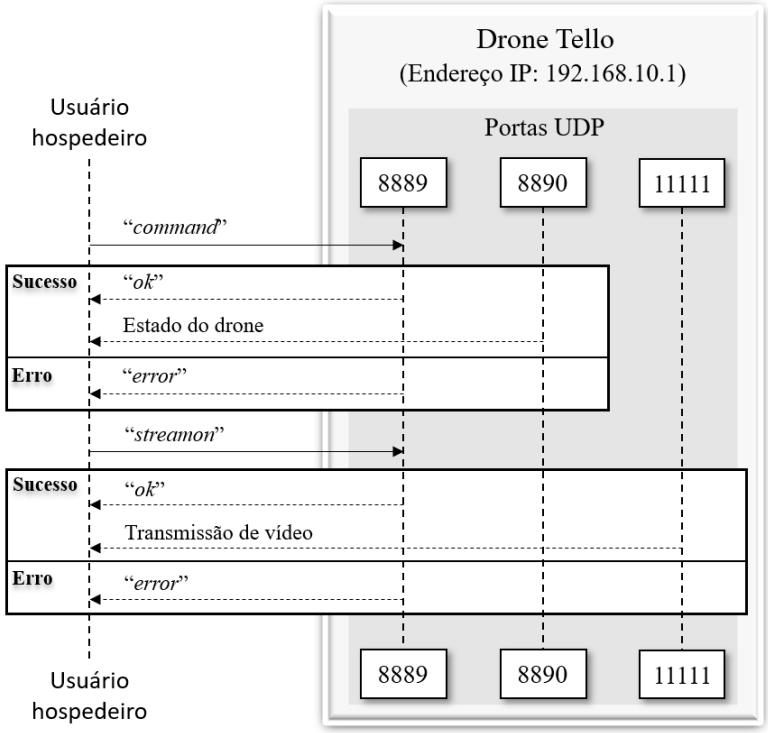
- Comunicação com UDP

Cada etapa do processo de troca de dados entre dois ou mais dispositivos compostos por hardware ou software é ditada por protocolos. No caso do drone Tello, o transporte dos dados na comunicação sem fio é realizado seguindo o *User Datagram Protocol*, UDP.

Sumariamente, algumas características desse protocolo são: ele é um modelo de transporte simplificado; ele não oferece garantias de que a mensagem enviada pelo emissor chegará ao receptor, caracterizando um serviço não confiável de transferência de dados; as mensagens obtidas pelo receptor podem chegar fora de ordem; e, por fim, há um controle no nível da aplicação sobre quais dados são enviados e quando. (KUROSE; ROSS, 2014)

A partir da Figura 6 é possível identificar o endereço IP do drone, as portas utilizadas e suas características, descritos no SDK. Além disso, o diagrama apresenta um exemplo da sequência de comunicação via UDP entre o drone e o usuário. O diagrama indica o comando emitido, as possíveis respostas do Tello e a associação entre a porta e o tipo de dado.

Figura 6 – Diagrama da comunicação UDP entre usuário e o drone Tello



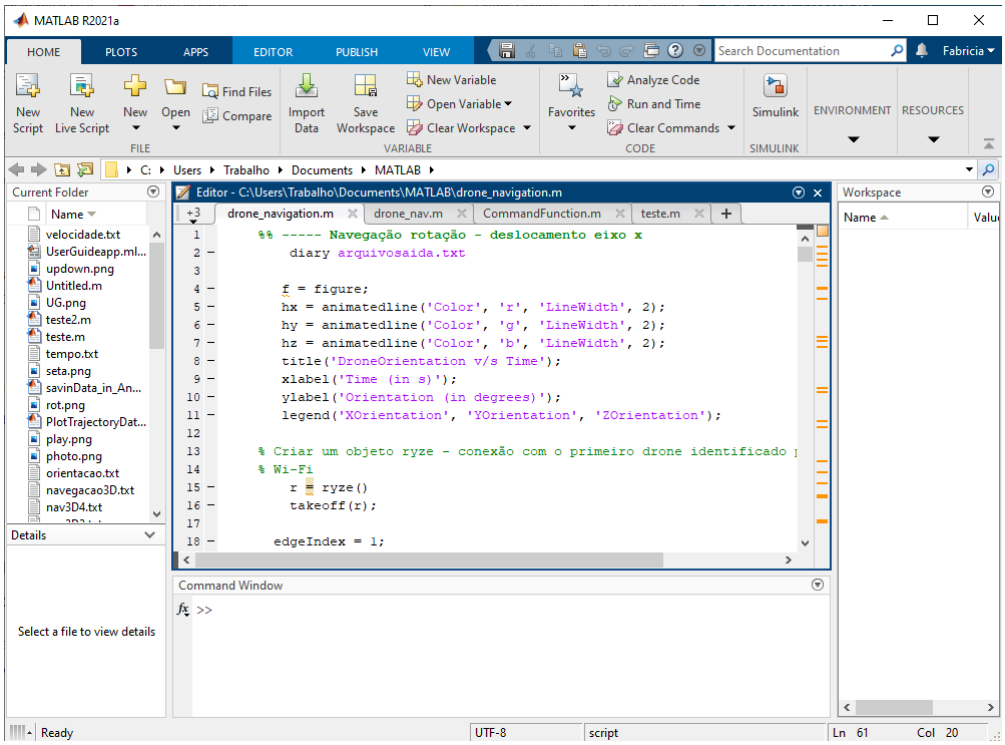
Fonte: Adaptação, (TUKIMAT, 2020)

2.4 Software MATLAB

O MATLAB é uma plataforma de programação e de computação numérica voltada para a análise de dados, o desenvolvimento de algoritmos e a criação de modelos (MATHWORKS, 1994-2021c). A sua página inicial, Figura 7, permite o acesso às ferramenta e configurações, na parte superior, e é composta pelos painéis:

1. pasta atual ou *Current Folder*, que permite o acesso de arquivos;
2. janela de comandos ou *Command Window*, local em que os comandos são digitados e as respostas são apresentadas;
3. editor ou *Editor*, que permite a criação de códigos em arquivos ".m" e, ao serem compilados, apresentam as respostas na janela de comandos;
4. espaço de trabalho ou *Workspace*, que permite o acesso aos dados criados ou importados de arquivos.

Figura 7 – Página inicial do MATLAB com *Editor* aberto



Fonte: Captura de tela do software MATLAB

Algumas das áreas em que a plataforma pode ser utilizada são robótica, sistemas de controle, aprendizado de máquina, processamento de sinais e de imagens, testes e medições. Essa vasta quantidade de aplicações é justificada pelas capacidades e ferramentas disponíveis.

No caso do projeto proposto, o sistema de navegação foi desenvolvido no MATLAB, versão 2021, e a Tabela 4 apresenta as principais ferramentas utilizadas e suas respectivas aplicações.

Tabela 4 – Principais ferramentas do MATLAB utilizadas no projeto.

Ferramenta	**Aplicação**
Pacotes	Download e uso de *toolboxes* oficiais do MATLAB
Hardware	Conexão do MATLAB com o drone Tello
Gráficos	Geração de gráficos com os dados obtidos do drone
Análise	Manipulação e interpretação dos dados obtidos
Desenvolvimento de algoritmo	Aplicação de funções matemáticas e algoritmos de processamento de dados
Construção de aplicativos	Criação do sistema como um aplicativo no *App Designer*
Compartilhamento web e desktop	Implantação do programa desenvolvido no MATLAB na web

Fonte: Adaptação, (MATHWORKS, 1994-2021c)

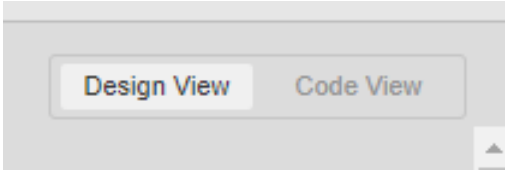
O pacote da plataforma utilizado no projeto foi o MATLAB *Support Package for Ryze Tello Drones* (MATHWORKS, 1994-2021b). O sistema de navegação foi realizado como um aplicativo no *App Designer* (MATHWORKS, 1994-2021d).

2.4.1 *App Designer*

Dentre as capacidades do MATLAB, o desenvolvimento de aplicativos para web e desktop se dá por meio do *App Designer*, Figura 9.

O *App Designer* é um ambiente integrado que permite com que o usuário construa a Interface Gráfica do Usuário (GUI, *Graphical User Interface*), no modo *Design View*, e escreva o código referente ao comportamento do aplicativo, modo *Code View*. O modo é selecionado na parte superior direita da tela, na região indicada na Figura 8.

Figura 8 – Região para alternar o modo entre *Design View* e *Code View*

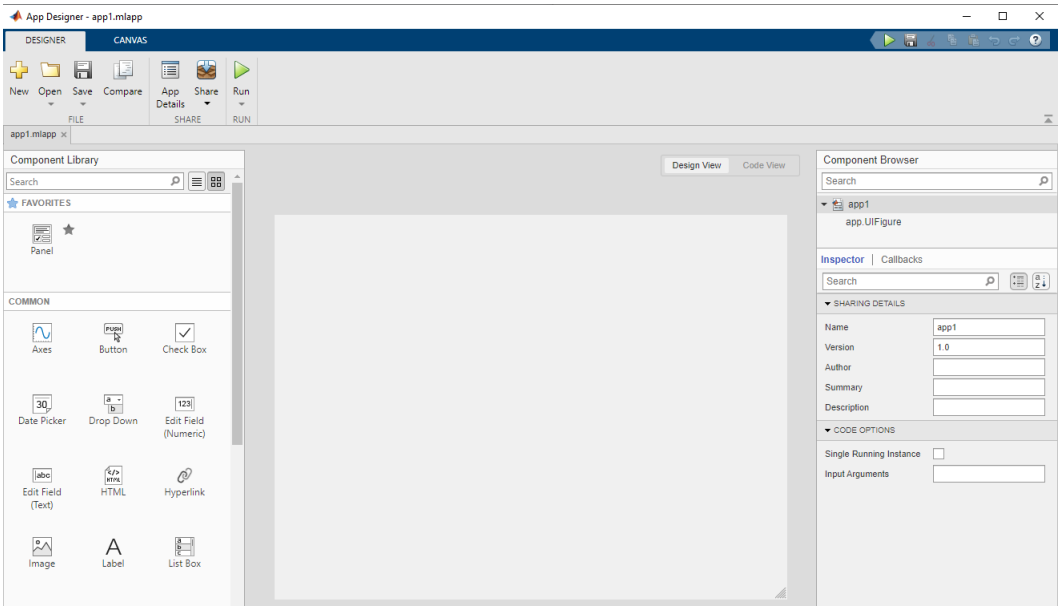


Fonte: MATLAB *App Designer*

A GUI é construída a partir do processo de "arrastar e soltar" (*drag and drop*) os componentes visuais, apresentados na parte esquerda da Figura 9. Cada elemento

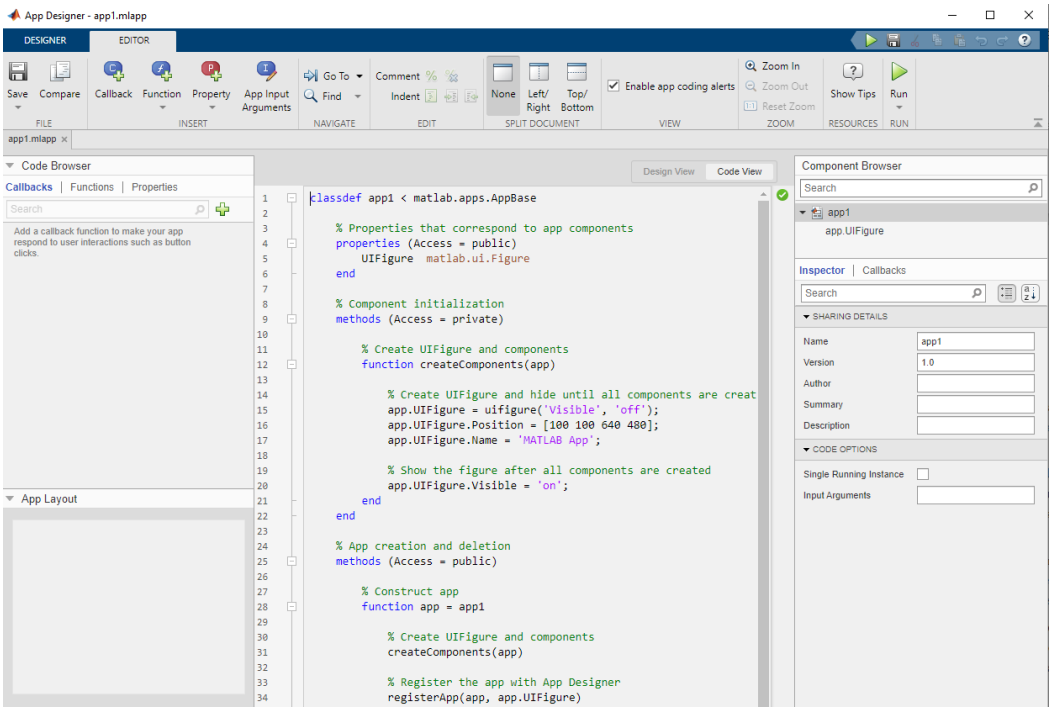
selecionado gera automaticamente um código na seção de edição que não pode ser alterado, Figura 10. Contudo, o usuário consegue inserir trechos de código ao criar eventos (*Callbacks*), funções e propriedades. O Apêndice B apresenta as etapas de como acessar o *App Designer* e as suas funcionalidades.

Figura 9 – Página inicial do *App Designer*



Fonte: MATLAB *App Designer*

Figura 10 – Página inicial do *App Designer*



Fonte: MATLAB *App Designer*

2.4.2 MATLAB *Support Package for Ryze Tello Drones*

O MATLAB *Support Package for Ryze Tello Drones* é o pacote que proporciona uma interface para o controle do Tello por meio do MATLAB (MATHWORKS, 1994-2021b).

Tabela 5 – Funções do MATLAB *Support Package for Ryze Tello Drones* utilizadas no projeto.

Sintaxe da função	**Descrição**
Conexão	
droneObj = ryze()	Conectar com o primeiro drone Ryze disponível no Wi-Fi
Navegação	
takeoff(droneObj)	Iniciar a decolagem do drone
land(droneObj)	Iniciar a aterrissagem gradual do drone
move(droneObj, displacement)	Deslocar o drone em cada eixo na distância relativa correspondente ao vetor *displacement* = [x y z]
*moveback(droneObj)**	Deslocar o drone para trás
*moveforward(droneObj)**	Deslocar o drone para frente
*moverdown(droneObj)**	Deslocar o drone para baixo
*moveup(droneObj)**	Deslocar o drone para cima
*moveright(droneObj)**	Deslocar o drone para a sua direita
*moveleft(droneObj)**	Deslocar o drone para a sua esquerda
turn(droneObj, angle)	Rotacionar o drone em um ângulo especificado (*angle*)
Aquisição de imagem	
camera(droneObj)	Conectar com a câmera do drone
snapshot(droneObj)	Capturar um *frame* do vídeo
preview	Apresentar dados de vídeo ao vivo
closePreview(droneObj)	Fechar janela de visualização da câmera
Dados de voo	
[height, time] = readHeight(droneObj)	Ler a altura atual do drone em relação à superfície de partida (*height*) e a hora que o sistema recebeu o dado (*time*, especificado em *datetime*)
[speed, time] = readSpeed(droneObj)	Ler a velocidade atual do drone (*speed*) e a hora que o sistema recebeu o dado (*time*, especificado em *datetime*)

Fonte: Adaptação, (MATHWORKS, 1994-2021b)

No caso, tomando como base as funções do SDK, previamente apresentado, foram desenvolvidas novas funções que permitem com que o programador conecte, controle

e adquira os dados do drone em um nível de programação mais alto. Assim, para o desenvolvimento do sistema de navegação foram utilizadas as funções do *Support Package* descritas na Tabela 5. É válido salientar que esse pacote não contém todos os possíveis comandos apresentados no SDK.

Como será observado na etapa de desenvolvimento, faz-se necessário criar um objeto no código que representa a conexão entre o MATLAB e o Tello. Dessa forma, cada função direcionada ao drone contém o objeto como argumento e, para fins de exemplo na Tabela 5, esse objeto é atribuído à variável chamada "droneObj".

Por fim, os comandos da Tabela 5 que são acompanhados por um asterisco indicam a possibilidade de inserir argumentos e de alterar configurações predefinidas. Dessa forma, a alteração é realizada por argumentos com o nome e o valor desejado, conforme consta na Tabela 6. O valor predefinido *true* do *WaitUntilDone* implica no bloqueio da execução da linha de comando do MATLAB até que o comando seja realizado. É válido salientar que a função *move* permite a alteração dos argumentos *Speed* e *WaitUntilDone*.

Tabela 6 – Possibilidades de argumentos das funções da Tabela 5 com asterisco.

Argumento	**Tipo de variável**	**Unidade de medida**	**Valor predefinido**
duration	Escalar real positivo	Segundos (s)	0.5 s
Speed	*double*	Metros por segundo (m/s)	0.4 m/s
Distance	*double*	Metros (m)	Não possui
WaitUntilDone			*true*
Exemplos			
moveback(droneObj,duration) (duration = 1)			
moveback(droneObj,1,'Speed',2)			
moveback(droneObj,'Distance',0.5,'Speed',0.4)			
moveback(droneObj,'Distance',2,'WaitUntilDone',false)			

Fonte: Autoria Própria, 2021

2.5 Design de Interação

A partir do momento em que um sistema é desenvolvido, a interação homem-máquina é posta como prioridade na definição da interface. No caso, o objetivo é de garantir sua usabilidade, evitando erros e experiências ruins por parte o usuário.

A interação com um sistema é realizada por meio de instruções, conversação, manipulação e exploração. Assim, os conceitos do design são aplicados em cada componente físico (por ex., botões), perceptível (por ex., gráficos) e conceitual (por ex., intruções de uso) da interface.

2.5.1 Regras de Ouro

O projeto proposto tomou como base as oito regras de ouro do design de interface, propostas no livro *Designing the user interface* (SHNEIDERMAN; PLAISANT, 2005) e apresentadas a seguir em tradução livre.

1. **Primeira regra: manter consistência**

A consistência é garantida quando: as mesmas sequências de ações são exigidas em situações semelhantes; as terminologias são idênticas em *prompts*, *menus* e telas de ajuda; os layouts, cores e fontes são empregados em todo o texto

2. **Segunda regra: atender às capacidades universais**

Com o intuito de melhorar a qualidade percebida do sistema, a interface deve atender às necessidades de usuários com os variados conhecimentos de tecnologia, especialidades, faixas etárias e deficiências.

Por exemplo, deve-se utilizar recursos para novatos, como explicações, e recursos para especialistas, como atalhos e ritmo mais rápido.

3. **Terceira regra: oferecer feedback informativo**

Para cada ação do usuário, deve haver feedback do sistema. Isso permite com que o usuário saiba a página em que se encontra, o que é apresentado, o que deve fazer e para onde será direcionado ao realizar uma ação.

4. **Quarta regra: projetar diálogos que indiquem o final de uma ação**

O feedback informativo na conclusão de um grupo de ações dá aos operadores a satisfação da realização. O usuário é comunicado quando a sequência de ações chega ao fim, evitando que surjam dúvidas em relação as suas ações.

5. **Quinta regra: prevenir erros**

O sistema deve evitar que os usuários cometam erros graves. Assim, a interface deve detectar o erro cometido e oferecer instruções simples, construtivas e específicas para a recuperação. Por exemplo, caso o usuário digite carateres do tipo letra em uma caixa de edição para números, o dado não deve ser processar e a interface deve comunicar o tipo de dado aceito.

6. **Sexta regra: permitir a reversão de ações**

Quando possível, o sistema deve permitir a reversão de ações, eliminando a preocupação do usuário de cometer algum erro.

7. **Sétima regra: oferecer a sensação de controle**

Em uma interface, busca-se que os usuários sejam os iniciadores das ações, em vez de os respondentes às ações. Assim, devem ser evitadas estruturas que incapacitam o usuário em obter as informações necessárias e em produzir a ação desejada.

8. **Oitava regra: reduzir a carga de memória de curto prazo**

A limitação do processamento de informações na memória de curto prazo dos humanos requer que as exibições sejam mantidas simples. Dessa forma, busca-se que a frequência do movimento da janela seja reduzida, bem como o tempo de treinamento para domínio básico da interface.

3 Desenvolvimento

O estudo e a compreensão dos princípios apresentados no Capítulo 2 formam a base para o desdobramento do projeto proposto. Assim, o sistema de navegação foi desenvolvido utilizando a linguagem MATLAB de programação, a interface gráfica de usuário *App Designer* e o MATLAB *Support Package for Ryze Tello Drones*, todos no software MATLAB.

Além dos materiais utilizados, neste capítulo serão descritos os ensaios executados no desenvolvimento do sistema de navegação e os seus requisitos.

3.1 Preparação

Os objetos necessários para o desenvolvimento do projeto são apresentados na Figura 11, com exceção do computador, e estão listados a seguir:

1. computador com o software MATLAB e conexão Wi-Fi;
2. drone Tello;
3. 3 baterias;
4. base para carregar as baterias;
5. cabo USB para carregar a bateria no drone ou na base;
6. óculos transparentes de proteção;
7. luva grossa;
8. fita métrica de 7,5m.

O Tello é um drone de pequeno porte e o seu uso para fins recreativos não demanda Equipamentos de Proteção Individual (EPI), principalmente quando são aplicadas medidas básicas de segurança (por ex., voando a uma distância de pessoas e fiações elétricas). Contudo, a partir do momento em que são realizados sucessivos testes de deslocamento do quadricóptero, aumentando a ocorrência de situações em que é necessário pegar o drone no ar, a presença de EPI é imprescindível.

O primeiro passo para iniciar os experimentos é conectar o drone ao computador via Wi-Fi. Em seguida, deve-se posicionar o drone seguindo as orientações do VPS, seção 2.3.4.

Figura 11 – Materiais utilizados no desenvolvimento do projeto

Fonte: Autoria Própria, 2021

- Conexão do drone

Ao pressionar o botão de energia, posição 5 da Figura 3, o indicador de estado da aeronave, posição 3, apresentará a sequência de cores: verde seguido do vermelho e, posteriormente, alternará entre as cores verde, amarelo e vermelho. A partir do momento em que o indicador piscar apenas a luz amarela, o drone pode ser identificado pelo Wi-Fi do computador.

Em seguida, o usuário segue os passos usuais de conexão de um computador a uma rede Wi-Fi. Caso o drone esteja com sua configuração de comunicação predefinida, o nome identificador começará com "TELLO" e a conexão é aberta, sem senha.

A partir do momento em que o MATLAB conecta-se ao drone, ou seja, um objeto é criado (por ex., *droneObj*, seção 2.4.2) e recebe os dados do estado inicial, o LED pisca na cor verde.

3.2 Requisitos do Sistema de Navegação

A partir do conhecimento das ferramentas do MATLAB, em especial o *App Designer*, foram definidos os requisitos do sistema de navegação. A Tabela 7 os apresenta em tópicos

gerais e específicos.

Tabela 7 – Requisitos gerais e específicos do sistema de navegação.

Requisitos gerais	**Requisitos específicos**
Controle manual do drone	1. Botões para decolar e pousar o drone 2. Controlar o deslocamento por meio de botões 3. Controlar o deslocamento por meio de valores inseridos para cada eixo
Controle automático do drone	4. Fazer com que o drone siga uma trajetória predefinida
Visualização do dados	5. Apresentar o deslocamento em tempo real em um gráfico 6. Apresentar as imagens capturadas pelo drone em tempo real
Orientações ao usuário	7. Apresentar um guia de instruções de uso 8. Apresentar aviso em situações, entradas e comandos não aceitos

Fonte: Autoria Própria, 2021

3.3 Ensaios

Como será observado a partir da descrição dos ensaios, o desenvolvimento do projeto iniciou com testes no *Editor* do MATLAB. O objetivo era estudar o comportamento do drone a partir dos comandos. Posteriormente, diante das informações obtidas e tomando como base os requisitos 7, a implementação foi realizada no *App Designer*.

3.3.1 Ensaio 1 - Leitura da Velocidade

O primeiro ensaio consistiu na avaliação do tempo de envio do comando pelo MATLAB e o tempo de resposta do drone. O código, apresentado no Apêndice D, consiste em:

1. estabelecer a comunicação com o drone;
2. enviar o comando de decolagem;
3. enviar sucessivos comandos de leitura de velocidade (40);
4. pousar o drone e encerrar a comunicação.

Os resultados obtidos são apresentados na Tabela 8 e indicam que o processo de enviar o comando e receber a leitura ocorre em milésimos de segundos.

Tabela 8 – Hora de recebimento das respostas ao comando de leitura de velocidade.

Envios de leitura	**Hora de recebimento da resposta**
20 solicitações iniciais	11 horas 33 minutos 51,61 segundos
20 solicitações finais	11 horas 33 minutos 51,71 segundos

Fonte: Autoria Própria, 2021

3.3.2 Ensaio 2 - Deslocamento e Velocidade

Neste ensaio, os testes com o drone foram baseados no exemplo *Read and Plot Navigation Data using* MATLAB (MATHWORKS, 1994-2021a). No caso, um gráfico é gerado em tempo real com a orientação do drone durante o seu deslocamento.

O código disponibilizado comanda o drone deslocar para frente $0,5$m a uma velocidade de $0,5$m/s e, após 2 segundos, comanda a sua rotação em 90° para a direita, em torno do eixo Z. O percurso é realizado quatro vezes, fazendo com que o drone realize o percurso de um quadrado, e o drone envia os dados de orientação enquanto desloca.

Assim, alterando minimamente o código-exemplo, Apêndice D, o ensaio consistiu em duas execuções para avaliar o desempenho do drone ao seguir os comandos. Foi verificado quanto tempo demorava para o drone executar a trajetória e foram marcados, com a fita métrica, o ponto de partida e chegada do drone. A partir desses valores medidos foi possível calcular a velocidade média desenvolvida.

Tabela 9 – Valores obtidos de cada deslocamento do drone.

Distância percorrida (m)	**Intervalo de tempo (s)**	**Velocidade média (m/s)**
Teste 1		
0,70	1,74	0,40
0,60	1,63	0,37
0,70	1,72	0,41
0,50	1,35	0,37
Teste 2		
0,50	1,8	0,28
0,60	1,64	0,36
0,85	1,77	0,48
0,70	1,69	0,41

Fonte: Autoria Própria, 2021

No caso, o intervalo obtido corresponde ao tempo de deslocamento do drone. A Tabela 9 apresenta os valores do intervalo e do deslocamento realizado (medido com a fita métrica). Assim, comparando-os em relação aos valores esperados de 0,5m e 0,5m/s,

pode-se inferir que o maior desvio foi de 15cm, em uma velocidade próxima à definida, e a velocidade mais divergente foi de 0,28m/s, em um deslocamento igual ao esperado.

3.3.3 Ensaio 3 - Deslocamento e Função Velocidade com *WaitUntilDone*

No Ensaio 3, o código (vide Apêndice D) foi desenvolvido pelo autor para teste de funções das Tabelas 5 e 6 e para avaliar o desempenho do drone. Em cada teste, o código comandava o drone decolar, deslocar 2m para frente e pousar. Após o percurso, o tempo de deslocamento até o início do pouso era anotado, bem como a posição final medida com a fita métrica.

Assim, para cada teste, a Tabela 10 apresenta: o valor de velocidade definido (v_d), a distância percorrida (d_p), o tempo de duração do deslocamento (t), a velocidade média calculada (v_m) e o valor que foi definido para o argumento *WaitUntilDone* (Tabela 6).

Tabela 10 – Valores obtidos nos testes do Ensaio 3.

Teste	v_d **(m/s)**	d_p **(m)**	t **(s)**	v_m **(m/s)**	***WaitUntilDone***
1	0,4	2,28	6,53	0,35	false
2	0,4	0,90	3,50	0,26	false
3	0,4	1,20	4,55	0,26	false
4	0,2	0,74	3,44	0,22	false
5	0,2	2,20	9,91	0,22	false
6	0,2	2,10	10,25	0,20	false
7	0,2	2,33	22,57	-	true

Fonte: Autoria Própria, 2021

No caso do teste 7, o valor *true* para o *WaitUntilDone* implica que não é possível obter a velocidade durante o deslocamento. Ao executar o comando de ir para frente, os comandos de leituras de velocidade só são executados quando o deslocamento é finalizado. Por essa razão que o tempo decorrido foi significativamente maior em relação outros testes e, ainda, usar esse valor no cálculo da velocidade média não seria fidedigno.

Nos demais casos, com o valor do argumento *false*, a leitura da velocidade era realizada durante o deslocamento do drone. Nos testes 2, 3 e 4, a distância percorrida foi inferior à esperada tendo em vista que, no código, o comando de pouso era enviado antes que o drone executasse todo o percurso. Como consequência e somado ao valor *false*, o Tello pousava assim que as leituras encerravam, um comportamento esperado.

Por fim, é válido salientar que as velocidades médias obtidas, principalmente para os testes 5 e 6, corresponderam ao valor definido. Em relação ao teste 1, infere-se que a aeronave percorreu o deslocamento esperado em menor tempo, em razão da velocidade

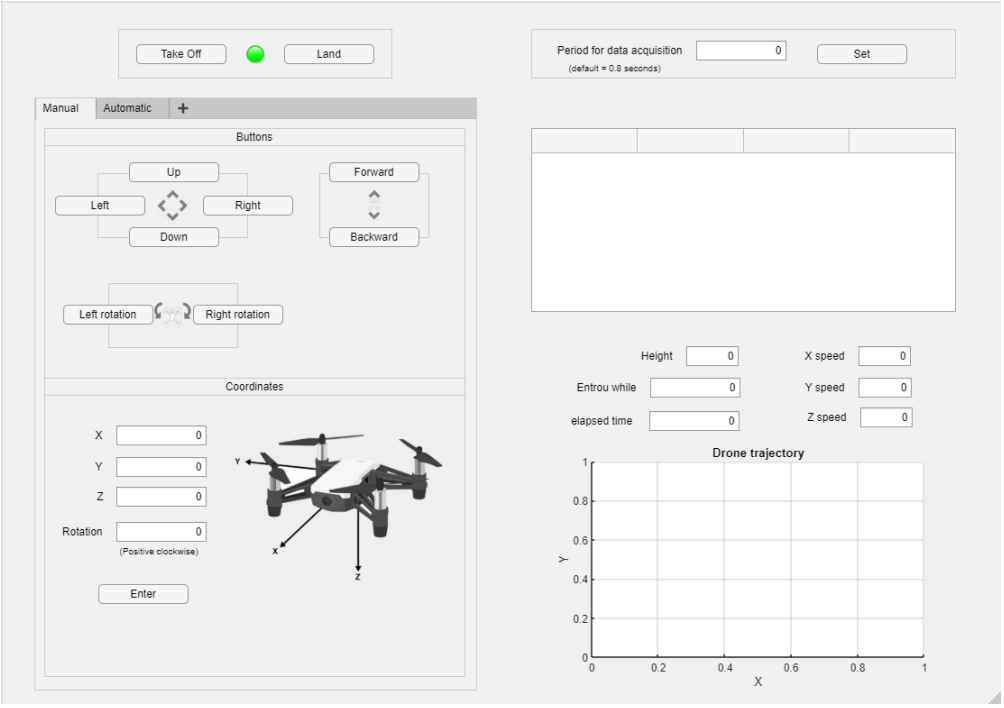
maior. Contudo, quando a execução do comando de deslocamento finalizou, o envio dos comandos de leitura perdurou.

3.3.4 Ensaio 4 - Controle manual (*App Designer*)

A partir dos requisitos, Tabela 7, os testes de construção do sistema de navegação iniciaram. Esse processo foi realizado de forma gradativa, tanto em relação à interface quanto ao código. Dessa forma, o Ensaio 4 consistiu no teste da GUI e do código desenvolvidos. O detalhamento de ambos será apresentado no Capítulo 4, em suas versões finais.

A Figura 12 exibe o primeiro protótipo desenvolvido. Partindo dele, o usuário controlava o drone por meio de botões ou ao inserir o deslocamento desejado em cada coordenada. Os comandos de leituras recebiam os dados do Tello, que eram apresentados por meio do gráfico e da tabela.

Figura 12 – GUI do primeiro protótipo do sistema



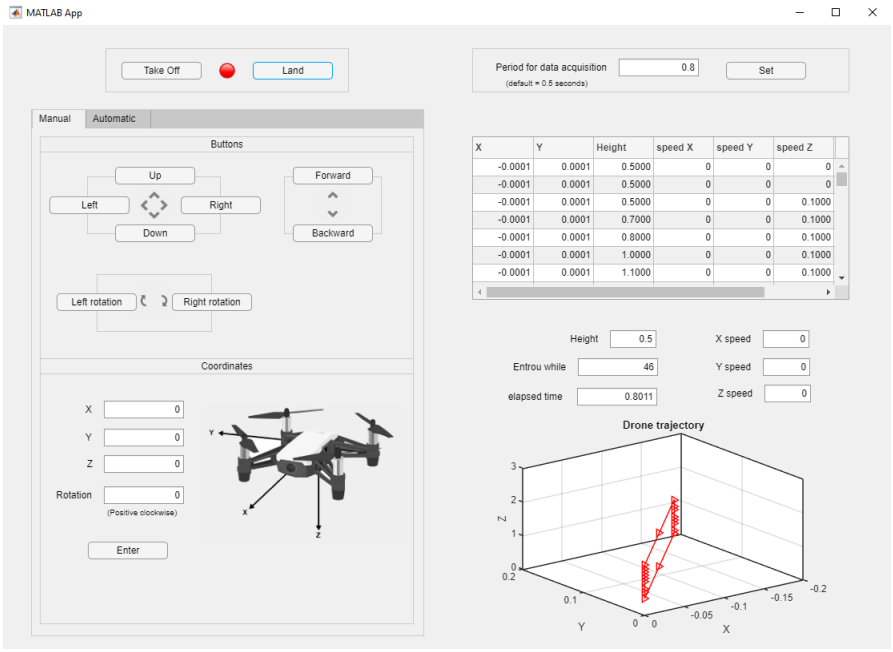
Fonte: Autoria Própria, 2021

Durante todos os testes, a distância percorrida ao apertar um botão era de 1m. Os valores variados eram referentes a velocidade e ao período entre comandos de leitura de dados. Assim, inicialmente foram testados intervalos de aquisição dos dados de 0,5s, 0,6s e 0,7s, mas o drone travou no ar nos três casos.

O teste com valor de 0,8s de amostragem foi executado. Nesse caso, a velocidade de deslocamento do drone foi definida como 0,1m/s (a mínima possível para que houvesse

um número maior de amostrar no percurso). Assim, considerando que os comandos e o delocamento do Tello foram realizados nos eixos X e Z, os dados obtidos apontavam um deslocamento também no eixo Y, como mostra o gráfico na Figura 13. Nesse caso, os comandos enviados foram para que o drone deslocasse para frente, para cima, para trás e para baixo, todos em um passo fixo.

Figura 13 – Teste com velocidade de deslocamento de 0.1m/s e amostragem em 0,8s



Fonte: Autoria Própria, 2021

Com o intuito de verificar o motivo, foram realizados dois testes: um variando o intervalo de aquisição de dados para 1s e mantendo a velocidade de 0,1m/s, com os comandos iguais ao da Figura 13; e o outro com a velocidade e o intervalo diferentes do primeiro caso, respectivamente, 0,2m/s e 1s (com a sequência de comandos de subir, direita, frente, direita, baixo e trás). O resultado em ambos os experimentos resultou também no gráfico com valores no Eixo Y, como exemplificam as Figuras 14 e 15.

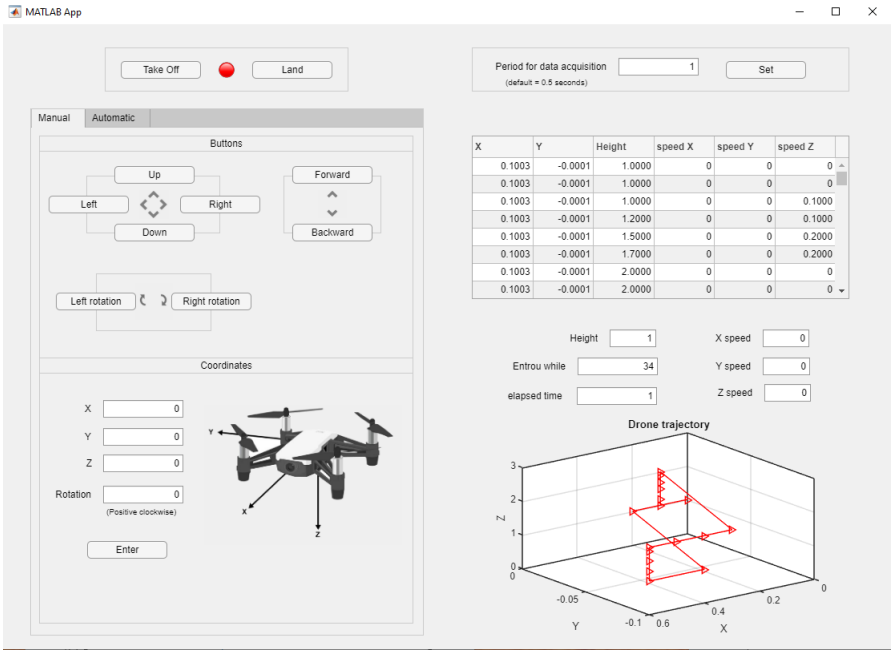
A Figura 15 apresenta os gráficos gerados no *Editor* do MATLAB com os dados obtidos no *App Designer*. O código que permitiu a análise dos dados e geração dos gráficos consta no Apêndice D.

Assim, ao analisar os gráficos e as tabelas de dados dos testes supracitados, foi possível identificar uma sequência de valores zero e valores solitários de -0.1 ou +0.1, como observa-se no Eixo Y na janela das velocidades na Figura 15.

Ademais, no mesmo instante em que aparecia a sequência "zero, valor diferente de zero e zero", percebeu-se que o drone estava finalizando a operação de deslocamento.

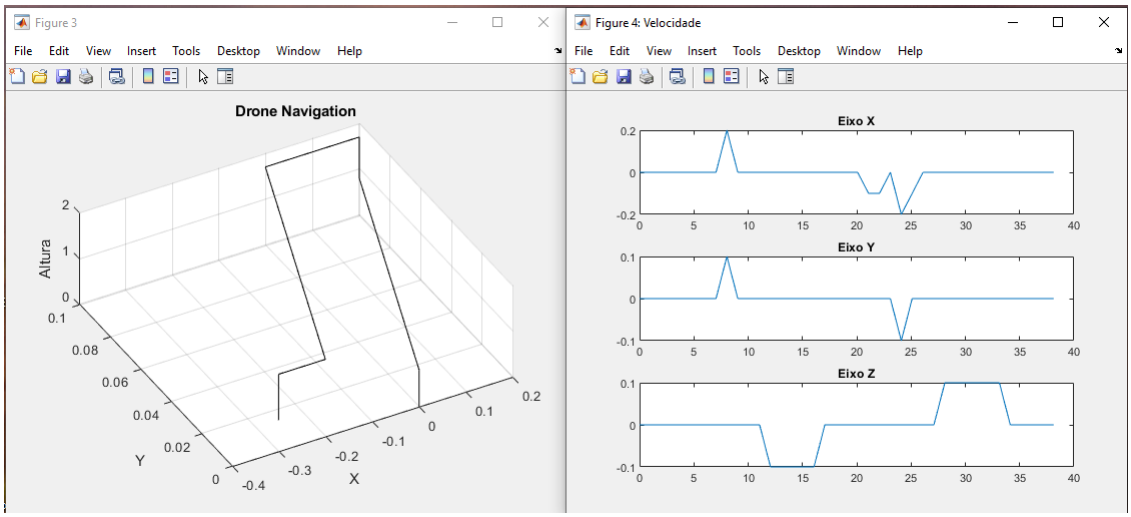
Isto é, quando o controle interno "freava" para finalizar a operação, os sensores do

Figura 14 – Teste com velocidade de deslocamento de 0.2m/s e amostragem em 1s



Fonte: Autoria Própria, 2021

Figura 15 – Teste com velocidade de deslocamento de 0.1m/s e amostragem em 1s - Da direita para a esquerda: gráfico da trajetória desenvolvida em função das leituras e gráfico da velocidade desenvolvida em cada eixo em função do tempo



Fonte: Autoria Própria, 2021

drone identificavam o valor diferente de zero no Eixo Y, resultante da guinada. Consequentemente, como a distância percorrida pelo drone é calculada a partir da multiplicação entre o tempo decorrido no deslocamento e somatório das leituras das velocidades instantâneas, o valor solitário era perpetuado na soma, inclinando o gráfico.

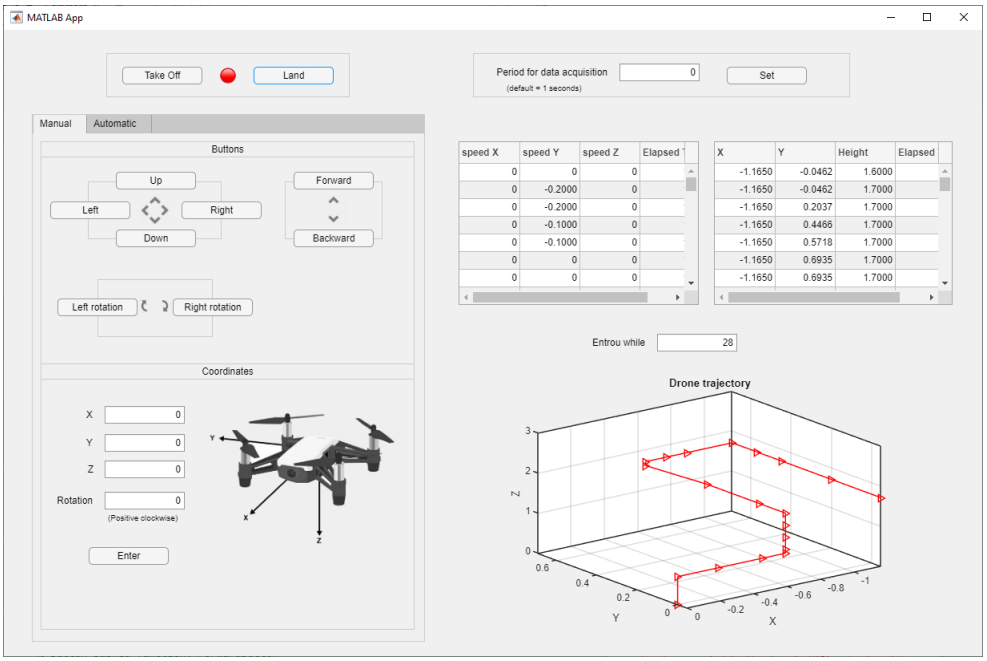
O problema foi resolvido por meio de um filtro. No caso, a curva no gráfico é gerada com uma amostra de atraso, permitindo com que, caso a sequência ocorra, o valor seja

retirado do cálculo da distância.

Considerando que a aeronave apresentava um comportamento não uniforme de velocidade quando estava a 0,1m/s, acelerando ao final do trajeto, definiu-se 0,2m/s como velocidade mínima de referência. Além disso, alguns dos testes realizados com a amostragem de 0,8s resultaram no travamento do drone, de forma que o valor padrão definido foi o de 1s.

Por fim, utilizando esses valores padrões, o drone executou os comandos e a trajetória gerada foi acertiva, como mostra a Figura 16.

Figura 16 – Teste com filtro, velocidade de deslocamento de 0.2m/s e amostragem em 1s



Fonte: Autoria Própria, 2021

3.3.5 Ensaio 5 - Controle automático (*App Designer*)

Em continuidade ao processo de construção do sistema de navegação com base nos requisitos da Tabela 7, o controle automático do drone foi desenvolvido.

No caso, o deslocamento automático do drone é comandado a partir de coordenadas armazenadas em uma matriz de deslocamentos. A matriz é composta por 3 colunas, que representam o deslocamento que o drone deve executar em cada eixo (X, Y e Z), e as linhas formam a sequência dos deslocamentos.

Assim, ao selecionar a trajetória, o aplicativo obtém a matriz de deslocamentos do *Workspace* e apresenta a trajetória esperada. Caso o usuário deseje que o drone execute o percurso, os comandos de deslocamento são enviados pelo MATLAB a cada 1 segundo e

com o respectivo dado da linha da matriz (primeira linha, primeiro deslocamento; segunda linha, segundo deslocamento; e assim por diante).

É válido ressaltar que as matrizes desejadas devem constar no *Workspace* do MATLAB. O Apêndice D apresenta os códigos desenvolvidos que geram matrizes de deslocamento para as trajetórias: de um quadrado, de degraus em diferentes eixos, de senoide no eixo X e de elipse.

A apresentação da trajetória e a execução do percurso pelo drone foram concretizadas com sucesso. Contudo, o sistema tem a limitação de não conseguir solicitar os dados de deslocamento durante esse processo, impossibilitanto a geração do gráfico.

Por fim, o Apêndice D apresenta os demais códigos utilizados na análise dos dados durante a realização dos ensaios.

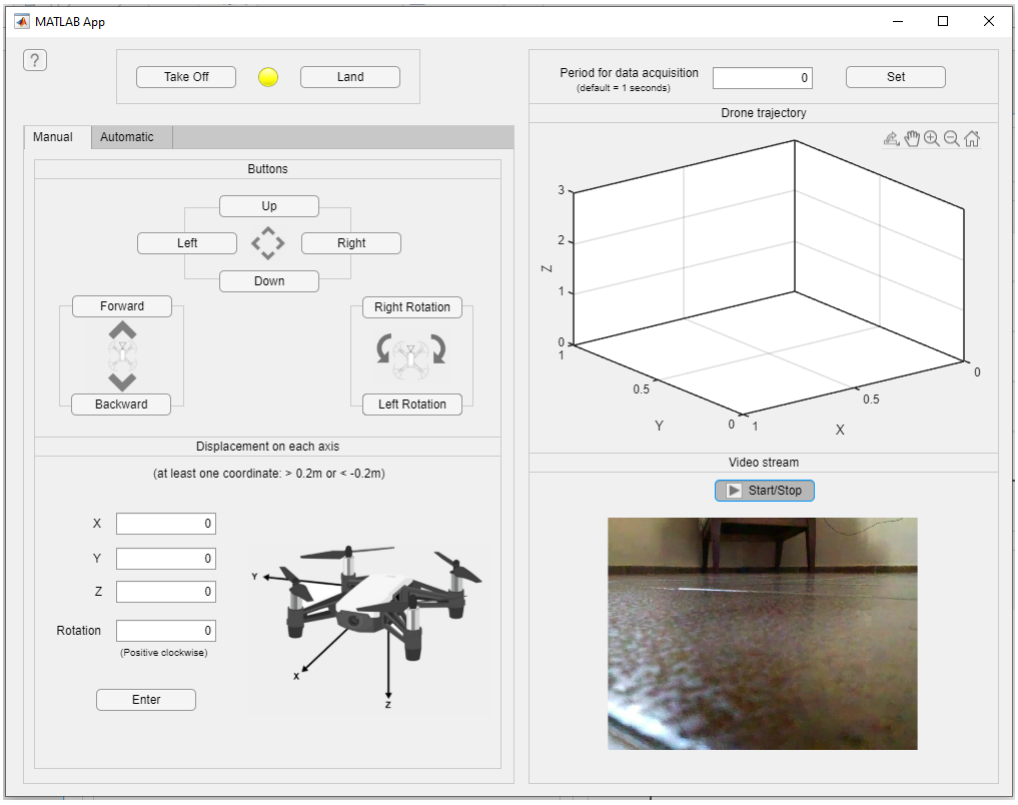
4 Sistema de Navegação do Drone Tello

Neste capítulo será apresentado o sistema de navegação do drone Tello. Composto por duas seções, a primeira descreve a Interface Gráfica de Usuário e a segunda discorre acerca do código desenvolvido.

4.1 Interface Gráfica do Usuário - *Design View*

A GUI foi elaborada a partir dos requisitos da Tabela 7 e buscando seguir as 8 regras de ouro do design de interface, seção 2.5.1. A Figura 17 apresenta o aspecto visual do sistema de navegação.

Figura 17 – Interface Gráfica do Usuário do sistema de navegação



Fonte: Autoria Própria, 2021

A seguir serão apontadas as particularidades da interface de acordo com os requisitos determinados.

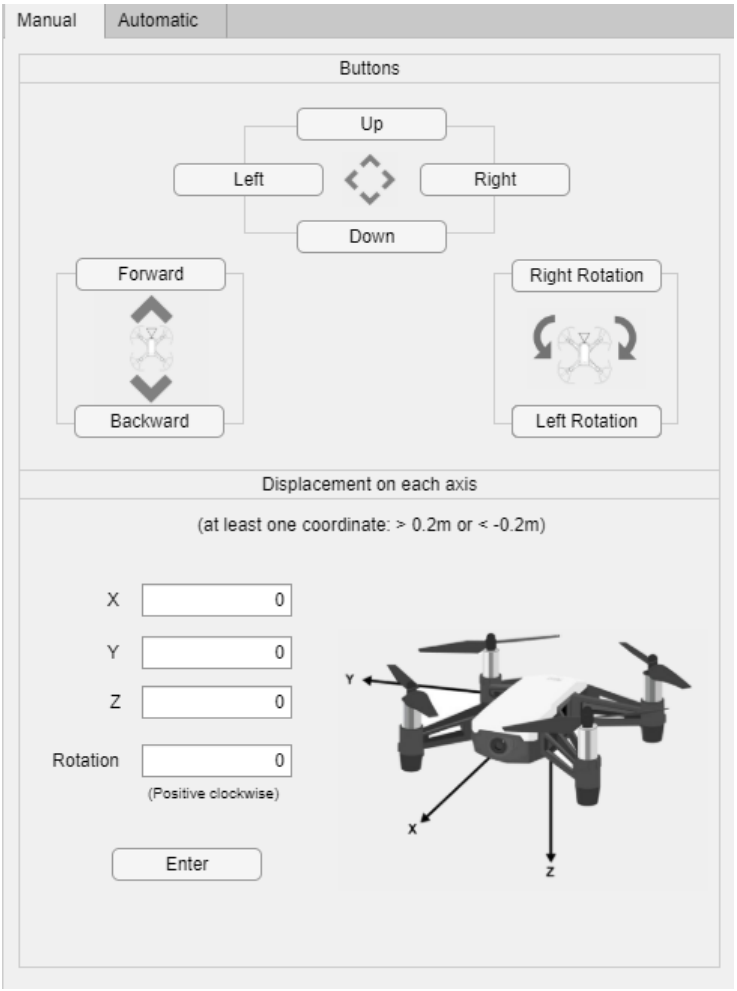
- Controle manual do drone

Os botões *Take Off* e *Land*, Figura 18, fazem com que o drone, respectivamente, decole ou aterrisse. Inicialmente, o lâmpada de estado apresenta a cor amarela, indicando que o drone ainda não entrou em operação. Caso o botão *Take Off* seja apertado, a cor passará para o verde, sinalizando que o Tello está em voando. Por fim, ao apertar para que a aeronave pouse, a lâmpada ficará vermelha.

Figura 18 – Botões *Take Off*, *Land* e lâmpada de estado

Fonte: Autoria Própria, 2021

Figura 19 – Abas *Manual* e *Automatic* e o controle manual



Fonte: Autoria Própria, 2021

A Figura 19 apresenta que a mudança do controle manual para o automático se dá por meio do acesso em abas. Ademais, pode-se observar a presença de botões que

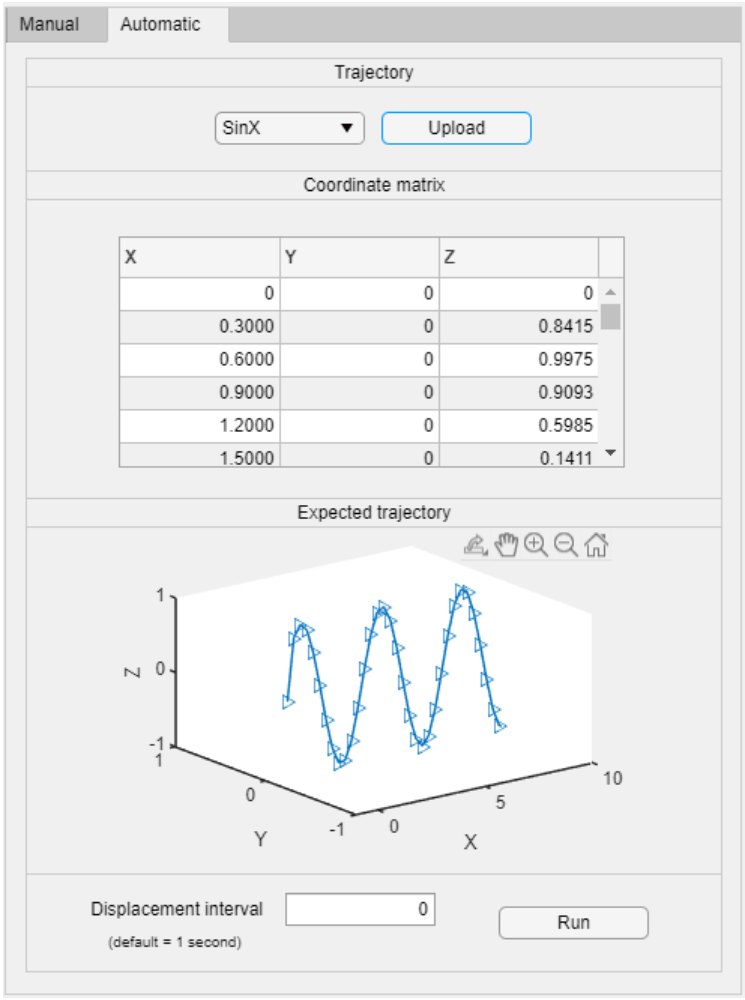
controlam o Tello no deslocamento nos 3 eixos, em todos os sentidos, e na rotação em torno do eixo Z.

Por fim, os componentes responsáveis pelo controle manual por meio de valores inseridos para cada eixo constam na área *Displacement on each axis*. Os valores são enviados para o drone a partir do momento em que o botão *Enter* é pressionado.

Em razão da especificação de um deslocamento de $0,2$m em pelo menos um dos eixos, Tabela 2, o usuário é informado do limite mínimo que deve inserir. Além disso, a imagem do drone com o seus eixos de coordenadas indica o sentido de deslocamento para valores positivos ou negativos.

- Controle automático do drone

Figura 20 – Controle automático



Fonte: Autoria Própria, 2021

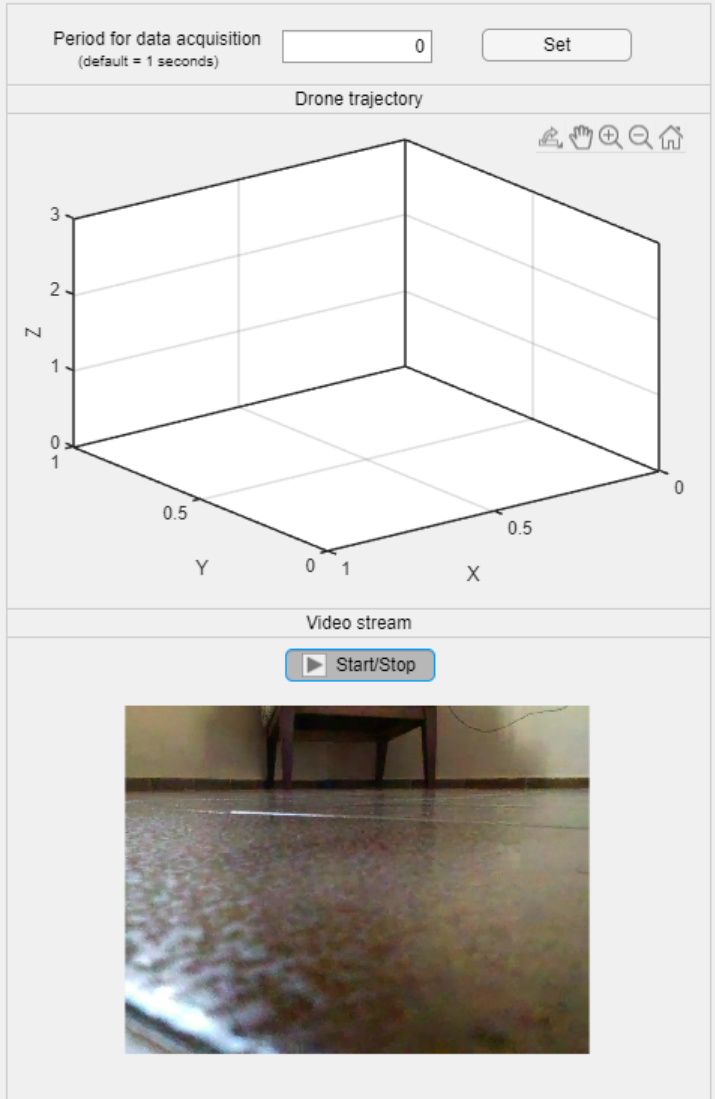
Ao selecionar a aba *Automatic*, o Tello receberá os comandos de acordo com deslocamentos predefinidos, caracterizando o controle automático.

Na Figura 20, na área *Trajectory*, o usuário seleciona o trajeto e pressiona o botão *Upload*. Em seguida, respectivamente nas regiões *Coordinate matrix data* e *Expected trajectory* serão apresentadas as coordenadas e a trajetória escolhida.

O espaço *Displacement interval* habilita o usuário a definir a taxa de envio dos comandos pelo MATLAB, predefinido como 1 segundo de intervalo. Por fim, os comandos são enviados ao drone ao acionar o botão *Run*.

- Visualização dos dados

Figura 21 – Visualização dos dados



Fonte: Autoria Própria, 2021

Em relação aos dados recebidos da aeronave, a interface apresenta em tempo real a trajetória percorrida e as imagens capturadas pela câmera do Tello. O gráfico *Drone trajectory*, Figura 21, é atualizado de acordo com o valor definido pelo usuário no espaço *Period for data acquisition* ou no intervalo predefinido de 1 segundo.

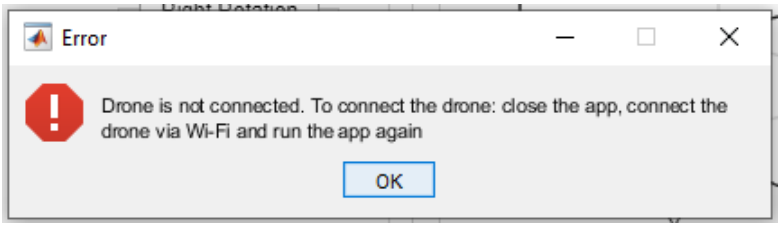
O botão *Start/Stop* é de estado. Quando acionado, as imagens capturadas são apresentadas no espaço *Video stream*, Figura 21. Ao selecioná-lo novamente, a transmissão é encerrada.

- Orientações ao usuário

Em sistemas interativos, a comunicação com o usuário é fundamental, principalmente para informá-lo sobre as condições de uso e caso ocorra algum problema. Assim, para garantir que cada ação do usuário resulte em uma consequência, evitando dúvidas, foram utilizadas caixas de mensagens de erro.

A Figura 22 apresenta a caixa de mensagem que aparece quando o aplicativo é iniciado e o drone não está conectado via Wi-Fi. A mensagem informa, em tradução livre, "Drone não está conectado. Para conectar o drone: feche o aplicativo, conecte o drone via Wi-Fi e inicie o aplicativo novamente" ("*Drone is not connected. To connect the drone: close the app, connect the drone via Wi-Fi and run the app again*"). Nessa condição, o mesmo aviso aparece caso algum botão de comando ou acesso de dados (por ex., câmera) seja apertado.

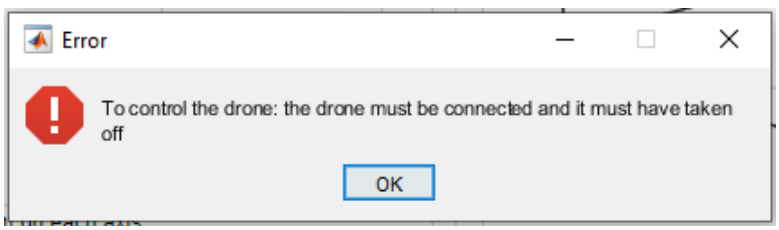
Figura 22 – Caixa de mensagem para avisar que o drone não está conectado



Fonte: Autoria Própria, 2021

A partir do momento em que o drone está conectado, não decolou e o usuário seleciona algum comando de deslocamento, o aviso da Figura 23 aparece. Ele indica que "Para controlar o drone: o drone deve estar conectado e precisa ter decolado" ("*To control the drone: the drone must be connected and it must have taken off*").

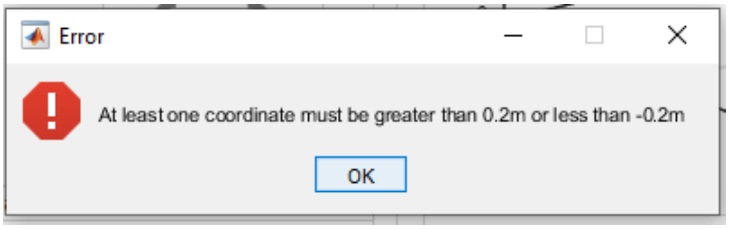
Figura 23 – Caixa de mensagem para avisar que o drone precisa decolar para ser controlado



Fonte: Autoria Própria, 2021

Como apresentado na Figura 19 e na Tabela 2, pelo menos um dos valores inseridos de X, Y ou Z na área *Displacement on each axis* deve ser maior que $0,2$m ou menor que $-0,2$m. Dessa forma, se a especificação não for seguida e o usuário pressionar o botão *Enter*, surge a caixa de mensagem da Figura 24. Sumariamente, a notificação informa que os limites a serem seguidos.

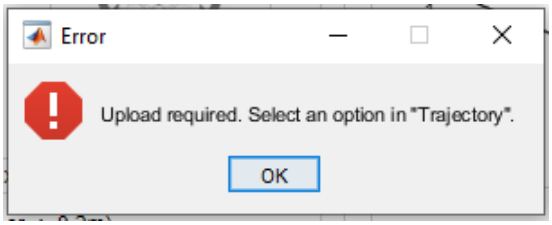
Figura 24 – Caixa de mensagem para avisar os valores aceitáveis na área *Displacement on each axis*



Fonte: Autoria Própria, 2021

Em relação ao controle automático, faz-se necessária a seleção da trajetória a ser desenvolvida pelo drone. Consequentemente, caso o usuário pressione o botão *Run* sem definí-la, a mensagem da caixa de erro apresenta que "Envio (da trajetória) necessário. Selecione uma opção na área *Trajectory*" (*"Upload required. Select an option in Trajectory"*), Figura 25.

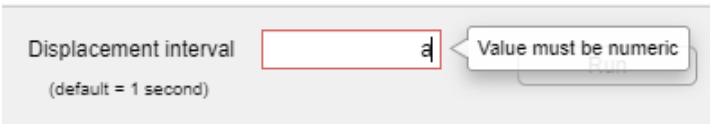
Figura 25 – Caixa de mensagem para avisar que é necessária a seleção da trajetória no controle automático



Fonte: Autoria Própria, 2021

Ademais, as caixas de texto da área *Displacement on each axis*, de *Displacement interval* e de *Period for data acquisition* só aceitam valores numéricos. A Figura 26 apresenta o exemplo da notificação ao usuário, "Valor deve ser numérico" (*"Value must be numeric"*), caso ele insira valores não aceitáveis, como uma letra.

Figura 26 – Notificação para valores não aceitáveis inseridos nas caixas de texto

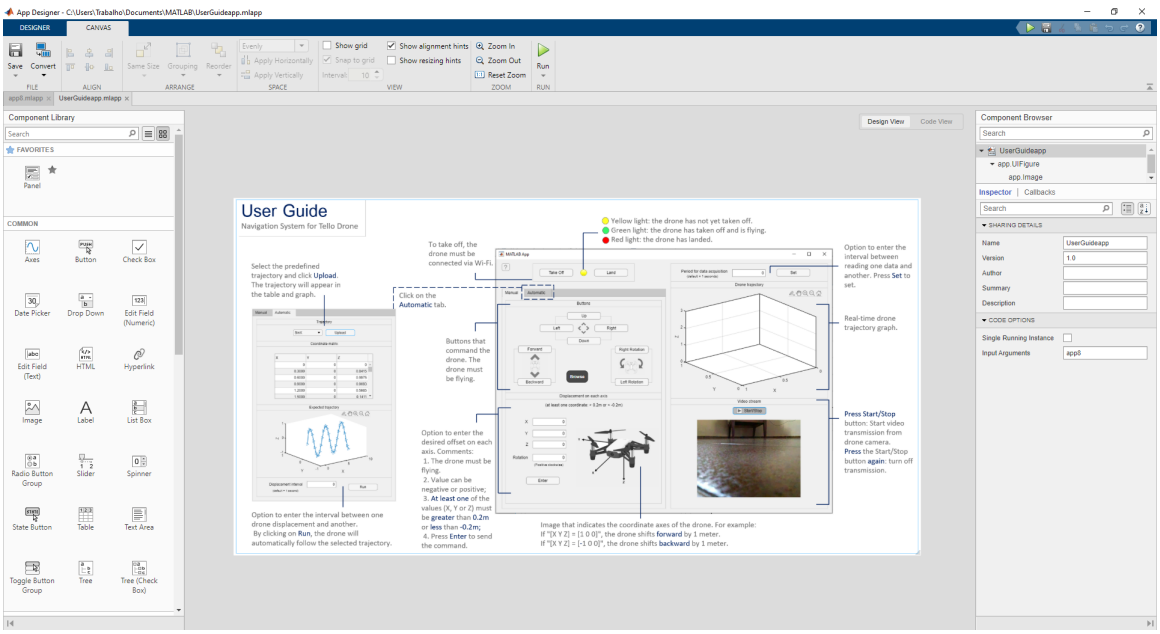


Fonte: Autoria Própria, 2021

Por fim, no Apêndice A é possível observar o Guia de Instruções do Usuário (*User Guide*). Ele é disposto na tela quando o botão com o símbolo do ponto de interrogação é pressionado. O Guia contém informações acerca do sistema de navegação, indicando desde o significado das cores da lâmpada até as possíveis opções e ações do usuário.

Para a abertura do guia nas proporções desejadas, foi necessária a criação de um novo aplicativo, cujo código será explanado na seção a seguir. A Figura 27 apresenta o *Design View* do aplicativo *UserGuideapp*.

Figura 27 – *Design View* do aplicativo *UserGuideapp*



Fonte: Autoria Própria, 2021

4.2 Código - *Code View*

O desenvolvimento no *Code View* do *App Designer* utiliza a linguagem MATLAB. Como apresentado na seção 2.4.1, os componentes inseridos no *Design View* geram códigos de inicialização que não podem ser alterados. Dessa forma, esta seção aborda o código desenvolvido pelo autor em trechos, enquanto o código desenvolvido pelo autor consta na íntegra no Apêndice C,

A inicialização das variáveis é realizada no trecho de código *properties*, indicando as propriedades do aplicativo. A Figura 28 apresenta as variáveis do código e suas respectivas explicações em inglês.

Após o trecho *porperties*, existe o trecho *methods*. Ele contém funções chamadas em outras partes do código. No caso, as suas funções são:

Figura 28 – Trecho de código *properties*

```
properties (Access = private)
    r;              % Ryze object - drone
    Axes;           % UIAxes object for Data and Plot
    cameraObj;      % Camera object

    speedMatrix;        % Matrix for storing speed data acquired from the drone
    displacementMatrix; % Matrix for storing displacement data calculated
    workspaceMatrix;    % Matrix with coordinates defined in the workspace

    % Auxiliary variables of message box
    aux_control_disabled = 0; % Indicates the drone is not connected
    aux_upload_enabled = 0;

    % Buttons Panel
    distance = 1;             % Standard distance for displacement
    speed = 0.2;              % Standard speed for displacement
    angleLeft = deg2rad(-45); % Standard angle for rotation
    angleRight = deg2rad(45); % Standard angle for rotation

    % Reading data
    speedR;                   % Drone speed obtained
    time;                     % Datetime when MATLAB receives speed read
    height;                   % Drone height obtained
    Xspeed;                   % Drone speed X axis
    Yspeed;                   % Drone speed Y axis
    Zspeed;                   % Drone speed Z axis
    deltaX;                   % X axis - Distance traveled between the two speed readings
    deltaY;                   % Y axis - distance traveled between the two speed readings
    coordX = 0;               % Drone position X axis
    coordY = 0;               % Drone position Y axis
    elapsedTime = 0;          % Variable to save the elapsed time - tic and toc
    auxFirst = 0;             % Auxiliary variable - first time DataAcquisitionFnc is called
    auxTime = 0;              % tic and toc auxiliary variable - deltaTime
    DroneInTheAir = 0;        % Variable to enter while and message box auxiliary variable
    pauseDataAndPlot;         % Pause value changed in the UI

    % Workspace Coordinates
    row = 1;
    sizeWorkspaceMatrix;
    moveX = 0;                      % X coordinate to which the drone should move
    moveY = 0;                      % Y coordinate to which the drone should move
    moveZ = 0;                      % Z coordinate to which the drone should move
    displacementInterval;           % Displacement interval changed in the UI

    % User Guide app - Appears when question mark icon is pressed
    UserGuideapp;
end
```

Fonte: Autoria Própria, 2021

1. *DataAndPlotFcn*, responsável por solicitar os dados ao drone, usá-los no cálculo do deslocamento, apresentá-los no gráfico e inserí-los nas matrizes de velocidades, *speedMatrix*, e de deslocamentos, *displacementMatrix*, apresentado na Figura 29;

2. *WorkspaceCoordinatesFnc*, que lê as linhas da matriz de deslocamentos, obtida do *Workspace*, e envia os comandos ao Tello, apresentado na Figura 30;

3. *connection_mb*, apresenta a caixa de mensagem informando que o drone não está conectado, a estrutura é análoga à apresentada na Figura 31;

4. *takeoff_mb*, apresenta a caixa de mensagem informando que o drone não decolou, a

estrutura é análoga à apresentada na Figura 31;

5. *errorvalue_mb*, apresenta a caixa de mensagem avisando as condições das variáveis, a estrutura é análoga à apresentada na Figura 31;

6. *upload_mb*, apresenta a mensagem de que é necessário escolher uma opção de trajetória, apresentado na Figura 31.

Figura 29 – Trecho de código *methods* com a função *DataAndPlotFcn*

```
methods (Access = private)

    function DataAndPlotFnc(app)
        while (app.DroneInTheAir)
            % Record elapsed time - Used to calculate drone position
            if (app.auxFirst ~= 0)
                app.elapsedTime = toc;
            end
            tic;

            % Read drone speed and height
            [app.speedR,app.time] = readSpeed(app.r);
            app.height = readHeight(app.r);

            app.Xspeed = app.speedR(1);
            app.Yspeed = app.speedR(2);
            app.Zspeed = app.speedR(3);

            % Save speed data in speedMatrix
            app.speedMatrix = circshift(app.speedMatrix, 1);
            app.speedMatrix(1,:) = [app.Xspeed app.Yspeed app.Zspeed app.elapsedTime];

            % Calculate the distance traveled between the two speed readings
            app.deltaX = (app.Xspeed)*(app.elapsedTime);
            app.deltaY = (app.Yspeed)*(app.elapsedTime);

            % Calculate the distance traveled between the two speed readings
            % X coordinate - Filter for non-displacement velocity data
            if ((app.speedMatrix(1,1) == 0) && (app.speedMatrix(3,1) == 0))
               app.coordX = app.displacementMatrix(2,1);
               app.displacementMatrix(1,1) = app.displacementMatrix(2,1);
            end

            % Calculate the distance traveled between the two speed readings
            % Y coordinate - Filter for non-displacement velocity data
            if ((app.speedMatrix(1,2) == 0) && (app.speedMatrix(3,2) == 0))
               app.coordY = app.displacementMatrix(2,2);
               app.displacementMatrix(1,2) = app.displacementMatrix(2,2);
            end

            % Total Distance - Actual Drone Position
            app.coordX = app.deltaX + app.coordX;
            app.coordY = app.deltaY + app.coordY;

            % Save data in dataMatrix
            app.displacementMatrix = circshift(app.displacementMatrix, 1);
            app.displacementMatrix(1,:) = [app.coordX app.coordY app.height app.elapsedTime];

            % Plot UIAxes - Delay in one reading
            addpoints(app.Axes, app.displacementMatrix(2,1), app.displacementMatrix(2,2), ...
                app.displacementMatrix(2,3));
            drawnow

            app.auxFirst = 1;
            pause(app.pauseDataAndPlot);
        end

    end
```

Fonte: Autoria Própria, 2021

Ao definir as variáveis em *properties*, o seu acesso em outras partes do código é realizado por referência. Dessa forma, como pode ser observado na Figura 29, são utilizados os caracteres "app." seguido da variável desejada para acessá-la. A mesma estrutura é escrita quando deseja-se acessar as propriedades de um objeto ou componente definido na GUI.

Figura 30 – Trecho de código *methods* com a função *WorkspaceCoordinatesFnc*

```
function WorkspaceCoordinatesFnc(app)
    while (app.row <= app.sizeWorkspaceMatrix)
        % Move drone
        app.moveX = app.workspaceMatrix(app.row,1);
        app.moveY = app.workspaceMatrix(app.row,2);
        app.moveZ = app.workspaceMatrix(app.row,3);

        move(app.r,[app.moveX app.moveY app.moveZ], 'Speed', app.speed, 'WaitUntilDone', ..
           | false);

        app.row = app.row + 1;
        pause(app.displacementInterval)
    end

end
```

Fonte: Autoria Própria, 2021

Figura 31 – Trecho de código *methods* com a função *upload_mb*

```
    function upload_mb(app)
        msgbox('Upload required. Select an option in "Trajectory".','Error','error');
    end

end
```

Fonte: Autoria Própria, 2021

Dando sequência à análise do código, um novo trecho de métodos é descrito. Nesse caso, as funções inicializadas são chamadas de *Callbacks*, que contêm os códigos de reação aos eventos ocorridos na GUI e estão listadas a seguir.

1. Função *startupFcn*

A primeira função, Figura 32, é chamada assim que o aplicativo inicia, definindo as condições iniciais das tabelas, matrizes, figuras e da cor da lâmpada. Além disso, é nesse trecho que o MATLAB faz a conexão com o drone e, caso não seja possível, chama a função de erro *connection_mb*.

Figura 32 – Função *startupFcn*

```
% Callbacks that handle component events
methods (Access = private)

    % Code that executes after component creation
    function startupFcn(app)
        app.Lamp.Color = [1.00 1.00 0.07];

        % Create a Ryze object - connection with the first drone identified through Wi-Fi
        % Message box appears if drone has not been connected
        try
            app.r = ryze();
        catch
            connection_mb(app);
            app.aux_control_disabled = 1;
        end

        % Matrix data
        app.speedMatrix = zeros(100,4);
        app.displacementMatrix = zeros(100,4);

        % Automatic - Table
        app.UITable2.ColumnName = {'X', 'Y', 'Z'};

        % Figure Drone trajectory
        app.Axes = animatedline(app.UIAxes, 'Color', 'r', 'Marker','>');
        app.UIAxes.XDir = 'reverse';
        app.UIAxes.ZLim = [0 3];
        app.UIAxes.Box = 1;
        app.UIAxes.View = [-37.5 30];   % 3D view

        % Figure Expected trajectory
        app.UIAxesExpected.View = [-37.5 30];   % 3D view
    end
```

Fonte: Autoria Própria, 2021

2. Função *UIFigureCloseRequest*

A função *UIFigureCloseRequest*, Figura 33, é executada quando o aplicativo é fechado. Ela tem o intuito de limpar os objetos criados para o drone Ryze e para a câmera, além de deletar o próprio aplicativo.

Figura 33 – Função *UIFigureCloseRequest*

```
% Close request function: UIFigure
function UIFigureCloseRequest(app, event)
    clear app.cameraObj;
    clear app.r;
    delete(app)
end
```

Fonte: Autoria Própria, 2021

3. Função *TakeOffButtonPushed*

A partir do momento que o drone está conectado e o usuário pressiona o botão *Take Off*, o evento associado é ativado na função *TakeOffButtonPushed*, cujo código consta na Figura 34. Antes de fazer o Tello decolar e de definir o intervalo de aquisição de dados, ela verifica se a conexão foi realizada e notifica, chamando a função *connection_mb*, caso contrário.

Figura 34 – Função *TakeOffButtonPushed*

```
% Button pushed function: TakeOffButton
function TakeOffButtonPushed(app, event)
    % Message box appears if drone is not connected
    if(app.aux_control_disabled == 1)
        connection_mb(app);

    % Drone connected
    else
        app.Lamp.Color = 'g';
        takeoff(app.r);

        % Period for data acquisition
        if(app.PeriodfordataacquisitionEditField.Value == 0)
            app.pauseDataAndPlot = 1;
        else
            app.pauseDataAndPlot = app.PeriodfordataacquisitionEditField.Value;
        end

        % Activate while of the DataAndPlotFnc
        % and message box auxiliary variable
        app.DroneInTheAir = 1;

        DataAndPlotFnc(app);
    end
end
```

Fonte: Autoria Própria, 2021

4. Função *LandButtonPushed*

O botão de pouso, *Land*, aciona o evento que verifica se o drone está voando e, em caso afirmativo, inicia o processo de aterrissagem, como pode ser visto na Figura 35. Além disso, os dados obtidos são enviados para o *Workspace* e as informações contidas no objeto do drone são apagadas.

Figura 35 – Função *LandButtonPushed*

```
% Button pushed function: LandButton
function LandButtonPushed(app, event)
    % Message box appears if drone did not take off
    if(app.DroneInTheAir == 0)
        takeoff_mb(app);

    % Drone took off
    else
        app.Lamp.Color = 'r';

        app.row = 1;

        % Send data to workspace
        assignin("base",'speed_data',app.speedMatrix);
        assignin("base",'displacement_data',app.displacementMatrix);

        app.DroneInTheAir = 0;
        land(app.r);
        clear app.r;
    end
end
```

Fonte: Autoria Própria, 2021

5. Funções dos botões de deslocamento

As funções *UpButtonPushed*, *DownButtonPushed*, *LeftButtonPushed*, *RightButtonPushed*, *ForwardButtonPushed* e *BackwardButtonPushed* possuem a mesma estrutura, apresentada como exemplo na Figura 36. O comando associado ao botão só é enviado ao drone quando ele encontra-se no ar.

Figura 36 – Função *UpButtonPushed*

```
% Button pushed function: UpButton
function UpButtonPushed(app, event)
    % Message box appears if drone did not take off
    if(app.DroneInTheAir == 0)
        takeoff_mb(app);

    % Drone took off
    else
        moveup(app.r, 'Distance', app.distance, 'Speed', app.speed, 'WaitUntilDone', false);
    end
end
```

Fonte: Autoria Própria, 2021

De forma análoga ao descrito acima, os botões de rotação (*RightRotationButtonPushed* e *LeftRotationButtonPushed*) apresentam a mesma estrutura, alterando apenas o comando, como consta na Figura 37.

Figura 37 – Função *RightRotationButtonPushed*

```
% Button pushed function: RightRotationButton
function RightRotationButtonPushed(app, event)
    % Message box appears if drone did not take off
    if(app.DroneInTheAir == 0)
        takeoff_mb(app);

    % Drone took off
    else
        turn(app.r,app.angleRight);
    end
end
```

Fonte: Autoria Própria, 2021

6. Função *EnterButtonPushed*

Ao pressionar o botão *Enter* no painel *Displacement on each axis*, o sistema escreve os valores de coordenadas nas variáveis e inicia a parte de verificação, como apresenta o código da Figura 38.

Para os casos em que o drone não tenha decolado ou a entrada não é um valor aceitável, as mensagens de textos são acionadas, respectivamente *connection_mb* e *errorvalue_mb*. Considerando as condições corretas, o comando de delocamento é enviado ao Tello.

Figura 38 – Função *EnterButtonPushed*

```
% Button pushed function: EnterButton
function EnterButtonPushed(app, event)
    X = app.XEditField.Value;
    Y = app.YEditField.Value;
    Z = app.ZEditField.Value;
    angle = deg2rad(app.RotationEditField.Value);

    % Message box appears if drone did not take off
    if(app.DroneInTheAir == 0)
        takeoff_mb(app);
    % Message box appears if value is not accepted
    elseif( (X <= 0.2) && (X >= -0.2) && ...
            (Y <= 0.2) && (Y >= -0.2) && ...
            (Z <= 0.2) && (Z >= -0.2) )
        errorvalue_mb(app);
    % Drone took off and input is accepted
    else
        move(app.r,[X Y Z], 'Speed', app.speed, 'WaitUntilDone', false);
        turn(app.r,angle);
    end
end
```

Fonte: Autoria Própria, 2021

7. Funções *UploadButtonPushed* e *RunButtonPushed*

No caso do controle automático da aeronave, o código é dividido em duas etapas. A primeira consiste na função *UploadButtonPushed* responsável por obter as matrizes do *Workspace*, verificar se os dados são aceitáveis nas especificações da Tabela 2 e, por fim, apresentar a trajetória na GUI.

A segunda etapa trata-se da função *RunButtonPushed*, que verifica se o drone está em voo e verifica se a trajetória foi escolhida na etapa 1. Caso afirmativo, ela chama a função *WorkspaceCoordinatesFnc* e, caso negativo, aciona a caixa de mesnagem.

As funções podem ser analisadas nas Figuras 39 e 40.

Figura 39 – Função *UploadButtonPushed*

```
% Button pushed function: UploadButton
function UploadButtonPushed(app, event)
    app.aux_upload_enabled = 1;

    % Clear table
    app.UITable2.Data = [];

    % Receive Drop Down selected data from workspace
    workspaceMatrix_received = evalin('base',app.DropDown.Value);
    expected_traj_matrixPlot = evalin('base',strcat('matrixPlot_',app.DropDown.Value));

    size_workspaceMatrix_received = size(workspaceMatrix_received);

    % Initialization of auxiliary variables
    auxWrite = 0;
    writeOK = 'false';

    % Verify and create Matrix with at least one of the coordinates is >= 0.2 or <= -0.2
    for auxLimits = 1:1:(size_workspaceMatrix_received(1))
        if(workspaceMatrix_received(auxLimits,1) >= 0.2 || workspaceMatrix_received(auxLimits,2) >= 0.2 || ...
           workspaceMatrix_received(auxLimits,1) <= -0.2 || workspaceMatrix_received(auxLimits,2) <= -0.2 || ...
           workspaceMatrix_received(auxLimits,3) <= -0.2)
            writeOK = true;
            auxWrite = auxWrite + 1;
        else
            writeOK = false;
        end

        if writeOK == true
            app.workspaceMatrix(auxWrite, :) = workspaceMatrix_received(auxLimits,:);
            app.sizeWorkspaceMatrix = auxWrite;
        end
    end

    % Plot workspaceMatrix in UITable2
    app.UITable2.Data = expected_traj_matrixPlot;

    % Plot Expected trajectory in UIAxes
    plot3(app.UIAxesExpected, expected_traj_matrixPlot(:,1), ...
        expected_traj_matrixPlot(:,2), expected_traj_matrixPlot(:,3), 'Marker','>');
end
```

Fonte: Autoria Própria, 2021

8. Funções *StartStopButton_2ValueChanged*

A visualização das imagens da câmera do drone é ativada a partir do momento em que o botão *Start\Stop* é pressionado e o evento *StartStopButton_2ValueChanged* é

Figura 40 – Função *RunButtonPushed*

```
% Button pushed function: RunButton
function RunButtonPushed(app, event)
    % Message box appears if drone did not take off
    if(app.DroneInTheAir == 0)
        takeoff_mb(app);
    % Message box appears if trajectory was not uploaded
    elseif(app.aux_upload_enabled == 0)
        upload_mb(app);
    % Drone took off and input is accepted
    else
        if(app.DisplacementintervalEditField.Value == 0)
            app.displacementInterval = 1;
        else
            app.displacementInterval = app.DisplacementintervalEditField.Value;
        end

        WorkspaceCoordinatesFnc(app);
    end
end
```

Fonte: Autoria Própria, 2021

chamado. O código da função apresenta-se na Figura 41

Assim, se o Tello não estiver conectado, aparecerá a caixa de mensagem gerada pela função *connection_mb*. Caso contrário, a conexão com a câmera é efetuada e o vídeo é apresentado. Por fim, a conexão é encerrada ao pressionar novamente o botão.

Figura 41 – Função *StartStopButton_2ValueChanged*

```
% Value changed function: StartStopButton_2
function StartStopButton_2ValueChanged(app, event)
    % Message box appears if drone is not connected
    if(app.aux_control_disabled == 1)
        connection_mb(app);

    % Drone connected
    else
        value = app.StartStopButton_2.Value;

        if(value)
            % Create a camera object - connection to Ryze drone's camera
            app.cameraObj = camera(app.r);

            % Plot video
            frame = snapshot(app.cameraObj);
            im = image(app.UIAxesVideo, zeros(size(frame),'uint8'));
            axis(app.UIAxesVideo,'image');

            preview(app.cameraObj,im);
        else
            closePreview(app.cameraObj);
            clear app.cameraObj;
        end
    end
end
```

Fonte: Autoria Própria, 2021

9. Funções *HelpButtonPushed*

No caso em que o botão da interrogação é pressionado, o sistema chama outro aplicativo (*UserGuideapp*) para apresentar a nova janela, como pode ser observado na Figura 42.

Figura 42 – Função *HelpButtonPushed*

```
% Button pushed function: HelpButton
function HelpButtonPushed(app, event)
    % User Guide app - Appears when question mark icon is pressed
    app.UserGuideapp = UserGuideapp(app); %#ok<ADPROPLC>
end
```

Fonte: Autoria Própria, 2021

O código do *UserGuideapp* consta na Figura 43 e sua interface foi apresentada na Figura 27. No caso, ele é inicializado ao ser chamado pelo aplicativo principal (função *startupFcn*) e, ao ser fechado, deleta o próprio aplicativo (função *UIFigureCloseRequest*).

Figura 43 – *Code View* do *UserGuideapp*

```
properties (Access = private)
    callingapp;
end


% Callbacks that handle component events
methods (Access = private)

    % Code that executes after component creation
    function startupFcn(app, app8)
        % UserGuideapp is called when question mark icon is clicked on
        % app8
        app.callingapp = app8;
    end

    % Close request function: UIFigure
    function UIFigureCloseRequest(app, event)
        delete(app)
    end
end
```

Fonte: Autoria Própria, 2021

4.3 Considerações sobre o Sistema de Navegação

A Tabela 11 apresenta alguns dos fatores que foram implementados no sistema no tocante às 8 regras de ouro do design de interface, seção 2.5.1.

Tabela 11 – Exemplos da aplicação das 8 regras no sistema de navegação

Regras	**Aplicação na Interface**
Manter consistência	Apresentado nos *layouts*, fontes e cores
Atender às capacidades universais	Presença de Guia de Instruções de uso
Oferecer feedback informativo	Os botões indicam quando são apertados
Projetar diálogos que indiquem o final de uma ação	As caixas de mensagens e a lâmpada oferecem o feedback
Prevenir erros	Nas mensagens indicativas do tipo de dado e nas caixas de mensagens
Permitir a reversão de ações	Valores incorretos não são mandados ao drone e podem ser corrigidos pelo usuário
Oferecer a sensação de controle	O usuário é quem comanda a aeronave e por meio do controle que desejar
Reduzir a carga de memória de curto prazo	Os movimentos que o usuário executa são poucos e concentrados nas áreas de controle

Fonte: Autoria Própria, 2021

Ademais, tomando como base os requisitos definidos na Tabela 7, apresenta-se na Tabela 12 as característica do sistema de navegação final. Observa-se que o sistema não alcançou o requisito de apresentar o deslocamento em tempo real em um gráfico para o controle automático.

Os testes realizados permitiram inferir três possíveis razões para tal situação. A primeira é a ocupação do canal de comunicação por meio dos comandos de deslocamento. Isso é observado tendo em vista que, ao receber um comando desse tipo, o drone emite uma resposta por meio da porta que também é reponsável por emitir o dado da leitura.

A segunda razão é a prioridade que o sistema interno do drone atribui para cada tipo de comando. Considerando que a prioridade do comando de deslocamento é maior que a prioridade do comando de leitura, o sucessivo envio de deslocamentos não permitem a execução das leituras.

A partir do fato de que leitura e resposta do drone são rápidas, como apresentado na Tabela 8 do Ensaio 1, considera-se a terceira razão como a mais plausível. No caso, o MATLAB não consegue processar duas funções *while* ao mesmo tempo, como as pre-

sentes nas funções de envio do comando de deslocamento e de aquisição de dados. Uma situação similar de não processamento aconteceu durante os testes utilizando, no envio de deslocamentos, uma função *while* e, no envio de dados, uma função acionada por um temporizador.

Tabela 12 – Análise dos requisitos do sistema de navegação.

Requisitos específicos	**Conclusão**
Botões para decolar e pousar o drone	Sim
Controlar o deslocamento por meio de botões	Sim
Controlar o deslocamento por meio de valores inseridos para cada eixo	Sim
Fazer com que o drone siga uma trajetória predefinida	Sim
Apresentar o deslocamento em tempo real em um gráfico para o controle manual	Sim
Apresentar o deslocamento em tempo real em um gráfico para o controle automático	Não
Apresentar as imagens capturadas pelo drone em tempo real	Sim
Apresentar um guia de instruções de uso	Sim
Apresentar aviso em situações, entradas e comandos não aceitos	Sim

Fonte: Autoria Própria, 2021

5 Conclusão

Neste trabalho de conclusão de curso obteve-se como resultado final o sistema de navegação do drone Tello proposto. A partir dele é possível controlar a aeronave, de forma manual ou automática, e observar a trajetória percorrida em tempo real por meio do gráfico e do acesso à câmera do Tello.

O projeto pontua as limitações do sistema, como o fato de que o controle automático não possui o acompanhamento em tempo real, e apresenta as especificações de operação do Tello. Além disso, o documento contém a descrição das etapas desenvolvidas, apresentando os problemas, as soluções e os resultados encontrados.

Por fim, o presente trabalho desenvolve e instrui acerca das ferramentas utilizadas, em especial o *App Designer*, bem como apresenta as etapas de construção de uma Interface Gráfica do Usuário, tomando como referência conceitos do Design.

Este trabalho de conclusão de curso demandou e aprofundou diversos conhecimentos adquiridos durante a graduação. Ademais, proporcionou o desenvolvimento das habilidades e conhecimentos acerca do design e do desenvolvimento de software. Assim, espera-se que o projeto motive novas implementações, principalmente considerando a abrangência das áreas em que os RAPs podem ser aplicados.

5.1 Trabalhos Futuros

Como proposta de melhorias para pesquisas similares, tem-se:

- a solvência da limitação supracitada;
- a implementação de novas formas de inserção das matrizes de deslocamentos;
- a habilitação do sistema em diferentes línguas;
- a adição do controle do drone por meio do processamento de imagens.

Referências

ALENCAR, A. et al. *Uso de aeronaves não tripuladas (DRONES) para pesquisa e monitoramento de peixe-boi-marinho e seu habitat. Brasília: ICMBio*. 2020. 45 p. Acessado em: 17 de agosto de 2021. Disponível em: <https://www.icmbio.gov.br/cma/images/stories/Publica%C3%A7%C3%B5es/protocolo_uso_de_aeronaves_nao_tripuladas_drones_para_pesquisa_e_monitoramento_de_peixe_boi_marinho_e_seu_habitat_1.pdf>. Citado na página 14.

ANAC. *RBAC-E n°94, Emenda n°1. REQUISITOS GERAIS PARA AERONAVES NÃO TRIPULADAS DE USO CIVIL*. 2021. Acessado em: 23 de setembro de 2021. Disponível em: <https://www.anac.gov.br/assuntos/legislacao/legislacao-1/rbha-e-rbac/rbac/rbac-e-94-emd-01/@@display-file/arquivo_norma/RBACE94EMD01.pdf>. Citado na página 14.

BRESCIANI, T. Modelling, identification and control of a quadrotor helicopter. *MSc theses*, 2008. Citado 2 vezes nas páginas 14 e 17.

FERREIRA, F. P. *Controle de estabilidade de um quadricóptero*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2015. Citado na página 17.

HALLIDAY et al. *Fundamentos de física 1: mecânica*. [S.l.]: Livros Técnicos e Científicos Rio de Janeiro, 1996. Citado na página 16.

KARDASZ, P. et al. Drones and possibilities of their using. *Journal of Civil & Environmental Engineering*, v. 6, n. 3, p. 1–7, 2016. Citado na página 14.

KEANE et al. A brief history of early unmanned aircraft. *Johns Hopkins APL Technical Digest*, v. 32, n. 3, p. 558–571, 2013. Citado na página 14.

KUROSE, F.; ROSS, K. Redes de computadores e a internet. ed. *São Paulo:[sn]*, v. 656, 2014. Citado na página 24.

MATHWORKS. *Exemplo Read and Plot Navigation Data using MATLAB no site MathWorks*. 1994–2021. Acessado em: 5 de julho de 2021. Disponível em: <https://www.mathworks.com/help/supportpkg/ryzeio/ug/read-plot-navigation-data-using-matlab-support-package-for-ryze-tello-drones.html>. Citado 2 vezes nas páginas 35 e 89.

MATHWORKS. *Página MATLAB Support Package for Ryze Tello Drones no site MathWorks*. 1994–2021. Acessado em: 1 de julho de 2021. Disponível em: <https://www.mathworks.com/help/supportpkg/ryzeio/>. Citado 2 vezes nas páginas 26 e 28.

MATHWORKS. *Página principal do MATLAB no site MathWorks*. 1994–2021. Acessado em: 29 de setembro de 2021. Disponível em: <https://www.mathworks.com/products/matlab.html>. Citado 2 vezes nas páginas 25 e 26.

MATHWORKS. *Página Develop Apps Using App Designer no site MathWorks*. 1994–2021. Acessado em: 1 de julho de 2021. Disponível em: <https://www.mathworks.com/help/matlab/app-designer.html>. Citado na página 26.

MATHWORKS. *Página do comando move do MATLAB Support Package for Ryze Tello Drones*. 2021. Acessado em: 20 de agosto de 2021. Disponível em: <https://www.mathworks.com/help/supportpkg/ryzeio/ref/move.html>. Citado 2 vezes nas páginas 18 e 23.

RYZE. *TELLO SDK 2.0 User Guide v1.0*. 2018. Acessado em: 20 de junhode 2021. Disponível em: <https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf>. Citado 3 vezes nas páginas 9, 22 e 23.

RYZE. *TELLO User Manual v1.4*. 2018. Acessado em: 20 de junho de 2021. Disponível em: <https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20User%20Manual%20v1.4.pdf>. Citado 3 vezes nas páginas 19, 20 e 21.

RYZE. *Página TELLO SPECS do site da empresa Ryze Technology*. 2020. Acessado em: 17 de agosto de 2021. Disponível em: <https://www.ryzerobotics.com/tello/specs>. Citado na página 20.

SHNEIDERMAN, S. B.; PLAISANT, C. Designing the user interface 4 th edition. *ed: Pearson Addison Wesley, USA*, 2005. Citado na página 30.

TUKIMAT, M. B. *DJI Tello - Getting Started with Software Development*. 2020. Acessado em: 18 de junho de 2021. Disponível em: <https://youtu.be/uaXbfDFAp-0>. Citado na página 24.

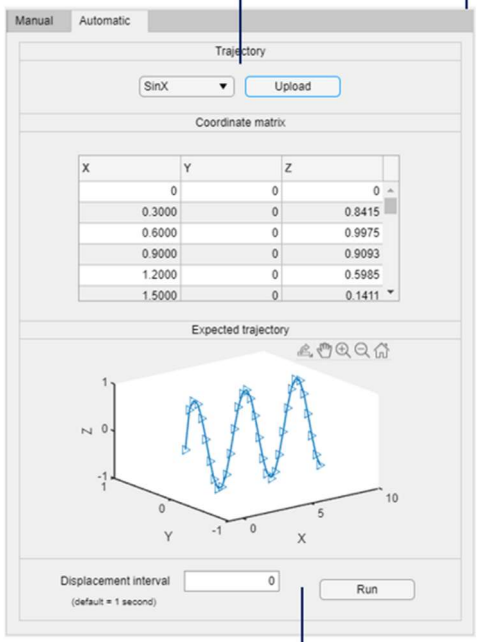
APÊNDICE A – Guia de Instruções do Usuário

Na próxima página é apresentado o Guia de Instruções do Usuário. O guia foi desenvolvido em inglês.

User Guide

Navigation System for Tello Drone

Select the predefined trajectory and click **Upload**. The trajectory will appear in the table and graph.

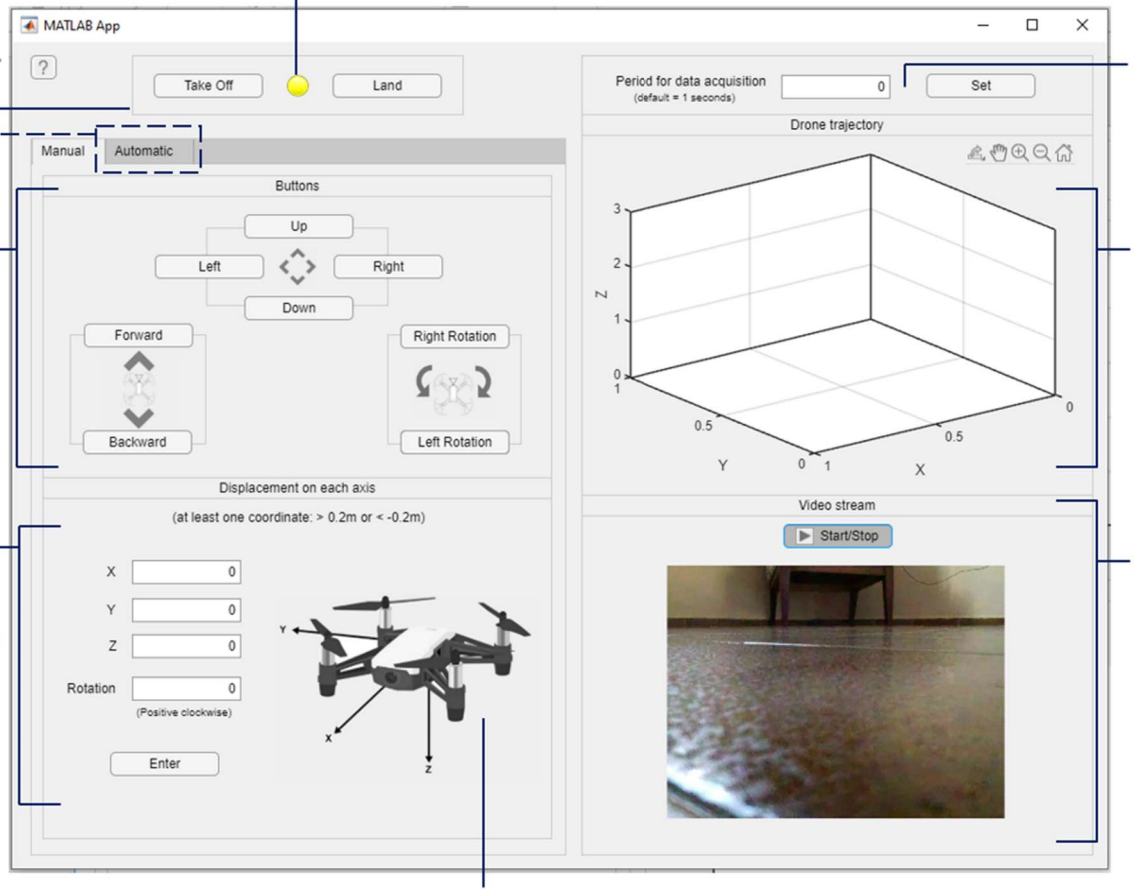


Option to enter the interval between one drone displacement and another.
By clicking on **Run**, the drone will automatically follow the selected trajectory.

Click on the **Automatic** tab.

To take off, the drone must be connected via Wi-Fi.

Yellow light: the drone has not yet taken off.
Green light: the drone has taken off and is flying.
Red light: the drone has landed.



Buttons that command the drone. The drone must be flying.

Option to enter the desired offset on each axis. Comments:
1. The drone must be flying.
2. Value can be negative or positive;
3. **At least one** of the values (X, Y or Z) must be **greater** than **0.2m** or **less** than **-0.2m**;
4. Press **Enter** to send the command.

Image that indicates the coordinate axes of the drone. For example:
If "[X Y Z] = [1 0 0]", the drone shifts **forward** by 1 meter.
If "[X Y Z] = [-1 0 0]", the drone shifts **backward** by 1 meter.

Option to enter the interval between reading one data and another. Press **Set** to set.

Real-time drone trajectory graph.

Press Start/Stop button: Start video transmission from drone camera.
Press the Start/Stop button **again**: turn off transmission.

APÊNDICE B – Guia do *App Designer*

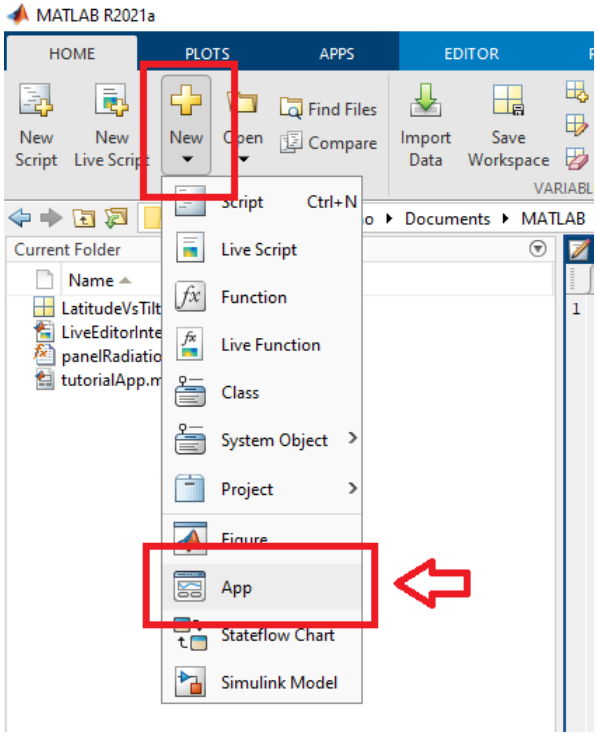
A seguir serão apresentadas as etapas necessárias para abertura do *App Designer*, bem como os componentes da interface e como desenvolver um programa.

B.1 Abrindo a tela do *App Designer*

Para acessar o *App Designer* é preciso ter o software MATLAB instalado no computador. Assim, na tela principal do software, existem duas formas de abrir a tela do *App Designer*:

1. selecionar, na parte superior direita do MATLAB, a opção *New* seguido da opção *App*, como apresenta a Figura 44.

Figura 44 – Opção de abertura do *App Designer*



Fonte: MATLAB *App Designer*

1. inserir na tela de comando, *Command Window*, a palavra "*appdesigner*" e pressionar a tecla *Enter*, como consta na Figura 45.

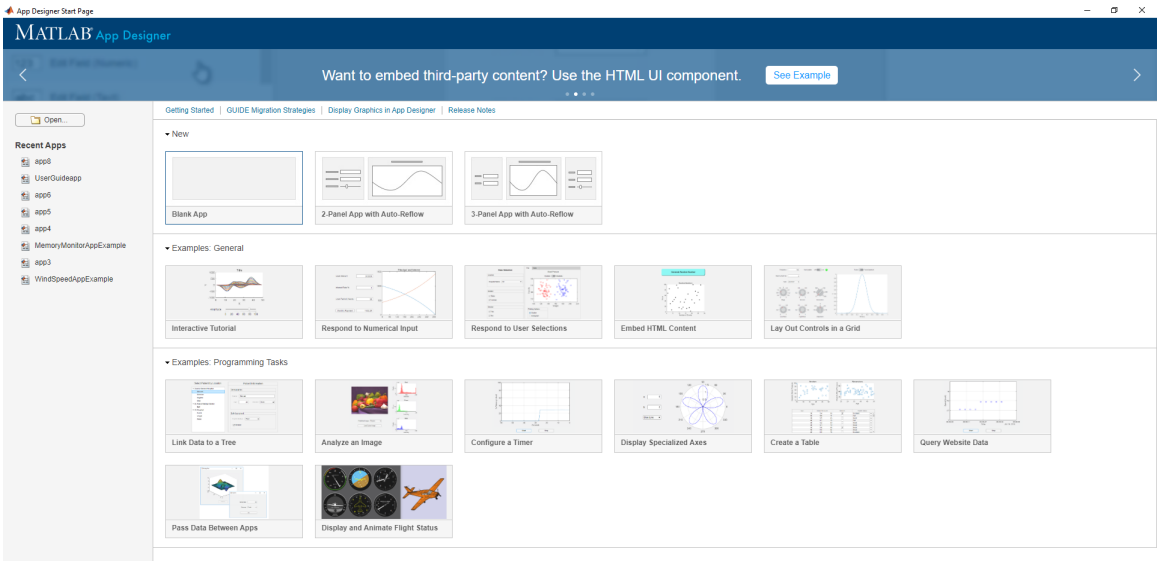
Figura 45 – Opção de abertura do *App Designer*

Fonte: MATLAB *App Designer*

B.2 Componentes do *App Designer*

Após a abertura do *App Designer*, a tela apresentada ao usuário consta na Figura 46. Nela é possível acessar os projetos recentes, o guia de como usar a plataforma (*Getting Started*), os exemplos de projetos e novos projetos em banco.

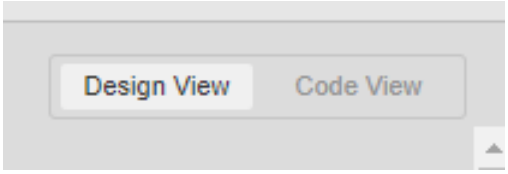
Figura 46 – Opções de projetos



Fonte: MATLAB *App Designer*

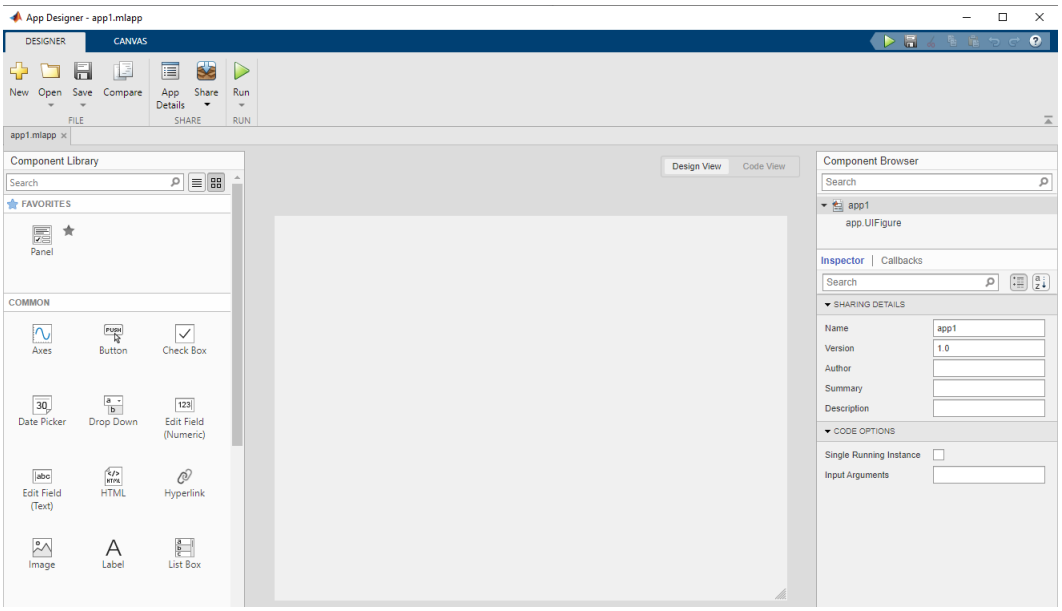
Ao selecionar um projeto em branco, a tela da Figura 48 aparece. Como apresentado na seção 2.4.1, existe um ambiente integrado que permite com que o usuário construa a Interface Gráfica do Usuário (GUI, *Graphical User Interface*), no modo *Design View*, e escreva o código referente ao comportamento do aplicativo, modo *Code View*. O modo é selecionado na parte superior direita da tela, na região indicada na Figura 47.

Figura 47 – Região para alternar o modo entre *Design View* e *Code View*



Fonte: MATLAB *App Designer*

Figura 48 – Página inicial do *App Designer*, modo *Design View*

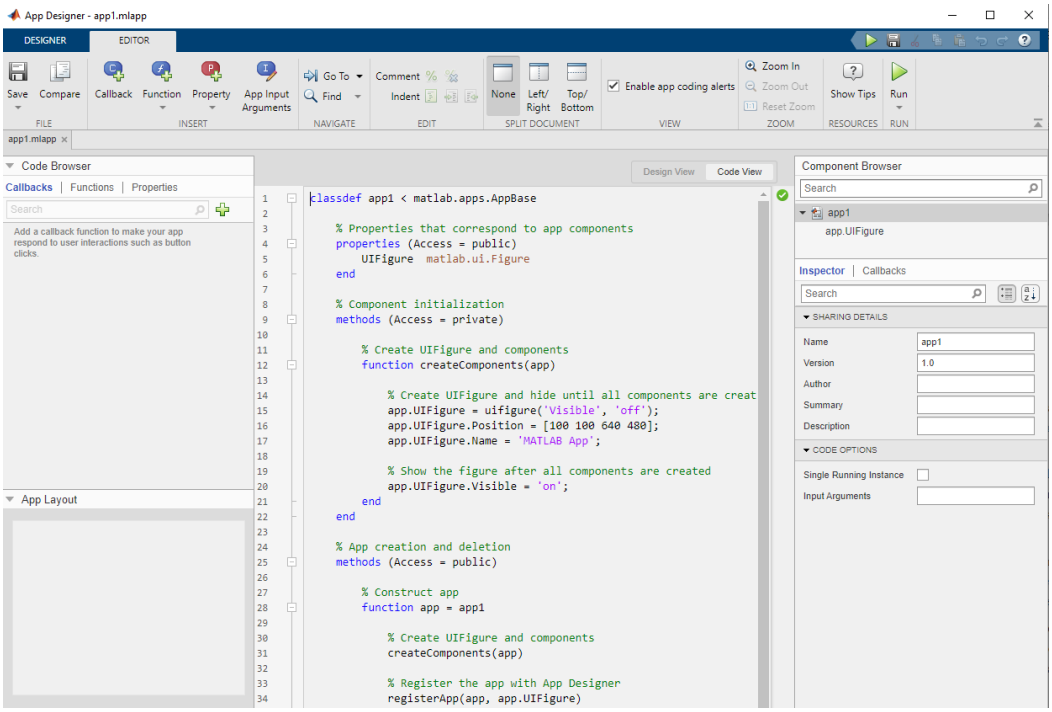


Fonte: MATLAB *App Designer*

A página inicial apresenta o modo *Design View*. No lado esquerdo, *Component Library*, são apresentados os componentes da GUI. Para colocar um componente GUI, basta selecionar o escolhido e arrastá-lo para o centro do retângulo cinza claro. Assim, no lado direito, *Component Browser*, aparecerão as características do item (como *labels*, *font* e *position*) e a lista de elementos da interface em construção.

Ao selecionar o modo *Code View*, a tela alterna para a que é apresentada na Figura 49. No lado inferior esquerdo, *App Layout*, é possível visualizar a interface desenvolvida no *Design View*. Logo acima, no *Code Browser*, são apresentadas listas com todos os *Callbacks*, funções e propriedades presentes no código.

Figura 49 – Página inicial do *App Designer*, modo *Code View*



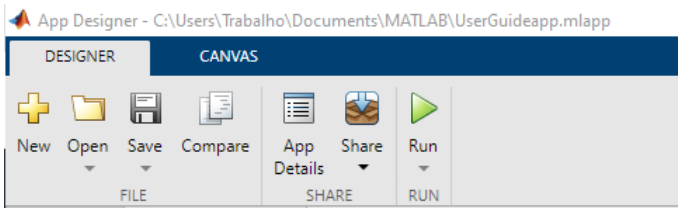
Fonte: MATLAB *App Designer*

O lado direito apresenta o mesmo *Component Browser* do *Design View*. Por fim, a parte central contém:

- códigos predefinidos do sistema, com configurações padrões dos aplicativos, e que não podem ser alterados (fundo fica cinza);
- códigos gerados automaticamente quando são inseridos componentes na GUI e que não podem ser alterados (fundo fica cinza);
- códigos contendo *Callbacks*, funções e propriedades desenvolvidos pelo programador (fundo fica branco).

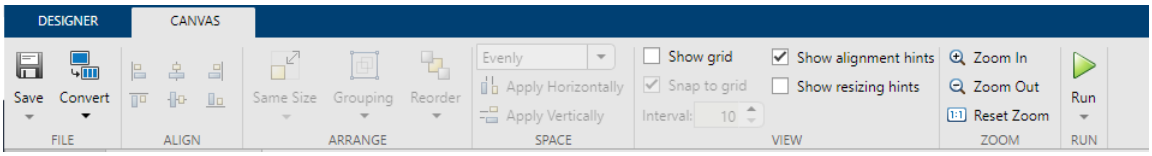
Por fim, na parte superior, existem três possíveis abas de configurações e ferramentas. A primeira, presente tanto no modo *Design View* quanto no *Code View*, é a *DESIGNER*, apresentada na Figura 50. A segunda, que aparece apenas no modo *Design View* e está ilustrada na Figura 51, é a *CANVAS*. Por fim, a terceira aba é a *EDITOR*, presente apenas no modo *Code View*, como mostra a Figura 52.

Figura 50 – Aba *DESIGNER*



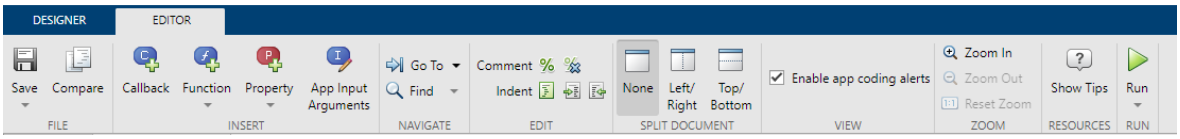
Fonte: MATLAB *App Designer*

Figura 51 – Aba *CANVAS*



Fonte: MATLAB *App Designer*

Figura 52 – Aba *Editor*



Fonte: MATLAB *App Designer*

B.3 Passo a passo de programa simples no *App Designer*

Após a descrição de como abrir o *App Designer* e das suas funcionalidades, desenvolve-se o primeiro aplicativo. Cada passo é listado a seguir.

O objetivo é, ao apertar o botão de soma, somar dois valores inseridos pelo usuário, apresentar o valor resultante em uma caixa de texto, em uma tabela e em um medidor (todos presentes em um *Panel*).

Dica: Ao realizar o passo a passo, observar as alterações que ocorrem no *Component Browser*.

1°. Arrastar e soltar os componentes listados a seguir. Atentar para colocar a caixa de texto, a tabela e o medidor sobre o *Panel*. (vide Figura 53)

   a) 3 *Edit Field*;

   b) 1 *Push Button*;

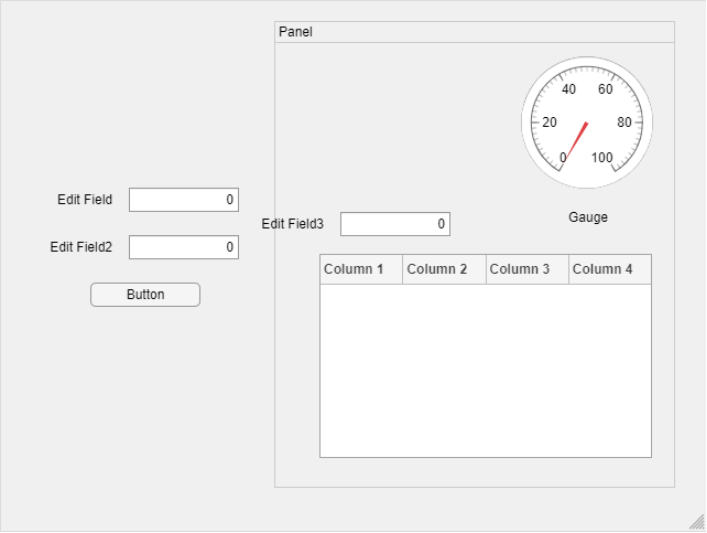
   c) 1 *Table*;

   d) 1 *Gauge*;

   e) 1 *Panel*.

Observação: Caso o *Panel* fique por cima dos demais componentes, basta clicar nele com o botão direito do mouse, clicar em *Reorder* e, posteriormente, *Send to Back*. O tamanho dos componentes também podem ser alterados, como acontecerá no segundo passo.

Figura 53 – Figura referente ao primeiro passo



Fonte: MATLAB *App Designer*

2°. Para renomear os componentes existem duas opções: clicar duas vezes no nome associado ao componente ou clicar no componente e alterar o nome em *Text* no *Componen Browser*. Assim, renomeia-se os componentes respectivamente como: (vide Figura 54)

a) "Número 1" para o primeiro *Edit Field*;

b) "Número 2" para o segundo *Edit Field*;

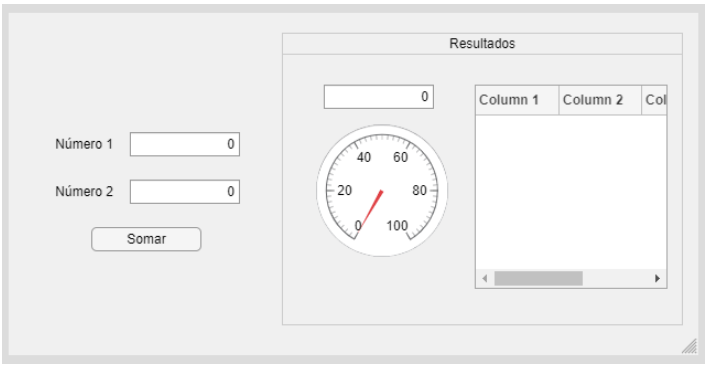
c) apagar o nome do terceiro *Edit Field*;

d) "Somar" para o *Push Button*;

e) apagar o nome do *Gauge*;

f) "Resultados" para o *Panel*.

Figura 54 – Figura referente ao segundo passo



Fonte: MATLAB *App Designer*

3°. A soma será realizada a partir do momento em que o botão "Somar" for apertado. Assim, o evento (*Callback*) a ser criado é referente a ele. Para tal, deve-se: (o resultado esperado apresenta-se na Figura 55)

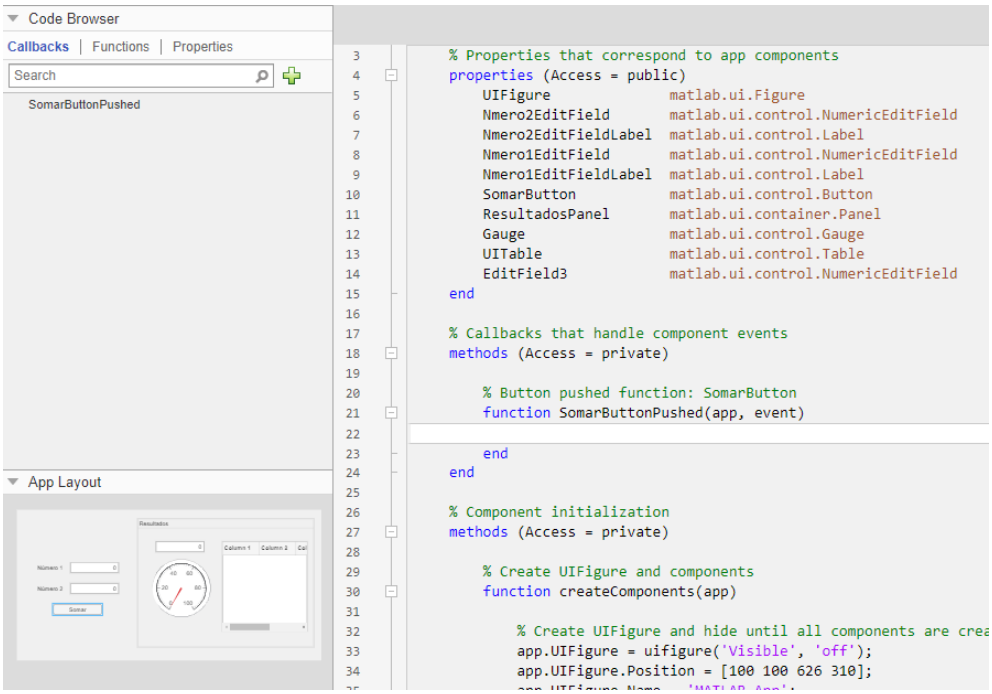
a) clicar no botão "Somar" com o botão direito do mouse;

b) clicar em *Callbacks*;

c) e clicar no *Add ButtonPushedFcn callback*.

Observação: Outra forma de criar um *Callback* (ou uma função ou uma propriedade) é clicando no símbolo de "+" (mais) no *Code Browser*.

Figura 55 – Figura referente ao terceiro passo



Fonte: MATLAB *App Designer*

4°. O código a ser desenvolvido consta na Figura 56. Observar que:

a) o acesso aos valores inseridos e às propriedades dos componentes é realizado por meio da sequência: "app." + nome_do_componente + propriedade desejada (no caso, *Value*);

b) o código apresenta duas formas de **receber** o valor inserido pelo usuário, de forma direta ou atribuindo a uma variável;

c) o código apresenta duas formas de **apresentar** o dado na interface, de forma direta ou por uma variável.

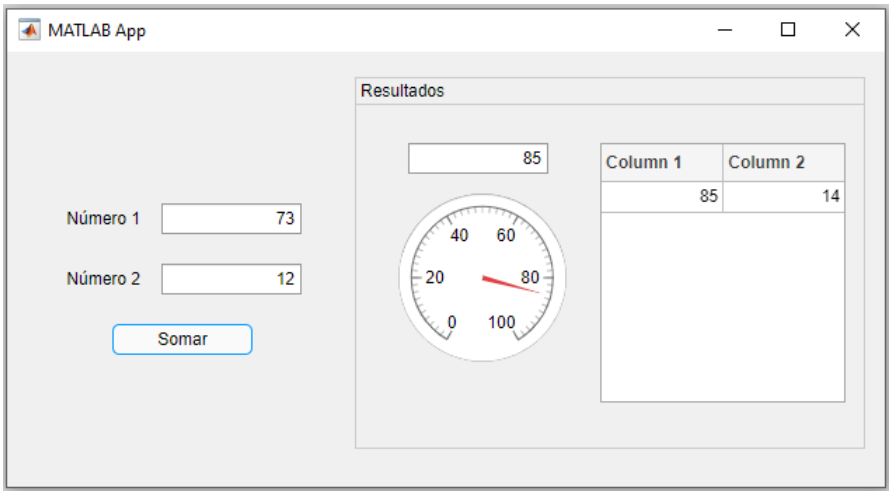
Figura 56 – Figura referente ao quarto passo

```
% Button pushed function: SomarButton
function SomarButtonPushed(app, event)
    % Recebe o dado do usuário em uma variável
    Numero2 = app.Nmero2EditField.Value;

    % Recebe o dado do usuário de forma direta na soma
    % Recebe o resultado da soma em uma variável
    Soma = Numero2 + app.Nmero1EditField.Value;

    % Apresenta o dado no componente EditField3
    app.EditField3.Value = Soma;

    % Realiza a soma de forma direta e
    % apresenta no medidor
    app.Gauge.Value = Numero2 + app.Nmero1EditField.Value;

    % Realiza diferentes cálculos e apresenta na Tabela
    app.UITable.Data = [Soma 2+Numero2];
end
```

Fonte: MATLAB *App Designer*

5°. Por fim, ao clicar em *Run* em qualquer uma das 3 abas, o aplicativo é gerado e o usuário pode inserir os valores e obter as respostas. A Figura 57 apresenta o aplicativo após realizar uma soma.

Figura 57 – Figura referente ao quinto passo



Fonte: MATLAB *App Designer*

APÊNDICE C – Código desenvolvido no *Code View* do *App Designer*

- **Código do aplicativo principal**

O bloco a seguir apresenta o código do sistema de navegação (aplicativo *app8.mlapp*).

```
properties (Access = private)
        r;              % Ryze object - drone
        Axes;           % UIAxes object for Data and Plot
        cameraObj;      % Camera object

        speedMatrix;        % Matrix for storing speed data acquired ...
            from the drone
        displacementMatrix; % Matrix for storing displacement data ...
            calculated
        workspaceMatrix;    % Matrix with coordinates defined in the ...
            workspace

        % Auxiliary variables of message box
        aux_control_disabled = 0; % Indicates the drone is not connected
        aux_upload_enabled = 0;

        % Buttons Panel
        distance = 1;             % Standard distance for displacement
        speed = 0.2;              % Standard speed for displacement
        angleLeft = deg2rad(-45); % Standard angle for rotation
        angleRight = deg2rad(45); % Standard angle for rotation

        % Reading data
        speedR;                   % Drone speed obtained
        time;                     % Datetime when MATLAB receives ...
            speed read
        height;                   % Drone height obtained
        Xspeed;                   % Drone speed X axis
        Yspeed;                   % Drone speed Y axis
        Zspeed;                   % Drone speed Z axis
        ΔX;                   % X axis - Distance traveled between ...
            the two speed readings
        ΔY;                   % Y axis - distance traveled between ...
            the two speed readings
        coordX = 0;               % Drone position X axis
```

```
30          coordY = 0;                 % Drone position Y axis
31          elapsedTime = 0;            % Variable to save the elapsed ...
                time - tic and toc
32          auxFirst = 0;               % Auxiliary variable - first time ...
                DataAcquisitionFnc is called
33          auxTime = 0;                % tic and toc auxiliary variable - ...
                ΔTime
34          DroneInTheAir = 0;          % Variable to enter while and ...
                message box auxiliary variable
35          pauseDataAndPlot;           % Pause value changed in the UI
36
37          % Workspace Coordinates
38          row = 1;
39          sizeWorkspaceMatrix;
40          moveX = 0;                          % X coordinate to which the ...
                drone should move
41          moveY = 0;                          % Y coordinate to which the ...
                drone should move
42          moveZ = 0;                          % Z coordinate to which the ...
                drone should move
43          displacementInterval;               % Displacement interval ...
                changed in the UI
44
45          % User Guide app - Appears when question mark icon is pressed
46          UserGuideapp;
47      end
48
49      methods (Access = private)
50
51          function DataAndPlotFnc(app)
52              while (app.DroneInTheAir)
53                  % Record elapsed time - Used to calculate drone position
54                  if (app.auxFirst ≠ 0)
55                      app.elapsedTime = toc;
56                  end
57                  tic;
58
59                  % Read drone speed and height
60                  [app.speedR,app.time] = readSpeed(app.r);
61                  app.height = readHeight(app.r);
62
63                  app.Xspeed = app.speedR(1);
64                  app.Yspeed = app.speedR(2);
65                  app.Zspeed = app.speedR(3);
66
67                  % Save speed data in speedMatrix
68                  app.speedMatrix = circshift(app.speedMatrix, 1);
```

```
69                  app.speedMatrix(1,:) = [app.Xspeed app.Yspeed ...
                        app.Zspeed app.elapsedTime];
70
71                  % Calculate the distance traveled between the two ...
                        speed readings
72                  app.ΔX = (app.Xspeed)*(app.elapsedTime);
73                  app.ΔY = (app.Yspeed)*(app.elapsedTime);
74
75                  % Calculate the distance traveled between the two ...
                        speed readings
76                  % X coordinate - Filter for non-displacement ...
                        velocity data
77                  if ((app.speedMatrix(1,1) == 0) && ...
                        (app.speedMatrix(3,1) == 0))
78                     app.coordX = app.displacementMatrix(2,1);
79                     app.displacementMatrix(1,1) = ...
                           app.displacementMatrix(2,1);
80                  end
81
82                  % Calculate the distance traveled between the two ...
                        speed readings
83                  % Y coordinate - Filter for non-displacement ...
                        velocity data
84                  if ((app.speedMatrix(1,2) == 0) && ...
                        (app.speedMatrix(3,2) == 0))
85                     app.coordY = app.displacementMatrix(2,2);
86                     app.displacementMatrix(1,2) = ...
                           app.displacementMatrix(2,2);
87                  end
88
89                  % Total Distance - Actual Drone Position
90                  app.coordX = app.ΔX + app.coordX;
91                  app.coordY = app.ΔY + app.coordY;
92
93                  % Save data in dataMatrix
94                  app.displacementMatrix = ...
                        circshift(app.displacementMatrix, 1);
95                  app.displacementMatrix(1,:) = [app.coordX app.coordY ...
                        app.height app.elapsedTime];
96
97                  % Plot UIAxes - Delay in one reading
98                  addpoints(app.Axes, app.displacementMatrix(2,1), ...
                        app.displacementMatrix(2,2), ...
                        app.displacementMatrix(2,3));
99                  drawnow
100
101                 app.auxFirst = 1;
```

```
102                 pause(app.pauseDataAndPlot);
103             end
104
105         end
106
107         function WorkspaceCoordinatesFnc(app)
108             while (app.row ≤ app.sizeWorkspaceMatrix)
109                 % Move drone
110                 app.moveX = app.workspaceMatrix(app.row,1);
111                 app.moveY = app.workspaceMatrix(app.row,2);
112                 app.moveZ = app.workspaceMatrix(app.row,3);
113
114                 move(app.r,[app.moveX app.moveY app.moveZ], 'Speed', ...
                        app.speed, 'WaitUntilDone', false);
115
116                 app.row = app.row + 1;
117                 pause(app.displacementInterval)
118             end
119
120         end
121
122
123         % Message boxes
124         function connection_mb(app)
125             msgbox('Drone is not connected. To connect the drone: ...
                    close the app, connect the drone via Wi-Fi and run ...
                    the app again','Error','error');
126         end
127         function takeoff_mb(app)
128             msgbox('To control the drone: the drone must be ...
                    connected and it must have taken off','Error','error');
129         end
130         function errorvalue_mb(app)
131             msgbox('At least one coordinate must be greater than ...
                    0.2m or less than -0.2m','Error','error');
132         end
133
134         function upload_mb(app)
135             msgbox('Upload required. Select an option in ...
                    "Trajectory".','Error','error');
136         end
137
138     end
139
140     % Callbacks that handle component events
141     methods (Access = private)
142
```

```
143         % Code that executes after component creation
144         function startupFcn(app)
145             app.Lamp.Color = [1.00 1.00 0.07];
146 
147             % Create a Ryze object - connection with the first drone ...
                    identified through Wi-Fi
148             % Message box appears if drone has not been connected
149             try
150                 app.r = ryze();
151             catch
152                 connection_mb(app);
153                 app.aux_control_disabled = 1;
154             end
155 
156             % Matrix data
157             app.speedMatrix = zeros(100,4);
158             app.displacementMatrix = zeros(100,4);
159 
160             % Automatic - Table
161             app.UITable2.ColumnName = {'X', 'Y', 'Z'};
162 
163             % Figure Drone trajectory
164             app.Axes = animatedline(app.UIAxes, 'Color', 'r', ...
                    'Marker','>');
165             app.UIAxes.XDir = 'reverse';
166             app.UIAxes.ZLim = [0 3];
167             app.UIAxes.Box = 1;
168             app.UIAxes.View = [-37.5 30];    % 3D view
169 
170             % Figure Expected trajectory
171             app.UIAxesExpected.View = [-37.5 30];    % 3D view
172         end
173 
174         % Close request function: UIFigure
175         function UIFigureCloseRequest(app, event)
176             clear app.cameraObj;
177             clear app.r;
178             delete(app)
179         end
180 
181         % Button pushed function: TakeOffButton
182         function TakeOffButtonPushed(app, event)
183             % Message box appears if drone is not connected
184             if(app.aux_control_disabled == 1)
185                 connection_mb(app);
186 
187             % Drone connected
```

```
188             else
189                 app.Lamp.Color = 'g';
190                 takeoff(app.r);
191 
192                 % Period for data acquisition
193                 if(app.PeriodfordataacquisitionEditField.Value == 0)
194                     app.pauseDataAndPlot = 1;
195                 else
196                     app.pauseDataAndPlot = ...
                        app.PeriodfordataacquisitionEditField.Value;
197                 end
198 
199                 % Activate while of the DataAndPlotFnc
200                 % and message box auxiliary variable
201                 app.DroneInTheAir = 1;
202 
203                 DataAndPlotFnc(app);
204             end
205         end
206 
207         % Button pushed function: LandButton
208         function LandButtonPushed(app, event)
209             % Message box appears if drone did not take off
210             if(app.DroneInTheAir == 0)
211                 takeoff_mb(app);
212 
213             % Drone took off
214             else
215                 app.Lamp.Color = 'r';
216 
217                 app.row = 1;
218 
219                 % Send data to workspace
220                 assignin("base",'speed_data',app.speedMatrix);
221                 assignin("base",'displacement_data',app.displacementMatrix);
222 
223                 app.DroneInTheAir = 0;
224                 land(app.r);
225                 clear app.r;
226             end
227         end
228 
229         % Button pushed function: UpButton
230         function UpButtonPushed(app, event)
231             % Message box appears if drone did not take off
232             if(app.DroneInTheAir == 0)
233                 takeoff_mb(app);
```

```
234
235             % Drone took off
236             else
237                 moveup(app.r, 'Distance', app.distance, 'Speed', ...
                        app.speed, 'WaitUntilDone', false);
238             end
239         end
240
241         % Button pushed function: DownButton
242         function DownButtonPushed(app, event)
243             % Message box appears if drone did not take off
244             if(app.DroneInTheAir == 0)
245                 takeoff_mb(app);
246
247             % Drone took off
248             else
249                 movedown(app.r, 'Distance', app.distance, 'Speed', ...
                        app.speed, 'WaitUntilDone', false);
250             end
251         end
252
253         % Button pushed function: LeftButton
254         function LeftButtonPushed(app, event)
255             % Message box appears if drone did not take off
256             if(app.DroneInTheAir == 0)
257                 takeoff_mb(app);
258
259             % Drone took off
260             else
261                 moveleft(app.r, 'Distance', app.distance, 'Speed', ...
                        app.speed, 'WaitUntilDone', false);
262             end
263         end
264
265         % Button pushed function: RightButton
266         function RightButtonPushed(app, event)
267             % Message box appears if drone did not take off
268             if(app.DroneInTheAir == 0)
269                 takeoff_mb(app);
270
271             % Drone took off
272             else
273                 moveright(app.r, 'Distance', app.distance, 'Speed', ...
                        app.speed, 'WaitUntilDone', false);
274             end
275         end
276
```

```
277         % Button pushed function: ForwardButton
278         function ForwardButtonPushed(app, event)
279             % Message box appears if drone did not take off
280             if(app.DroneInTheAir == 0)
281                 takeoff_mb(app);
282
283             % Drone took off
284             else
285                 moveforward(app.r, 'Distance', app.distance, ...
                        'Speed', app.speed, 'WaitUntilDone', false);
286             end
287         end
288
289         % Button pushed function: BackwardButton
290         function BackwardButtonPushed(app, event)
291             % Message box appears if drone did not take off
292             if(app.DroneInTheAir == 0)
293                 takeoff_mb(app);
294
295             % Drone took off
296             else
297                 moveback(app.r, 'Distance', app.distance, 'Speed', ...
                        app.speed, 'WaitUntilDone', false);
298             end
299         end
300
301         % Button pushed function: LeftRotationButton
302         function LeftRotationButtonPushed(app, event)
303             % Message box appears if drone did not take off
304             if(app.DroneInTheAir == 0)
305                 takeoff_mb(app);
306
307             % Drone took off
308             else
309                 turn(app.r,app.angleLeft);
310             end
311         end
312
313         % Button pushed function: RightRotationButton
314         function RightRotationButtonPushed(app, event)
315             % Message box appears if drone did not take off
316             if(app.DroneInTheAir == 0)
317                 takeoff_mb(app);
318
319             % Drone took off
320             else
321                 turn(app.r,app.angleRight);
```

```
322             end
323         end
324
325         % Button pushed function: EnterButton
326         function EnterButtonPushed(app, event)
327             X = app.XEditField.Value;
328             Y = app.YEditField.Value;
329             Z = app.ZEditField.Value;
330             angle = deg2rad(app.RotationEditField.Value);
331
332             % Message box appears if drone did not take off
333             if(app.DroneInTheAir == 0)
334                 takeoff_mb(app);
335             % Message box appears if value is not accepted
336             elseif( (X ≤ 0.2) && (X ≥ -0.2) && ...
337                     (Y ≤ 0.2) && (Y ≥ -0.2) && ...
338                     (Z ≤ 0.2) && (Z ≥ -0.2) )
339                 errorvalue_mb(app);
340             % Drone took off and input is accepted
341             else
342                 move(app.r,[X Y Z], 'Speed', app.speed, ...
                        'WaitUntilDone', false);
343                 turn(app.r,angle);
344             end
345         end
346
347         % Button pushed function: UploadButton
348         function UploadButtonPushed(app, event)
349             app.aux_upload_enabled = 1;
350
351             % Clear table
352             app.UITable2.Data = [];
353
354             % Receive Drop Down selected data from workspace
355             workspaceMatrix_received = ...
                    evalin('base',app.DropDown.Value);
356             expected_traj_matrixPlot = ...
                    evalin('base',strcat('matrixPlot_',app.DropDown.Value));
357
358             size_workspaceMatrix_received = ...
                    size(workspaceMatrix_received);
359
360             % Initialization of auxiliary variables
361             auxWrite = 0;
362             writeOK = 'false';
363
364             % Verify and create Matrix with at least one of the ...
```

```
                coordinates is ≥ 0.2 or ≤ -0.2
365             for auxLimits = 1:1:(size_workspaceMatrix_received(1))
366                 if(workspaceMatrix_received(auxLimits,1) ≥ 0.2 || ...
                       workspaceMatrix_received(auxLimits,2) ≥ 0.2 || ...
                       workspaceMatrix_received(auxLimits,3) ≥ 0.2 || ...
367                        workspaceMatrix_received(auxLimits,1) ≤ -0.2 ...
                               || workspaceMatrix_received(auxLimits,2) ≤...
                                -0.2 || ...
                               workspaceMatrix_received(auxLimits,3) ≤ -0.2)
368                     writeOK = true;
369                     auxWrite = auxWrite + 1;
370                 else
371                     writeOK = false;
372                 end
373
374                 if writeOK == true
375                     app.workspaceMatrix(auxWrite, :) = ...
                           workspaceMatrix_received(auxLimits,:);
376                     app.sizeWorkspaceMatrix = auxWrite;
377                 end
378             end
379
380             % Plot workspaceMatrix in UITable2
381             app.UITable2.Data = expected_traj_matrixPlot;
382
383             % Plot Expected trajectory in UIAxes
384             plot3(app.UIAxesExpected, expected_traj_matrixPlot(:,1), ...
                    expected_traj_matrixPlot(:,2), ...
                    expected_traj_matrixPlot(:,3), 'Marker','>');
385         end
386
387         % Button pushed function: RunButton
388         function RunButtonPushed(app, event)
389             % Message box appears if drone did not take off
390             if(app.DroneInTheAir == 0)
391                 takeoff_mb(app);
392             % Message box appears if trajectory was not uploaded
393             elseif(app.aux_upload_enabled == 0)
394                 upload_mb(app);
395             % Drone took off and input is accepted
396             else
397                 if(app.DisplacementintervalEditField.Value == 0)
398                     app.displacementInterval = 1;
399                 else
400                     app.displacementInterval = ...
                            app.DisplacementintervalEditField.Value;
401                 end
```

```
402
403                 WorkspaceCoordinatesFnc(app);
404             end
405         end
406
407         % Button pushed function: HelpButton
408         function HelpButtonPushed(app, event)
409             % User Guide app - Appears when question mark icon is ...
                    pressed
410             app.UserGuideapp = UserGuideapp(app);
411         end
412
413         % Value changed function: StartStopButton_2
414         function StartStopButton_2ValueChanged(app, event)
415             % Message box appears if drone is not connected
416             if(app.aux_control_disabled == 1)
417                 connection_mb(app);
418
419             % Drone connected
420             else
421                 value = app.StartStopButton_2.Value;
422
423                 if(value)
424                     % Create a camera object - connection to Ryze ...
                            drone's camera
425                     app.cameraObj = camera(app.r);
426
427                     % Plot video
428                     frame = snapshot(app.cameraObj);
429                     im = image(app.UIAxesVideo, ...
                            zeros(size(frame),'uint8'));
430                     axis(app.UIAxesVideo,'image');
431
432                     preview(app.cameraObj,im);
433                 else
434                     closePreview(app.cameraObj);
435                     clear app.cameraObj;
436                 end
437             end
438         end
439     end
```

- **Código do aplicativo *UserGuideapp***

```
properties (Access = private)
        callingapp;
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, app8)
            % UserGuideapp is called when question mark icon is ...
               clicked on
            % app8
            app.callingapp = app8;
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            delete(app)
        end
    end
```

APÊNDICE D – Códigos utilizados nos Ensaios

Os códigos aqui presentes foram desenvolvidos no Editor do MATLAB em linguagem MATLAB.

D.1 Ensaio 1

O bloco a seguir apresenta o código utilizado no Ensaio 1.

```
r = ryze()

takeoff(r)

c = clock
fix(c)
[s1,t1] = readSpeed(r)
c = clock
fix(c)
[s2,t2] = readSpeed(r)
c = clock
fix(c)
[s3,t3] = readSpeed(r)
c = clock
fix(c)
[s4,t4] = readSpeed(r)
c = clock
fix(c)
[s5,t5] = readSpeed(r)
c = clock
fix(c)
[s6,t6] = readSpeed(r)
c = clock
fix(c)
[s7,t7] = readSpeed(r)
c = clock
fix(c)
[s8,t8] = readSpeed(r)
c = clock
fix(c)
[s9,t9] = readSpeed(r)
```

```
32 c = clock
33 fix(c)
34 [s10,t10] = readSpeed(r)
35 [s11,t11] = readSpeed(r)
36 [s12,t12] = readSpeed(r)
37 [s13,t13] = readSpeed(r)
38 [s14,t14] = readSpeed(r)
39 [s15,t15] = readSpeed(r)
40 [s16,t16] = readSpeed(r)
41 [s17,t17] = readSpeed(r)
42 [s18,t18] = readSpeed(r)
43 [s19,t19] = readSpeed(r)
44 [s20,t20] = readSpeed(r)
45 [s21,t21] = readSpeed(r)
46 [s22,t22] = readSpeed(r)
47 [s23,t23] = readSpeed(r)
48 [s24,t24] = readSpeed(r)
49 [s25,t25] = readSpeed(r)
50 [s26,t26] = readSpeed(r)
51 [s27,t27] = readSpeed(r)
52 [s28,t28] = readSpeed(r)
53 [s29,t29] = readSpeed(r)
54 [s30,t30] = readSpeed(r)
55 [s31,t31] = readSpeed(r)
56 [s32,t32] = readSpeed(r)
57 [s33,t33] = readSpeed(r)
58 [s34,t34] = readSpeed(r)
59 [s35,t35] = readSpeed(r)
60 [s36,t36] = readSpeed(r)
61 [s37,t37] = readSpeed(r)
62 [s38,t38] = readSpeed(r)
63 [s39,t39] = readSpeed(r)
64
65 land(r)
66
67 clear r
```

D.2 Ensaio 2

O bloco a seguir apresenta o código utilizado no Ensaio 2 e tem como referência o código do exemplo *Read and Plot Navigation Data using* MATLAB (MATHWORKS, 1994-2021a).

```
% ----- Navegacao rotacao - deslocamento eixo x
    diary arquivosaida.txt

   f = figure;
   hx = animatedline('Color', 'r', 'LineWidth', 2);
   hy = animatedline('Color', 'g', 'LineWidth', 2);
   hz = animatedline('Color', 'b', 'LineWidth', 2);
   title('DroneOrientation v/s Time');
   xlabel('Time (in s)');
   ylabel('Orientation (in degrees)');
   legend('XOrientation', 'YOrientation', 'ZOrientation');

% Criar um objeto ryze - conexao com o primeiro drone identificado por
% Wi-Fi
    r = ryze()
    takeoff(r);

  edgeIndex = 1;
  distance = 0.4;
  speed = 0.4;
  tObj = tic;
  while(edgeIndex ≤ 4)
      % Move the drone unblocking the command line
      tplot = tic
      moveforward(r, 'Distance', distance, 'Speed', speed, ...
          'WaitUntilDone', false);
      % Plot orientation while drone is moving
      while(toc(tplot)<distance/speed)
         orientation = rad2deg(readOrientation(r));
         fprintf('Answer: (%d, %d, %d) \n', orientation)
         %fid=fopen('MyFile.txt','wt');
         %fprintf(fid, '%f r\n', orientation');
         %fclose(fid);

         tStamp = toc(tObj)
         addpoints(hx, tStamp, orientation(2));
         addpoints(hy, tStamp, orientation(3));
         addpoints(hz, tStamp, orientation(1));
         drawnow;
     tempo_decorrido = toc(tplot)
```

```
40        end
41       % Turn the drone after it has traversed one side of the square path
42       pause(2);
43       turn(r, deg2rad(90));
44       edgeIndex = edgeIndex+1;
45    end
46
47    orientation = rad2deg(readOrientation(r));
48     tStamp = toc(tObj);
49     addpoints(hx, tStamp, orientation(2));
50     addpoints(hy, tStamp, orientation(3));
51     addpoints(hz, tStamp, orientation(1));
52     drawnow;
53     land(r);
54
55     clear r;
```

D.3 Ensaio 3

A Tabela 13 apresenta os valores que foram alterados no código para cada teste. O bloco a seguir apresenta o código utilizado no Ensaio 3.

Tabela 13 – Valores alterados para cada teste.

Teste	***speed***	***WaitUntilDone***
1	0,4	false
2	0,4	false
3	0,4	false
4	0,2	false
5	0,2	false
6	0,2	false
7	0,2	true

Fonte: Autoria Própria, 2021

```
1  %% ------- X - deslocamento precisao
2
3  % Criar um objeto ryze - conexao com o primeiro drone identificado por
4  % Wi-Fi
5      r = ryze()
6      takeoff(r);
7
8      cont = 0;
9      distance = 2;
```

```
10      speed = 0.4;
11      tObj = tic;
12
13      % Mover o drone e desbloquear a linha de comando
14      moveforward(r, 'Distance', distance, 'Speed', speed, ...
            'WaitUntilDone', false);
15
16      while(cont ≤ 200)
17        tplot = tic
18        readSpeed(r)
19        interval_speedread = toc(tplot)
20        cont = cont+1;
21      end
22
23      interval_all = toc(tObj);
24
25      land(r)
26      clear r
```

D.4 Ensaio 5

O bloco a seguir apresenta os códigos utilizados na geração de matrizes de deslocamento.

- **Trajetória de um quadrado**

```
1  %% Square
2  Square = [0 1 0; 1 0 0; 0 -1 0; -1 0 0]
3
4  sizeA = size(Square);
5
6  aux_sum = [0 0 0];
7  matrixPlot_Square = zeros(sizeA(1)+1,sizeA(2))
8  matrixPlot_Square(1,:) = aux_sum;
9
10 for i=1:1:sizeA(1)
11     matrixPlot_Square(i+1,:) = Square(i,:) + aux_sum
12     aux_sum = Square(i,:) + aux_sum
13 end
14
15 plot3(matrixPlot_Square(:,1), matrixPlot_Square(:,2), ...
       matrixPlot_Square(:,3))
```

- **Trajetória de um degrau nos eixos X e Y**

```
1
2  %% Step X-Y
3
4  StepX_Y = [1 0 0; 0 1 0; 1 0 0]
5
6  sizeStepX_Y = size(StepX_Y);
7
8  aux_sum = [0 0 0];
9  matrixPlot_StepX_Y = zeros(sizeStepX_Y(1)+1,sizeStepX_Y(2))
10 matrixPlot_StepX_Y(1,:) = aux_sum;
11
12 for i=1:1:sizeStepX_Y(1)
13     matrixPlot_StepX_Y(i+1,:) = StepX_Y(i,:) + aux_sum
14     aux_sum = StepX_Y(i,:) + aux_sum
15 end
16
17 plot3(matrixPlot_StepX_Y(:,1), matrixPlot_StepX_Y(:,2), ...
       matrixPlot_StepX_Y(:,3))
```

- **Trajetória de um degrau nos eixos Y e X**

```
1  %% Step Y-X
2
3  StepY_X = [0 1 0; 1 0 0; 0 1 0]
4
5  sizeStepY_X = size(StepY_X);
6
7  aux_sum = [0 0 0];
8  matrixPlot_StepY_X = zeros(sizeStepY_X(1)+1,sizeStepY_X(2))
9  matrixPlot_StepY_X(1,:) = aux_sum;
10
11 for i=1:1:sizeStepY_X(1)
12     matrixPlot_StepY_X(i+1,:) = StepY_X(i,:) + aux_sum
13     aux_sum = StepY_X(i,:) + aux_sum
14 end
15
16 plot3(matrixPlot_StepY_X(:,1), matrixPlot_StepY_X(:,2), ...
       matrixPlot_StepY_X(:,3))
```

- **Trajetória de um degrau nos eixos X e Z**

```
1
2  %% Step X-Z
3
4  StepX_Z = [1 0 0; 0 0 1; 1 0 0]
5
6  sizeStepX_Z = size(StepX_Z);
7
8  aux_sum = [0 0 0];
9  matrixPlot_StepX_Z = zeros(sizeStepX_Z(1)+1,sizeStepX_Z(2))
10 matrixPlot_StepX_Z(1,:) = aux_sum;
11
12 for i=1:1:sizeStepX_Z(1)
13     matrixPlot_StepX_Z(i+1,:) = StepX_Z(i,:) + aux_sum
14     aux_sum = StepX_Z(i,:) + aux_sum
15 end
16
17 plot3(matrixPlot_StepX_Z(:,1), matrixPlot_StepX_Z(:,2), ...
       matrixPlot_StepX_Z(:,3))
```

- **Trajetória de um degrau nos eixos Y e Z**

```
1
2  %% Step Y-Z
3
4  StepY_Z = [0 1 0; 0 0 1; 0 1 0]
5
6  sizeStepY_Z = size(StepY_Z);
7
8  aux_sum = [0 0 0];
9  matrixPlot_StepY_Z = zeros(sizeStepY_Z(1)+1,sizeStepY_Z(2))
10 matrixPlot_StepY_Z(1,:) = aux_sum;
11
12 for i=1:1:sizeStepY_Z(1)
13     matrixPlot_StepY_Z(i+1,:) = StepY_Z(i,:) + aux_sum
14     aux_sum = StepY_Z(i,:) + aux_sum
15 end
16
17 plot3(matrixPlot_StepY_Z(:,1), matrixPlot_StepY_Z(:,2), ...
       matrixPlot_StepY_Z(:,3))
```

- **Trajetória senoidal**

```
1
2  %% Sin X - No filter because X coordinate is always 0.3 (>0.2)
3  clc, clear all;
4
5  writeOK = 'false';
6  cont_ΔX = 1;
7  X(1) = 0;
8  cont_sin_matrix = 0;
9
10 for t = 1:0.5:20
11     t;
12     cont_ΔX = cont_ΔX+1;
13     X(cont_ΔX) = sin(t);
14     ΔX(cont_ΔX-1) = X(cont_ΔX) - X(cont_ΔX-1);
15 end
16
17 ΔXtransp = ΔX';
18 size_ΔXtransp = size(ΔXtransp);
19
20 %sin_matrix = zeros(size_ΔXtransp(1),3);
21
22 for cont_write = 1:1:(size_ΔXtransp(1))
23     SinX(cont_write,:) = [0.3 0 ΔXtransp(cont_write,1)];
24 end
25
26 size_sin_matrix = size(SinX);
27
28 aux_sum = [0 0 0];
29 matrixPlot_SinX = zeros(size_sin_matrix(1)+1,size_sin_matrix(2));
30 matrixPlot_SinX(1,:) = aux_sum;
31
32 for i = 1:1:size_sin_matrix(1)
33    matrixPlot_SinX(i+1,:) = SinX(i,:) + aux_sum(1,:);
34    aux_sum(1,:) = SinX(i,:) + aux_sum(1,:);
35 end
36
37 plot3(matrixPlot_SinX(:,1), matrixPlot_SinX(:,2), matrixPlot_SinX(:,3))
```

- **Trajetória em elipse**

```
1
2  %% Tilted ellipse
3  clc, clear all;
4
```

```
writeOK = 'false';
cont_Δ = 1;
X(1) = 0;
Y(1) = 0;
cont_Ellipse = 0;

for t = 1:0.5:20
    t;
    cont_Δ = cont_Δ + 1;
    X(cont_Δ) = sin(t);
    Y(cont_Δ) = cos(t)+sin(t);
    Z(cont_Δ) = 0.3;
    ΔX(cont_Δ-1) = X(cont_Δ) - X(cont_Δ-1);
    ΔY(cont_Δ-1) = Y(cont_Δ) - Y(cont_Δ-1);

end

matrixPlot_Ellipse = [X' Y' Z'];
plot3(matrixPlot_Ellipse(:,1),matrixPlot_Ellipse(:,2), ...
            matrixPlot_Ellipse(:,3))

ΔXtransp = ΔX';
ΔYtransp = ΔY';
ΔZtransp = Z';

size_ΔXtransp = size(ΔXtransp);

for cont_write = 1:1:(size_ΔXtransp(1))
    if( (ΔXtransp(cont_write) ≥ 0.2) || (ΔXtransp(cont_write) ≤ ...
       -0.2) || ...
            (ΔYtransp(cont_write) ≥ 0.2) || (ΔYtransp(cont_write) ≤ ...
               -0.2) || ...
            (ΔZtransp(cont_write) ≥ 0.2) || (ΔZtransp(cont_write) ≤ ...
               -0.2))
        writeOK = true;
        cont_Ellipse = cont_Ellipse + 1;
    else
        writeOK = false;
    end

    if writeOK == true
        Ellipse(cont_Ellipse,:) = [ΔXtransp(cont_write,1) Δ...
           Ytransp(cont_write,1) ΔZtransp(cont_write,1)];
    end
end

size_sin_matrix = size(Ellipse);
```

```
48 sub(1,:) = Ellipse(1,:);
49
50 for cont_plot = 2:1:(size_sin_matrix(1))
51    sub(cont_plot,1) = Ellipse(cont_plot,1) + Ellipse(cont_plot - 1,1);
52    sub(cont_plot,2) = Ellipse(cont_plot,2) + Ellipse(cont_plot - 1,2);
53    sub(cont_plot,3) = Ellipse(cont_plot,3);
54 end
55
56 figure(2)
57 plot3(sub(:,1),sub(:,2),sub(:,3))
```

- **Código que gera gráficos no *Editor* com os dados das matrizes enviadas do aplicativo para o *Workspace***

É válido ressaltar que as matrizes enviadas apresentavam o dado mais recente no topo da matriz e o mais antigo na última linha e por isso que foi gerada uma matriz invertida.

```
1  % Matrix [rows columns]
2  rowNcolumn = size(displacement_data);
3  matrix = zeros(rowNcolumn(1),rowNcolumn(2));
4  speed_matrix = zeros(rowNcolumn(1),rowNcolumn(2));
5
6  % Plot configurations
7  line = animatedline;
8  grid on;
9  title('Drone Navigation');
10 xlabel('X');
11 ylabel('Y');
12 zlabel('Altura');
13 view(3)
14 hold on
15
16 % Plot displacement 3D
17 figure(1)
18 i = 1;
19 elapsed_time = 0;
20 for row=(rowNcolumn(1)):-1:1
21     elapsed_time = displacement_data(row,4) + elapsed_time;
22     matrix(i,:) = [displacement_data(row,1) displacement_data(row,2) ...
           displacement_data(row,3) elapsed_time];
23     speed_matrix(i,:) = [speed_data(row,1) speed_data(row,2) ...
           speed_data(row,3) elapsed_time];
24     i = i+1;
```

```
25      addpoints(line, displacement_data(row,1), ...
            displacement_data(row,2), displacement_data(row,3));
26      drawnow
27  end
28
29  % Plot displacement separately
30  figure('Name', 'Distancia')
31  subplot(3,1,1)
32  plot(matrix(:,4), matrix(:,1))
33  title('Eixo X')
34
35  subplot(3,1,2)
36  plot(matrix(:,4), matrix(:,2))
37  title('Eixo Y')
38
39  subplot(3,1,3)
40  plot(matrix(:,4), matrix(:,3))
41  title('Eixo Z')
42
43  % Plot speed separately
44  figure('Name', 'Velocidade')
45  subplot(3,1,1)
46  plot(speed_matrix(:,4), speed_matrix(:,1))
47  title('Eixo X')
48
49  subplot(3,1,2)
50  plot(speed_matrix(:,4), speed_matrix(:,2))
51  title('Eixo Y')
52
53  subplot(3,1,3)
54  plot(speed_matrix(:,4), speed_matrix(:,3))
55  title('Eixo Z')
```

- **Código que salva as matrizes enviadas do aplicativo para o *Workspace* e as matrizes invertidas**

```
data_speed_received = speed_data;
data_matrix_speed_inverted = speed_matrix;

data_disp_received = displacement_data;
data_matrix_disp_inverted = matrix;

filename1 = ...
    'C:\Users\Trabalho\Desktop\nomepasta\app\data_speed_received.mat';
save(filename1, 'data_speed_received');

filename2 = ...
    'C:\Users\Trabalho\Desktop\nomepasta\app\data_matrix_speed_inverted.mat';
save(filename2, 'data_matrix_speed_inverted');

filename3 = ...
    'C:\Users\Trabalho\Desktop\nomepasta\app\data_disp_received.mat';
save(filename3, 'data_disp_received');

filename4 = ...
    'C:\Users\Trabalho\Desktop\nomepasta\app\data_matrix_disp_inverted.mat';
save(filename4, 'data_matrix_disp_inverted');
```