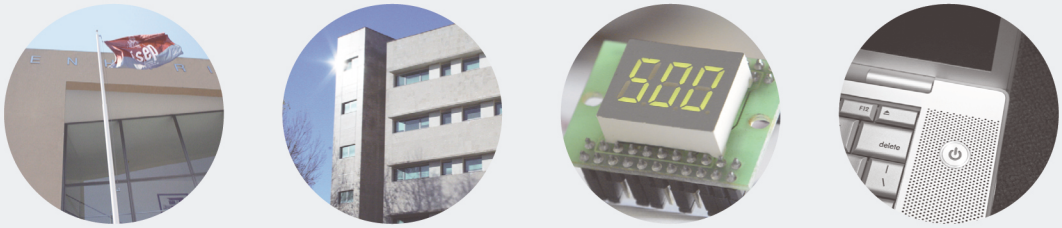
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Manobra de Inspeção de Eólicas com recurso a um UAV VTOL

DIANA CARINA CASTANHEIRA SALGADO
novembro de 2020

POLITÉCNICO
DO PORTO

Manobra de Inspeção de Eólicas com recurso a um UAV VTOL

Diana Carina Castanheira Salgado
Nº 1150895

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2020

Dissertação para satisfação parcial dos requisitos do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Diana Carina Castanheira Salgado

Nº 1150895

Supervisor: André Miguel Pinheiro Dias

apd@isep.ipp.pt

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

17 de Novembro de 2020

Agradecimentos

A realização desta dissertação contou com o apoio e incentivo incondicional de algumas pessoas, que tornaram todo este processo uma realidade e aos quais estarei eternamente grata.

Em primeiro lugar quero agradecer ao orientador desta dissertação, o Professor Doutor André Miguel Pinheiro Dias pelo desafio proposto, pela ajuda e disponibilidade demonstrada durante todo o processo de investigação. Sem a sua ajuda, a conclusão desta dissertação não teria sido possível.

Deixo também o meu agradecimento aos meus colegas de curso, em especial ao meu colega Rui Loureiro, com quem formei grupo em todas as unidades curriculares do mestrado. Obrigada pela ajuda, partilha de conhecimentos e pela oportunidade de trabalhar em grupo contigo. Sou grata por todos os momentos, mesmo os menos bons, mas também pelos momentos de conquista e pela aprendizagem adquirida. Sem a tua ajuda, chegar a este ponto não seria possível. O caminho que percorremos para finalizar as nossas dissertações é comum, por isso só desejo que o faças a "voar"numa linha recta e perfeita!

Às minhas amigas Rita Sanches e Sara Afonso que, mesmo não entendendo nada do que eu estava a fazer, sempre me incentivaram e me apoiaram no alcance os meus objetivos.

Aos investigadores do laboratório, em particular à Ana Pires pelos momentos de descontração, pela ajuda e integração no laboratório e pela partilha de apontamentos sempre que eu não conseguia tirá-los. Foste, sem dúvida, uma das melhores pessoas que conheci no mestrado e que pretendo guardar no coração. Desejo-te tudo de bom e que a tua vida seja feita de sucessos.

Agradeço também aos investigadores do Centro de Robótica e Sistemas Autónomos (CRAS) do INESCTEC, Alexandre Oliveira e Tiago Santos pela prontidão na ajuda prestada e pela partilha dos seus conhecimentos.

Aos meus pais, Libório Salgado e Ilda Castanheira, estarei eternamente grata pela oportunidade que me deram, pelo incentivo e pelo apoio constante durante estes cinco anos de estudo. Nunca me esquecerei de todas as idas a casa, onde vocês me diziam "não consegues fazer as cadeiras? Podes sempre levar algumas aqui de casa!". Agradeço também aos meus irmãos pelo apoio que me deram, em particular à minha irmã, Vera Salgado, por me oferecer a sua casa que é sempre o meu porto de abrigo, pelo carinho e pela ajuda que me ofereceu durante os meus tempos de estudante. A vocês estarei eternamente agradecida, não só pela ajuda que me deram mas também pela pessoa que me tornei.

Obrigada.

Diana Carina Castanheira Salgado

Abstract

The *Unmanned Aerial Vehicle* (UAV) are, more and more, inserted in the daily life of the Human Being. Its use has allowed the replacement of human beings in some tasks followed by risk, such as the inspection of electrical assets such as wind power. In addition to the human risk factor associated with these operations, the time spent on navigation is also high.

One solution found is to use multirotor in this type of inspection. Despite the advantages that these vehicles present, their autonomy does not allow for chain wind inspections. In this dissertation, we propose the development of a *Vertical Take-Off and Landing* (VTOL) that allows the inspection of wind farms on land and offshore, and the development of an inspection maneuver that allows better use of the potential of the vehicle when it is in airplane mode and drone mode.

To better understand the topic of trajectory planning, in the specific case of the inspection process, we proceeded to a more in-depth study of the algorithm developed by the ETH university, the Structural Inspection Path Planner, which consists of calculating the optimal path to inspect a structure. This study served as a starting point and comparison for the development of a new solution, designed specifically for wind farm inspection.

In terms of results and implementation, we carry out the development and assembly of a VTOL, the parameterization of an autopilot to work in hybrid mode, and the development of an inspection method directed to the specific case of the inspection of a wind farm.

Keywords: VTOL, wind inspection, trajectory planning, autonomous control

Esta página foi intencionalmente deixada em branco.

Resumo

Os *Unmanned Aerial Vehicle* (UAV) estão, cada vez mais, inseridos no quotidiano do Ser Humano. A sua utilização veio permitir a substituição do ser humano em algumas tarefas consideradas de risco, tais como a inspeção de ativos elétricos como eólicas. Para além do fator de risco humano associado a estas operações, o tempo despendido na inspeção também é elevado.

Uma solução encontrada passa pela utilização de *multirotors* neste tipo de inspeções. Apesar das vantagens que estes veículos apresentam, a sua autonomia não viabiliza inspeções de eólicas em cadeia. Nesta dissertação propõe-se o desenvolvimento de um *Vertical Take-Off and Landing* (VTOL) que permita a inspeção de parques eólicos em terra e em *offshore*, e o desenvolvimento de uma manobra de inspeção que permita um melhor aproveitamento das potencialidades do veículo quando se encontra em modo avião e em modo drone.

Nos sentido de melhor entender o tópico de planeamento de trajetórias, no caso concreto do processo de inspeção, procedeu-se ao estudo mais aprofundado do algoritmo desenvolvido pela universidade *Swiss Federal Institute of Technology in Zurich* (ETH), o *Structural Inspection Path Planner*, que consiste no cálculo do caminho ótimo para efetuar a inspeção de uma estrutura. Este estudo serviu como ponto de partida e de comparação para o desenvolvimento de uma solução nova, projetada especificamente para inspeção de eólicas.

Em termos de resultados e implementação, efetuamos o desenvolvimento e montagem de um VTOL, a parametrização de um *autopilot* para funcionar em modo híbrido e o desenvolvimento de um método de inspeção direcionado para o caso concreto da inspeção de uma eólica.

Palavras-Chave: VTOL, inspeção de eólicas, planeamento de trajetórias, controlo autónomo

Esta página foi intencionalmente deixada em branco.

Conteúdo

Lista de Figuras

Esta página foi intencionalmente deixada em branco.

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Lista de Algoritmos

Esta página foi intencionalmente deixada em branco.

Lista de Acrónimos

2D *Two-Dimensional*

3D *Three-Dimensional*

ADC *Analog Digital Converter*

AUV *Autonomous Underwater Vehicles*

BVP *Boundary Value Problem*

BVS *Boundary Value Solver*

CAN *Controller Area Network*

CRAS Centro de Robótica e Sistemas Autónomos

CSIRO *Commonwealth Scientific and Industrial Research Organisation*

EDP Energias de Portugal

ENU *East-North-Up*

ETH *Swiss Federal Institute of Technology in Zurich*

FOV *Field-Of-View*

HFOV *Horizontal Field-Of-View*

GPIO *General Purpose Input/Output*

GPS *Global Positioning System*

GNSS Global Navigation Satellite System

HITL Hardware In The Loop

ISEP *Instituto Superior de Engenharia do Porto*

I2C *Inter-Integrated Circuit*

LASMU *Laboratório de Sistemas Multirobóticos*

LiDAR *Light Detection And Ranging*

LKH *Lin-Kernighan-Helsgaun Heuristic*

LSA *Laboratório de Sistemas Autónomos*

PSO *Particle Swarm Optimization*

PWM *Pulse Width Modulation*

RRT* *Rapidly-exploring Random Tree*

ROS *Robotic Operating System*

ROV *Remotely Operated Underwater Vehicle*

SITL *Software In The Loop*

SPI *Serial Peripheral Interface*

STL *Standard Triangle Language*

TSP *Travelling Salesman Problem*

UART *Universal Asynchronous Receiver-Transmitter*

UAV *Unmanned Aerial Vehicle*

UGV *Unmanned Ground Vehicles*

USB *Universal Serial Bus*

VFOV *Vertical Field-Of-View*

VTOL *Vertical Take-Off and Landing*

Capítulo 1

Introdução

Nas últimas décadas, o progressivo avanço tecnológico a nível de controlo, processamento e sensorização têm permitido um enorme crescimento na utilização de robôs para fins civis e militares. Estes robôs podem ser aéreos, *Unmanned Aerial Vehicle* (UAV), *Autonomous Underwater Vehicles* (AUV), *Remotely Operated Underwater Vehicle* (ROV), ou *Unmanned Ground Vehicles* (UGV). Independentemente do meio onde se inserem, a sua utilização permitiu a substituição do ser humano em tarefas de grande risco. Serve de exemplo a deteção de fugas de gás [16] e a inspeção de infraestruturas [17,18]. Para além disso, é possível a utilização destes robôs para a inspeção de edifícios abandonados ou danificados, que apresentem perigo para o ser humano [19, 20] e cuja inspeção seja difícil de ser realizada com sucesso. Existe ainda a possibilidade da utilização destes robôs em cenários de busca e salvamento, [21, 22], em locais de difícil acesso, e no processo de deteção de vitimas, ou de recolha de dados estruturais para posterior salvamento.

O avanço tecnológico permitiu a melhoria na construção deste tipo de robôs, sendo que cada vez mais se tem reduzido o tamanho dos mesmos de modo a facilitar a sua locomoção. No entanto, esta redução de tamanho acarreta problemas, nomeadamente no controlo e estabilização dos robôs, tendo em conta a sua vulnerabilidade em relação ao meio onde operam. Para além disso, passou a ser fundamental a autonomia destes robôs, sendo capazes de evitar a colisão com outros objetos (*collision avoidance*) e capazes de planear a sua trajetória.

Dependendo do tipo de aplicação, existem diferentes tipos de UAVs desenvolvidos. Os multi-rotor UAVs [23] podem ser classificados dependendo do número de motores que apresentam, estando representado na figura 1.1(a) um *quadcopter* (quatro motores). Este tipo de UAV tem a capacidade de aterrar e descolar verticalmente, não necessi-

tando de muito espaço para tal. No entanto, apresentam um tempo de voo limitado, desperdiçando muita energia para que seja possível a sua estabilização. Os *Fixed-Wing* UAV, representado na figura 1.1(b), têm a capacidade de voos longitudinais, gastando uma menor energia quando comparados com os *quadcopters*, sendo uma boa opção para efetuar mapeamento de áreas. No entanto, apresentam uma grande desvantagem que é o facto de não conseguirem permanecer parados no ar. Colocar este tipo de UAV no ar também é particularmente difícil, sendo necessário um espaço suficientemente grande para descolar/aterrar o UAV. Os *Single Rotor*, figura 1.1(c), em termos estruturais, são muito semelhantes aos helicópteros, possuindo apenas um motor maior do que os do *quadcopter*. No entanto, o facto de apresentarem motores de maior potência conferem um maior risco à segurança do seu utilizador [24], sendo necessário algum tipo de treino para operar este tipo de UAV. Os *Vertical Take-Off and Landing* (VTOL) são veículos híbridos, juntando os benefícios da utilização de um *fixed-wing* com a utilização de um *quadcopter*. Assim, estes UAV tem a capacidade de descolar/aterrar como um *quadcopter* mas também conseguem voos longitudinais. Na figura 1.1(d) encontra-se uma representação deste tipo de UAV.

(a) *Quadcopter*

(b) *Fixed-Wing* [25]

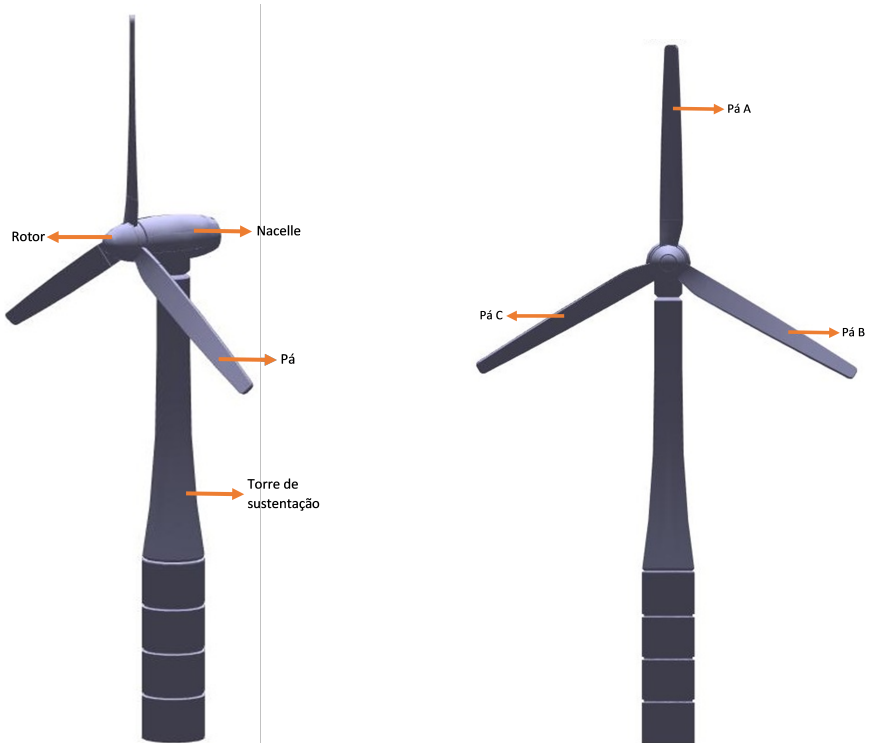
(c) *Single Rotor* [26]

(d) VTOL híbrido [27]

Figura 1.1: Diferentes tipos de UAV existentes

Na comunidade cientifica existe uma grande variedade de aplicações para os UAV, no entanto existe um caso que requer melhor atenção uma vez que vai de encontro com o tema desta dissertação: a utilização de UAV para a inspeção de infraestruturas, nomeadamente aerogeradores (também conhecidos como eólicas).

A inspeção de eólicas é importante pois permite verificar o estado dos materiais, evitando a sua corrosão. Neve, granizo, chuva, raios e pó são apenas alguns dos elementos aos quais as eólicas se encontram expostas. Uma eólica é constituída, essencialmente, por quatro componentes: a torre de sustentação, a *nacelle*, o *rotor* e as pás. O *rotor* é o componente que efetua a rotação, tendo acoplado as pás. A *nacelle* é a estrutura que se encontra a suportar o rotor rotor, conectada à torre de sustentação, sendo nesta zona onde se encontram o estator da eólica.. Na figura 1.2 encontra-se representada uma ventoinha eólica, estando, em 1.2(a), representados os principais componentes e, em 1.2(b), a enumeração das pás, que será utilizada posteriormente.



(a) Principais componentes de uma eólica

(b) Enumeração das pás da eólica utilizada nesta dissertação

Figura 1.2: Eólica

Durante muitos anos, para a inspeção das eólicas, eram utilizadas lentes telescópicas que permitiam a obtenção de imagens aproximadas das pás, verificando assim o seu estado. Para além deste método, existia a inspeção efetuada pelo ser humano, onde o operário teria que entrar na base da ventoinha e subir escadas até chegar à *nacelle* [28]. Na *nacelle*, o operário teria que escalar ao longo da pá para proceder à identificação e possível reparação do problema identificado. Na figura 1.3 encontram-se representadas algumas imagens alusivas à inspeção de uma eólica. Apesar de os operários utilizarem material adequado, este tipo de trabalho acarreta uma grande responsabilidade, sendo o risco de queda bastante elevado. Para além disso, é uma inspeção que apresenta um tempo de execução muito prolongado.

(a) Subida do operário até à *nacelle* [29]

(b) Chegada do operário à *nacelle* [29]

(c) Inspeção da estrutura exterior [29]

Figura 1.3: Inspeção a uma eólica efetuada por um operário

De modo a ser possível uma melhoria quer a nível de segurança dos operários, quer no tempo despendido numa inspeção, começaram a ser utilizados *quadcopters* para a inspeção exterior da eólica. Esta inspeção permite a deteção de falhas estruturais apresentando as seguintes vantagens [30]:

- Ambiente seguro durante a inspeção;
- Tempo de inspeção reduzido;
- Acesso a áreas que não seriam acessíveis pelo operário;
- Possível obtenção de imagens visuais e também imagens térmicas;
- Processo manual, introduzindo um nível de segurança grande uma vez que as eólicas encontram-se colocadas em locais com muito vento, sendo que a tele operação pode também ser um problema. Uma possível solução será a transição para uma solução autónoma, tendo por base a deteção da posição da eólica e o planeamento da trajetória.

No entanto, existem situações onde se procede à inspeção de não apenas uma, mas várias, distanciadas entre si (figura 1.4(a)). Estes locais são denominados por parques eólicos. Estes parques podem ser encontrados em terra ou em mar, sendo que em mar são apelidados de parques eólicos *offshore*.

A colocação de parques eólicos em alto mar acarreta algumas dificuldades, nomeadamente a montagem, a fixação e o transporte das fundações, e ainda a instalação de cabos submarinos responsáveis pelo transporte da energia gerada desde o parque eólico até terra. Em Portugal, e segundo [31], a primeira eólica colocada em alto mar data de 2011, tendo sido colocada perto da Póvoa de Varzim (figura 1.4(b)). Durante 5 anos, essa eólica conseguiu produzir energia suficiente para alimentar 1300 casas. Dado o sucesso desta experiência, a Energias de Portugal (EDP) passou para o planeamento da construção de um parque *offshore*, a 20 km da costa, perto de Viana do Castelo. Este parque tem assim a finalidade de produzir energia para abastecer 16 mil casas.

Para o caso de inspeção de parques eólicos, quer em terra quer em mar, a utilização de um *quadcopter* deixaria de ser viável pois este, devido à sua autonomia, não seria capaz de efetuar com sucesso a inspeção de todas as eólicas presentes no parque. Estas eólicas apresentam uma distância de 1 km entre elas. Para ter sucesso na inspeção, seria necessário proceder à troca da bateria do UAV em cada, o que tornaria o tempo de inspeção superior. Assim, e de modo a contornar este problema, sugere-se a utilização de um VTOL para a inspeção. Tal como referido anteriormente, este tipo de UAV tem a

capacidade de fazer a inspeção como um *quadcopter* e, de seguida, fazer a viagem entre eólicas como um *fixed-wing*. Isto permite uma redução no tempo total de inspeção do parque assim como um menor consumo de energia por parte do UAV.

(a) Parque eólico terrestre [1]

(b) Eólica em alto mar [31]

Figura 1.4: Parques eólicos EDP [1]

1.1 Enquadramento e Motivação

Esta dissertação foi proposta pelo *Laboratório de Sistemas Autónomos* (LSA)[1] do *Instituto Superior de Engenharia do Porto* (ISEP)[2], de modo a dar resposta a um problema ainda não investigado pelo laboratório. O LSA é um laboratório focado na investigação de soluções autónomas relacionadas com navegação, controlo e coordenação de variados robôs. Este laboratório conta com uma vasta diversidade de robôs desenvolvidos, quer aéreos, quer terrestres, quer marítimos, sendo alguns exemplo apresentados de seguida. O ROAZ II [32–34] (figura 1.5(a)), é um robô marítimo utilizado para tarefas de inspeção

[1] http://lsa.isep.ipp.pt/
[2] http://isep.ipp.pt/

subaquática e também para a coleta de dados. Outro exemplo é o TURTLE [35] (figura 1.5(b)), robô capaz de permanecer submerso durante longos períodos de tempo, sendo bastante utilizado para o mapeamento da costa portuguesa. Em relação a robôs aéreos, serve de exemplo o FALCOS [36,37] (figura 1.5(c)) que foi desenvolvido para deteção de fogos florestais, monitorização do meio ambiente e captura de imagens aéreas.

(a) ROAZ [38]

(b) TURTLE [35]

(c) FALCOS [39]

Figura 1.5: Exemplos de robôs desenvolvidos pelo laboratório LSA

Para além deste tipo de robôs, o laboratório trabalha com drones, sendo a sua área de aplicação a busca e salvamento [40] e a inspeção de linhas de alta tensão, este último em parceria com a EDP. Seguindo esta última linha de trabalho, surgiu a necessidade de trabalhar uma solução híbrida, de modo a aproveitar as vantagens da utilização de um *fixed-wing* para voos longitudinais com as vantagens da utilização de um drone para a parte de inspeção.

Assim, surgiu o primeiro VTOL, montado com material já existente no laboratório. Este VTOL, representado na figura 1.6, não conseguiu efetuar nenhuma transição para *fixed-wing* com sucesso devido a problemas estruturais.

Figura 1.6: VTOL desenvolvido no laboratório LSA

1.2 Objetivos

Esta dissertação tem como principal objetivo a criação de um método capaz de criar um caminho para a inspeção de eólicas, procurando endereçar o desenvolvimento de uma solução que permita não só optimizar o processo de inspeção, mas também melhorar a qualidade da informação adquirida. Esta inspeção deverá ser focada nas pás, uma vez que as mesmas correspondem à área de interesse de inspeção. Para tal, foram estabelecidos alguns objetivos que, no seu conjunto, permitirão chegar à solução final:

- Estudo de diferentes abordagens relacionadas com a inspeção de uma eólica;
- Instalação e compreensão do funcionamento do *Ardupilot* e PX4;
- Alteração do modo de voo de *fixed-wing* para *quadcopter* e exploração do modo híbrido;
- Implementação de uma manobra de inspeção capaz de visualizar todos os pontos de interesse;
- Realização da manobra de inspeção de forma autónoma;
- Validação do método implementado.

1.3 Estrutura do relatório

Esta dissertação encontra-se subdividida em 9 capítulos.

O primeiro capítulo é a introdução, onde é elaborada uma introdução ao assunto em estudo, sendo efetuado um breve enquadramento e onde são apresentados os objetivos para a realização deste projeto.

Segue-se o capítulo 2, estado de arte, onde é efetuado o levantamento dos diferentes tipos de UAV existentes, assim como diferentes formas de efetuar o planeamento de uma trajetória.

De seguida aparece o terceiro capítulo, onde são descritos teoricamente os diferentes temas estudados durante a realização do presente projeto.

O quarto capítulo corresponde à apresentação do algoritmo desenvolvido por investigadores da universidade *Swiss Federal Institute of Technology in Zurich* (ETH), que permite criar um caminho para a inspeção tendo como *input* a *mesh* do modelo em *Three-Dimensional* (3D) a ser inspecionado. É efetuado um estudo aprofundado da lógica existente neste algoritmo.

O quinto capítulo corresponde ao projeto, começando por ser descrita a arquitetura geral de sistema. De seguida, são enumerados os materiais necessário para a construção do veículo, os diferentes *firmwares* existentes e ainda uma descrição conceptual da manobra de inspeção desenvolvida. Este capítulo é concluído com as escolhas feitas para o projeto, assim como a arquitetura detalhada do sistema.

No capítulo seis é explicada a implementação realizada, quer na construção do veículo, quer no desenvolvimento do algoritmo apresentado no capítulo anterior. É ainda referido o algoritmo de controlo desenvolvido para o controlo autónomo do veículo e as alterações efetuadas no algoritmo da universidade ETH de modo a serem obtidos pontos de inspeção. Por fim, é apresentado o mundo desenvolvido para a simulação.

O capítulo 8 corresponde ao capítulo de resultados, onde é iniciado com a apresentação dos resultados dos testes de voo reais com o veículo utilizado. São também apresentados os resultados das simulações, nomeadamente a relação entre o peso e a massa e a simulação das manobras de inspeção.

Por fim, encontra-se o capítulo 9 onde são apresentadas as conclusões retiradas com a realização do presente projeto, sendo também apresentados aspetos a ser melhorados num trabalho futuro.

Esta página foi intencionalmente deixada em branco.

Capítulo 2

Estado de Arte

Neste capítulo serão abordados alguns temas considerados relevantes para a elaboração da presente dissertação. Serão apresentados os diferentes tipos de VTOL, bem como soluções comerciais. Para além disso, serão também abordados diferentes algoritmos utilizados para o planeamento de trajetórias.

2.1 Diferentes tipos de VTOL

Na atualidade, existem diversas situações onde a utilização de um VTOL híbrido é imprescindível. Tendo em conta as suas caraterísticas, o recurso à sua utilização tem sido cada vez maior, podendo ser esta para fins militares ou até mesmo para a busca e salvamento. Assim, é importante salientar que existem diferentes tipos de híbrido, sendo os mesmos apresentados de seguida.

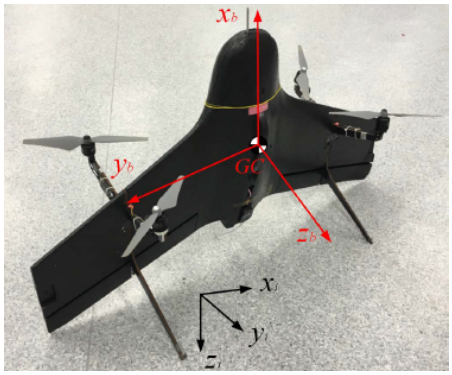
2.1.1 *Tail-sitter*

Um *tail-sitter* VTOL UAV descola em modo vertical utilizando a sua cauda para esse efeito, sendo efetuado posteriormente a inclinação horizontal da aeronave de modo a nivelar o seu voo. Se pretender aterrar, é efetuada uma inclinação de toda a aeronave de modo a que esta fique numa posição vertical, posição adequada para a aterragem. Na figura 2.1 é possível ver os passos efetuados pelo *tail-sitter* para uma transição.

Figura 2.1: Ilustração da transição efetuada por um *tail-sitter* VTOL [2]

Em [41], o autor refere que os *tail-sitter* são os UAV com a implementação mais simples uma vez que não requerem atuadores extra. No entanto, a descolagem e a aterragem do UAV constituem uma preocupação, uma vez que o desempenho do UAV é altamente prejudicado pela perturbação do vento causado pela própria asa. O autor apresenta então uma solução em termos de *design* e controlo de um *tail-sitter* com quatro motores, capaz de efetuar operações autonomamente no exterior. O autor afirma também que, quando comparado com um *tail-sitter* de apenas dois motores, um *tail-sitter* de quatro motores apresenta um controlo independente em *roll*, *pitch* e *yaw*, o que permite controlar o UAV de uma forma mais fácil na presença de vento. Esta caraterística permite uma maior liberdade no controlo das manobras do UAV.

Para além deste autor, são vários os estudos efetuados no que toca à implementação e desenvolvimento de *tail-sitter* VTOL. Como tal, servem de exemplo [2, 42], onde são abordados possíveis *design* da *framework* para o UAV, assim como controlo e otimização da sua trajetória. Na figura 2.2 estão representados os diferentes UAV referenciados na documentação citada anteriormente.



(a) *Tail-sitter* VTOL UAV desenvolvido em [41]

(b) Wingtra S100 [42]

(c) Pacflyer S100 [2]

Figura 2.2: *Tail-sitter* VTOL UAV desenvolvidos em diferentes literaturas

No entanto, em [43], são apresentadas alguns pontos negativos relativos à utilização deste tipo de VTOL. Tendo em atenção que os motores utilizados para voo longitudinal e para nivelar o UAV são os mesmos, não é possível o alcance de uma boa eficiência em ambos os modos de voo. Seria possível contornar este impedimento, aplicando um *propeller* com um *pitch* variável, uma vez que os *propellers* de multirotor necessitam de rodar mais rápido na descolagem enquanto que o *propeller* de *fixed wing* é utilizado para velocidades de rotação mais baixas. No entanto esta ação comprometeria o *design* mecânico da estrutura, a sua manutenção e o seu custo de fabrico. Para além disso, quanto maior a área da asa exposta aos ventos cruzados que se fazem sentir durante o voo longitudinal, pior será a *performance* desempenhada pelo *tail-sitter*.

2.1.2 *Tilt-rotor*

Semelhante ao *tail-sitter*, o *tilt-rotor* VTOL UAV utiliza os mesmos motores para a descolagem e para o voo longitudinal. No entanto, em vez de rodar toda a estrutura do UAV, são rodados apenas os seus motores. Assim, os motores encontram-se posicionados num determinado ângulo, permitindo descolar e aterrar eficazmente. À medida que o UAV vai ganhando velocidade, os seus motores vão rodando, passando então a servir para dar *thrust* em modo avião, sendo as asas do avião responsáveis pelo *lift* [43]. A figura 2.3 representa a transição efetuada por um *tilt-rotor* desde o momento da sua descolagem até à sua aterragem.

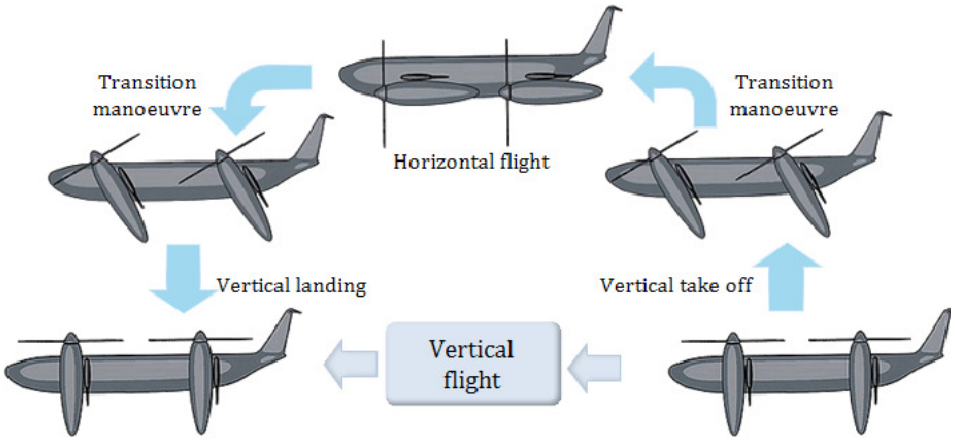
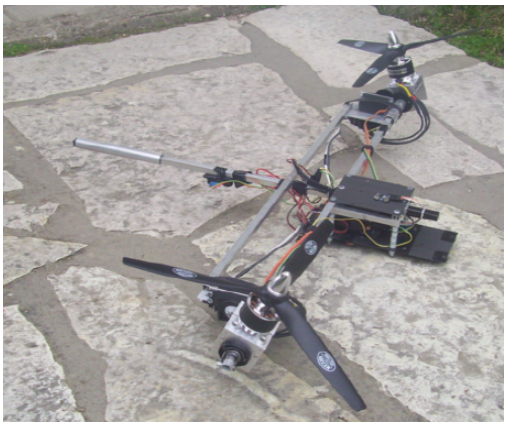


Figura 2.3: Exemplo da transição efetuada por tum *tilt-rotor* VTOL UAV [3]

Estes tipos de UAV tem a vantagem de utilizar os mesmos motores quer em modo drone, quer em modo avião, não existindo assim um peso extra nos diferentes tipos de voo. Existem várias abordagens para design deste tipo de UAV, como por exemplo em [3, 44–46], onde se aborda também a dinâmica e o controlo necessário para este tipo de UAV. A figura 2.4 apresenta diferentes estruturas desenvolvidas em diferentes literaturas.

(a) Quad Tiltrotor apresentado em [44]



(b) UPAT *tilt-rotor* apresentado em [45]

Figura 2.4: Protótipos de diferentes *tilt-rotor* apresentados em diferentes literaturas

2.1.3 *Dual System* VTOL

O *Dual System* VTOL é um UAV que apresenta dois sistemas de propulsão independentes, ou seja, apresenta motores específicos para descolagem/aterragem e outro motor para dar *thrust* quando se encontra em voo longitudinal.

Neste tipo de UAV, no modo de descolagem e aterragem, este comporta-se como um drone, estando o motor de *thrust* de avião desligado. Em modo de avião, ou seja, em voo longitudinal, os motores de *quadcopter* encontram-se desligados, poupando assim consumo de energia. Para o caso das transições, é possível ter os dois sistemas de propulsão ativos em simultâneo, evitando assim a queda do UAV.

O avançar dos tempos permitiu o desenvolvimento de diferentes *frames* e a procura pela melhor solução a utilizar. São vários os autores [43, 47, 48] que desenvolveram este tipo de UAV, sendo mencionado coisas tais como o modelo matemático e a descrição de como deverão ser efetuadas as transições. Para além disso, estes autores têm também em consideração coisas importantes em *fixed-wing*, tais como o ângulo de ataque e o coeficiente de *lift*.

(a) Design implementado em [47] (b) Design implementado em [48]

Figura 2.5: Diferentes implementações de *Dual System* apresentados em diferentes literaturas

A existência de dois sistemas de propulsão acaba por ser uma desvantagem uma vez que, em caso de voo longitudinal, o UAV tem que suportar os motores de *quadcopter*, diminuindo assim o *payload*.

2.2 Casos práticos da aplicação de UAV VTOL

Cada vez mais são as soluções desenvolvidas que tem como objetivo dar resposta a necessidades do quotidiano do ser humano. Para além de soluções comerciais, onde o utilizador compra o produto com um determinado fim de utilização, existem também competições com o objetivo de apoiar e incentivar a investigação na utilização de híbridos. Assim, de seguida, será apresentado o Canberra UAV, um VTOL híbrido utilizado na área de busca e salvamento.

2.2.1 Canberra UAV

O Canberra UAV é uma aeronave que foi desenvolvida para a participação no *UAV Challenge Outback Rescue* que é uma competição que decorre de dois em dois anos em Queensland, Austrália. Esta competição tem como objetivo a apresentação de aeronaves capazes de efetuar um determinado percurso que simula uma situação real, a procura e resgate de um ser humano, com uma duração total de uma hora.

Figura 2.6: Canberra UAV [4]

A *UAV Challenge Outback Rescue* teve o seu inicio no ano de 2007 sendo gerido pelo *Commonwealth Scientific and Industrial Research Organisation* (CSIRO), órgão de pesquisa australiana, e pela *Queensland University of Technology*. Esta prova começou a ser realizada pois existia a necessidade da utilização de UAV na procura e resgate em casos catastróficos. Até 2014, esta consistia na construção de uma aeronave que fosse capaz de sobrevoar uma área previamente determinada, fazer a localização de um espantalho, mais conhecido por Joe, e largar uma garrafa de água. Inicialmente, esta era uma competição com grande complexidade não sendo necessário garantir uma descolagem e aterragem autónomas [49]. Só em 2014, e após alguns anos de tentativas, é que os objetivos propostos pelos organizadores da prova foram cumpridos com sucesso, sendo o Canberra UAV o vencedor da competição.

Qualquer tipo de equipa pode participar, no entanto nem todas chegam a fazer a apresentação da sua aeronave em Queensland. As equipas vão sendo eliminadas através da documentação que vão ter que indo entregando durante o ano anterior à competição.

Figura 2.7: Joe, o espantalho [5]

Tendo em conta o sucesso na concretização da missão em 2014, foi necessário fazer umas alterações à competição na edição seguinte. Assim, em 2016, era necessário que a aeronave fizesse uma descolagem autónoma, localizasse o espantalho, aterrasse e recolhesse uma amostra de sangue. A aeronave tinha que ser capaz de encontrar um local que estivesse no mínimo a 30 metros e no máximo a 80 metros do espantalho de modo a garantir uma distância de segurança caso a aterragem corresse mal. Após a aterragem, a aeronave tinha que entrar num modo onde teria que esperar que fosse colocada a amostra de sangue. Para a confirmação dessa colocação, teria que se carregar num botão e só passado um minuto a aeronave voltaria a descolar. Após a descolagem, a aeronave tinha que ser capaz de voltar ao ponto inicial para deixar a amostra de sangue recolhida. Nas edições anteriores, um avião de asa fixa era mais do que suficiente para fazer o depósito da garrafa de água. No entanto, tendo em conta os requisitos da prova em 2016, foi necessária a utilização de aeronaves diferentes, sendo estas aeronaves capazes de efetuar descolagem e aterragem vertical mas que o voo fosse feito em modo asa fixa.

Tendo aeronaves diferentes foi necessário garantir que existiam modos de segurança para o caso de as comunicações falharem. Assim, para ser possível a recolha da amostra de sangue, a aeronave teria que estar sempre em contacto com a base. Para além disso, caso fosse perdida a conexão por mais de 10 segundos, a aeronave teria que ser capaz de retornar ao ponto de partida. Foi também importante demarcar uma área de modo a que, caso as aeronaves passassem essa área, estas teriam que se desligar. Esta área é denominada por *geofencing* e tinha como funcionalidade garantir a segurança caso alguma aeronave ficasse sem controlo.

2.3 Planeamento de trajetórias

Existem diferentes métodos para que seja possível o planeamento de trajetórias, que consiste no planeamento de um caminho a ser seguido pelo veículo utilizado. De seguida, serão expostos alguns métodos que dão resposta a este problema.

2.3.1 *Waypoints*

Para este tipo de planeamento, é necessário que sejam definidas as coordenadas a ser seguidas pelo veículo. Estas coordenadas podem ser locais, tendo como origem o ponto de descolagem do veículo, ou globais, correspondendo às coordenadas de *Global Positioning System* (GPS).

O utilizador, para utilizar este método, terá que identificar os pontos para o qual o UAV se deve dirigir. Esta introdução poderá ser feita de duas formas distintas, ou seja, poderá ser utilizado um *software* onde o utilizador defina as coordenadas de uma forma interativa, ou poderá ser utilizado um ficheiro que contenha essas mesmas coordenadas. Independentemente do método a utilizar, após o utilizador possuir as coordenadas dos pontos de interesse, é necessário enviar essa informação para o UAV. Após a receção, e assim que possuir um sinal de GPS válido, o UAV vai então percorrer os *waypoints* recebidos.

2.3.2 Mapas 3D

Neste tipo de planeamentos, é dado um mapa 3D no qual o UAV terá que ir de um ponto a outro, escolhendo o melhor caminho. Na figura 2.8 encontra-se o planeamento de uma trajetória recorrendo a um mapa 3D.

No entanto, para além do mapa 3D, é necessário utilizar algum algoritmo que seja capaz de calcular o melhor trajeto entre os pontos pretendidos. Assim, um bom exemplo disso será a utilização de um *Rapidly-exploring Random Tree* (RRT*). Este algoritmo começa por colocar no mundo vários pontos, entre uma posição inicial e uma posição final. De seguida, gera, de uma forma iterativa, ligações entre os pontos gerados, até encontrar o melhor trajeto a ser efetuado.

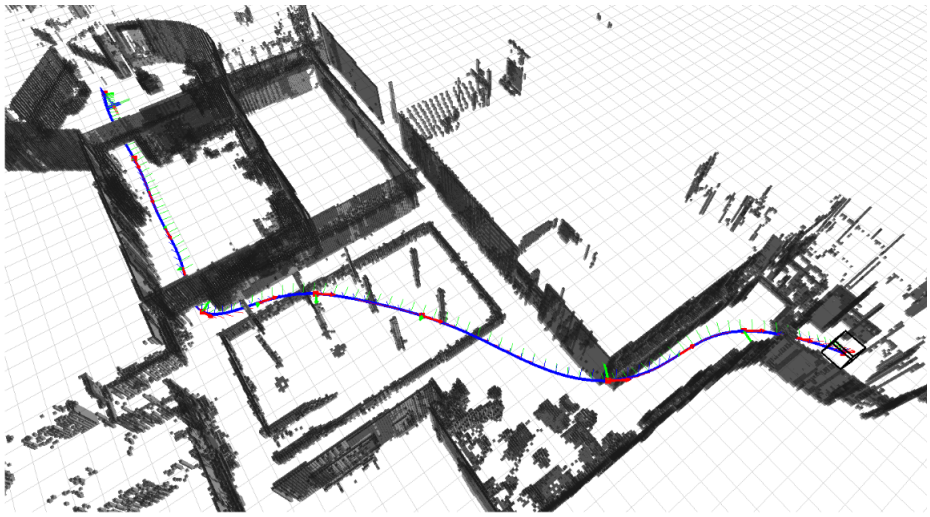


Figura 2.8: Trajetória 3D gerada num espaço real [6]

Na literatura, existem vários estudos e implementações onde se utiliza o método RRT* para o planeamento de trajetórias, como servem de exemplo [6, 50–52].

Os métodos apresentados anteriormente apresentam uma desvantagem que consiste na necessidade de existir conhecimento prévio em relação ao ambiente para o qual se pretende o planeamento da trajetória. Em caso de alteração desse ambiente, é necessário voltar a planear o trajeto encontrado. Assim, surgiu a necessidade da utilização da visão para o planeamento de trajetórias.

Em [7], é apresentada uma solução, onde se utiliza a visão para o planeamento de trajetória para explorar um local 3D desconhecido. Para tal, o ambiente passa a ser conhecido como um mapa de ocupação [53] construído através de imagens obtidas do ambiente em questão. Para além do espaço ocupado, é também demonstrado o espaço que se encontra livre e o espaço desconhecido. Após a obtenção do espaço livre, é utilizado um RRT* para o cálculo do melhor caminho. Na figura 2.9 encontra-se representado um exemplo de um mapa de ocupação, correspondendo a linha preta ao caminho calculado e a linha azul ao caminho efetuado pelo UAV.

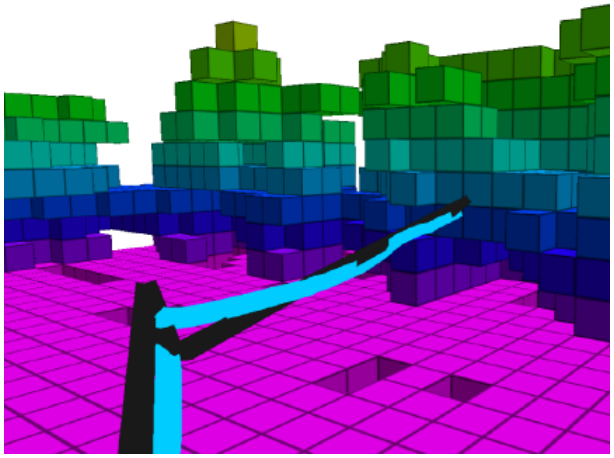


Figura 2.9: Exemplo de um mapa de ocupação contendo a trajetória planeada citado em [7]

2.4 Inspeção de estruturas

O planeamento de trajetórias tem sido cada vez mais utilizado para a inspeção de estruturas. Na literatura, existem diferentes abordagens estudadas, implementadas e testadas, sendo de seguida abordadas algumas destas soluções.

No trabalho de [54] é apresentada uma abordagem para a inspeção de um campo que contém painéis fotovoltaicos utilizando um multirotor UAV. O algoritmo utiliza *waypoints* para definir os painéis que deverão ser visitados, de modo a ser verificado o estado do mesmo. De modo a ser efetuada uma trajetória o mais suave possível, o autor propõe a criação de uma curva que contenha todos os *waypoints* pretendidos, denominada de curva de Bezier. Para a otimização desta curva, o autor propõe ainda a utilização do método *Particle Swarm Optimization* (PSO). Este método encontra os melhores pontos entre os *waypoints* definidos, otimizando assim a curva obtida, ou seja, tornando a curva obtida o mais suave possível. No que toca a resultados, este algoritmo tornou-se eficiente para planeamento de uma trajetória, sendo aplicado unicamente em ambientes sem vento. O autor considera que existem alguns pormenores que poderiam ser alterados num implementação futura de modo a tornar o algoritmo apropriado para a inspeção de quintas com elevadas dimensões.

Em [55] é implementada uma solução para a inspeção de uma eólica, utilizando um multirotor. O planeamento de trajetória é efetuado tendo como base conhecimento a priori de: número de pás do rotor, diâmetro das pás e distância de inspeção. Com base

na geometria da ventoinha, são gerados *waypoints* e é aplicada uma interpolação cúbica para gerar um caminho contínuo entre os *waypoints*. Para *collision avoidance*, o autor refere a utilização de um LiDAR, aplicando uma malha PD para a correção dos pontos obtidos. Segundo os testes efetuados pelo autor, a utilização do LiDAR para *collision avoidance* provou ser eficiente em ambientes *indoor*, tendo os objetos sido detetados com sucesso. O autor refere ainda que os próximos testes serão *outdoor*, onde será possível testar o algoritmo num ambiente real, sujeito ao efeito de vento e do sol.

O autor de [11] e [56] apresenta um algoritmo de planeamento de trajetórias capaz de calcular uma solução eficiente para uma estrutura 3D representada por uma *mesh* triangular. Este algoritmo seleciona pontos que permitam ver cada triângulo da *mesh* a ser inspecionada. É um algoritmo iterativo que, em cada iteração, calcula o melhor caminho para cada ponto utilizando o *Lin-Kernighan-Helsgaun Heuristic* (LKH). Em cada caminho calculado é também calculado o seu custo, utilizando para tal o *Boundary Value Solver* (BVS). Os resultados apresentados pelo autor sugerem que o algoritmo é bem sucedido para a inspeção de diferentes estruturas. Este algoritmo encontra-se disponibilizado *online*, no *Git Hub* do ETH, podendo ser utilizado por qualquer utilizador.

2.5 Discussão do Estado de Arte

Os diferentes tipos de VTOL apresentados contribuíram para um melhor conhecimento sobre as diferentes opções que poderiam ser utilizadas. Para além disso, foi possível clarificar o comportamento no que toca à descolagem e aterragem das diferentes estruturas.

Em relação ao exemplo dado, apesar de não ter a mesma finalidade de aplicação onde esta dissertação se insere, o *firmware* utilizado vai de encontro com o que será abordado posteriormente, neste caso o Ardupilot. No entanto, existem algumas diferenças entre o Canberra UAV e o VTOL em teste nesta dissertação. O Canberra UAV possui o motor frontal alimentado a gasolina enquanto que o resto da eletrónica a bordo é alimentada através de baterias, o que não acontece no VTOL que se pretende desenvolver durante a dissertação, onde toda a eletrónica terá de ser alimentada através de baterias.

No que toca ao planeamento de trajetórias, foi possível adquirir um maior conhecimento em relação aos métodos disponíveis para tal função. Este conhecimento vai permitir, posteriormente, uma melhor escolha sobre o método a implementar de modo a serem obtidos os melhores resultados possíveis.

Foram também estudados diferentes algoritmos desenvolvidos para a inspeção de estruturas. Este estudo permitiu adquirir um melhor conhecimento sobre as ideias desenvolvidas e testadas até à data, tendo existido uma maior curiosidade em relação ao

algoritmo apresentado em [11]. Tendo em atenção a disponibilidade deste algoritmo *online*, foi efetuado um estudo mais aprofundado sobre o mesmo, existindo um capítulo dedicado ao tema.

Esta página foi intencionalmente deixada em branco.

Capítulo 3

Fundamentos Teóricos

Neste capítulo vão ser abordados os temas que irão fazer parte do desenvolvimento desta dissertação. Será efetuada uma breve introdução à *framework* utilizada por robôs, seguindo-se uma breve explicação sobre os ângulos de Euler. Para além disso, serão abordados diferentes algoritmos que se encontram a ser utilizados pelo *Structural Inspection Path Planning*.

3.1 *Robotic Operating System* (ROS)

> The ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.[1]

ROS é uma *framework*, *open-source*, que se encontra a ser utilizada pela comunidade robótica, pelo fato de disponibilizar um conjunto de funcionalidades que permitem que a curva de integração de *software* em robôs seja mais célere.

Para esse efeito, disponibiliza um conjunto de ferramentas e bibliotecas que têm como objetivo simplificar o processo de desenvolvimento de funcionalidades em robôs, sendo possível a sua utilização numa vasta variedade de plataformas robóticas. Considerando o impacto na mesma na comunidade robótica, está disponível uma vasta gama de livros [57, 58] e de recursos *online* que permite que a curva de aprendizagem seja rápida.

Em [59], são mencionadas algumas caraterísticas sobre o ROS, nomeadamente:

- Comunicação entre processos recorrendo à tipologia Peer-To-Peer (P2P);

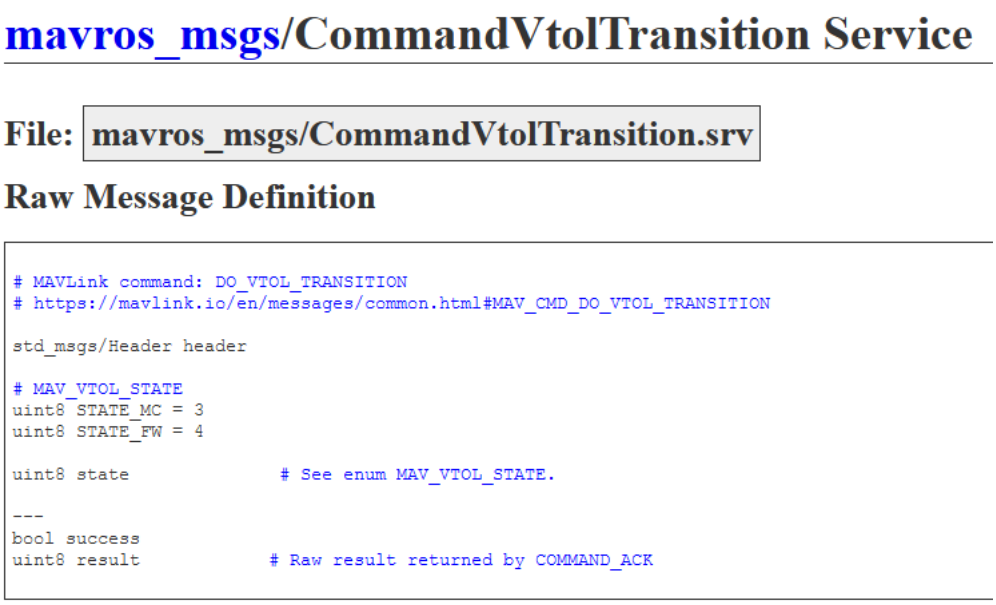
[1] https://www.ros.org/about-ros/

- Suporta diferentes linguagens, nomeadamente C++, Python, Octave e LISP;

- Disponibiliza ferramentas que permitem compilar e correr os diferentes componentes de ROS. Essas ferramentas servem, por exemplo, para navegação na *source code*, definir parâmetros, visualizar a conexão *peer-to-peer*, entre outros;

Na implementação de ROS, os conceitos fundamentais são nós, mensagens, tópicos e serviços. Os nós são processos que efetuam a computação necessária, comunicando entre si através do envio de mensagens. As mensagens são estruturas de dados que podem ser *integer*, *float*, *boolean*, entre outros. Um nó envia uma determinada mensagem para um tópico, sendo designado de *publisher* nestas situações. No entanto, um nó pode também subscrever-se a um determinado tópico, sendo então designado por *subscriber*. É importante referir que, para um determinado tópico, é possível existirem diversos *publishers* e *subscribers*, assim como um nó pode estar a publicar/subscrever para diferentes tópicos.

Em situações onde seja necessária uma comunicação síncrona, são utilizados serviços que são definidos por um nome e um determinado tipo de mensagens, uma para o pedido e outra para a resposta. O nome dos serviços é único, ou seja, só poderá existir um serviço chamado "Waypoint", por exemplo. Na figura 3.1 está representado o serviço que permite a transição de um VTOL.



mavros_msgs/CommandVtolTransition Service

File: mavros_msgs/CommandVtolTransition.srv

Raw Message Definition

```
# MAVLink command: DO_VTOL_TRANSITION
# https://mavlink.io/en/messages/common.html#MAV_CMD_DO_VTOL_TRANSITION

std_msgs/Header header

# MAV_VTOL_STATE
uint8 STATE_MC = 3
uint8 STATE_FW = 4

uint8 state              # See enum MAV_VTOL_STATE.

---
bool success
uint8 result            # Raw result returned by COMMAND_ACK
```

Figura 3.1: Serviço de ROS que permite efetuar a transição de um VTOL

A informação relacionada com ROS encontra-se devidamente documentada online [2], existindo fóruns onde o utilizador pode colocar as suas dúvidas à comunidade de *developers* [3].

3.2 Ângulos de Euler

Os ângulos de Euler são utilizados para descrever a orientação de um corpo num espaço 3D, sendo denominados por XYZ [60, 61]. No entanto, estes ângulos podem também ser utilizados para descrever a orientação relativa entre dois corpos, com sistemas de coordenadas diferentes. Estes ângulos são, representados na figura 3.2:

- Roll (ϕ): Rotação em torno do eixo X;
- Pitch (θ): Rotação em torno do eixo Y;
- Yaw (ψ): Rotação em torno do eixo Z.

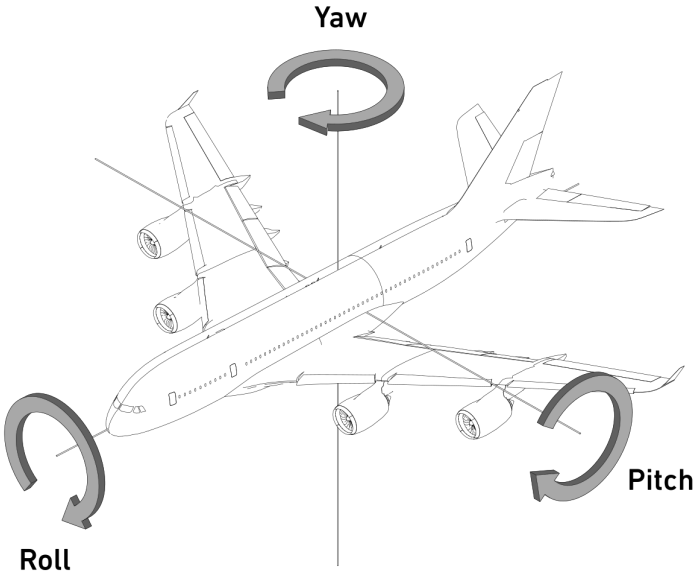


Figura 3.2: Ângulos de Euler representados num corpo rígido [8]

[2] http://wiki.ros.org/mavros_msgs
[3] https://discourse.ros.org/

Tendo como base um determinado referencial de coordenadas, é possível obter as matrizes de rotação nos eixos x, y e z do corpo rígido, em relação a esse mesmo eixo. Assim, as matrizes de rotação de um corpo rígido são:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

É assim então possível relacionar os eixos x, y e z do corpo rígido com um sistema de coordenadas fixo, utilizando para tal a matriz apresentada de seguida.

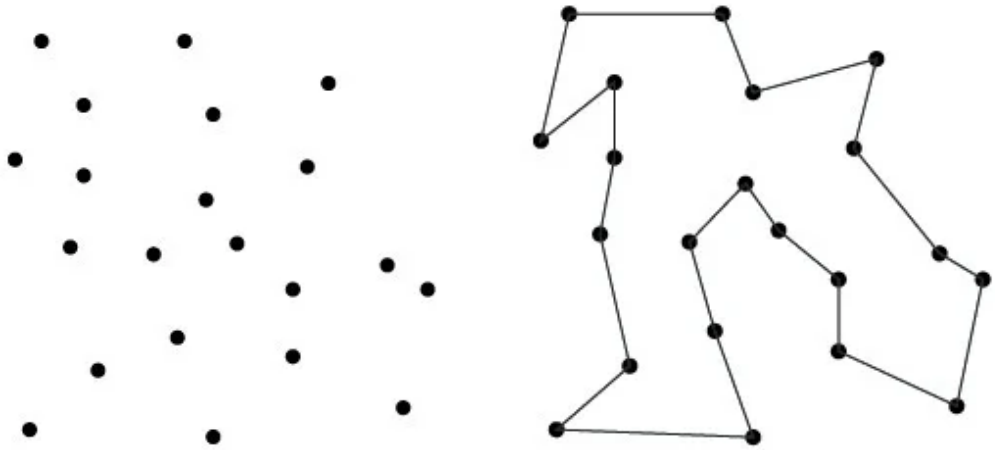
$$R_{xyz} = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi)-c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi)+s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi)+c(\phi)c(\psi) & (\phi)s(\theta)s(\psi)-s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}$$

Nesta matriz, $c(\phi) = \cos\phi$, $s(\phi) = \sin\phi$, $c(\theta) = \cos\theta$, $s(\theta) = \sin\theta$, $c(\psi) = \cos\psi$ e $s(\psi) = \sin\psi$.

3.3 *Travelling Salesman Problem* (TSP)

O TSP corresponde ao problema de encontrar o caminho mais curto a ser seguido dada uma lista de pontos que tem de ser percorridos [62]. A comunidade científica tem passado as últimas décadas a estudar este assunto, tentando encontrar a melhor solução para que o caminho percorrido apresente o menor custo e a menor distância possível. Em teoria, este problema seria de fácil resolução. No entanto, na prática, os cálculos efetuados para encontrar o melhor caminho são complexos, aumentando a sua complexidade com a adição de pontos a ser percorridos. Na figura 3.3(a) encontra-se representado um exemplo de um conjunto de pontos a ser percorridos. Por sua vez, na figura 3.3(b)

encontra-se representado o melhor caminho encontrado.



(a) Pontos que tem que ser percorridos no caminho [62]

(b) Melhor caminho encontrado [62]

Figura 3.3: Representação visual do TSP

Os algoritmos utilizados para a resolução deste problema encontram-se divididos em duas classes [9]:

- **Algoritmos exatos**: neste tipo de algoritmos, a solução é encontrada num número limitado de passos. São algoritmos muito complexos, exigindo uma grande capacidade de computação por parte do computador, como se encontra documentado em [63, 64].
- **Algoritmos aproximados (ou heurísticos)**: neste tipo de algoritmos, as soluções obtidas podem não ser otimizadas. Estes são algoritmos simples, com tempo de processamento curto.

Existem alguns métodos que são muito utilizados para dar resposta a este problema, nomeadamente:

- **The Brute-Force Approach**: este método calcula todos os caminhos possíveis, listando-os. De seguida, é necessário calcular a distância entre cada caminho calculado, escolhendo então aquele que apresentar uma distância mais pequena.
- **The Branch and Bound Method**: este método subdivide o problema inicial em problemas mais pequenos. Para cada sub-problema, são calculadas várias soluções,

sendo escolhida a melhor solução para o sub-problema em questão. A solução escolhida para um determinado sub-problema acaba por interferir nas soluções de outros sub-problemas. Para resolver o TSP utilizando este método, é necessário escolher o ponto inicial, atribuindo-lhe um limite elevado. De seguida, é necessário calcular o menor custo entre o próximo ponto e o ponto atual, adicionando a distância entre pontos à distância atual. É necessário repetir este processo enquanto a distância for inferior ao limite definido.

- **The Nearest Neighbor Method**: neste método, o ponto escolhido será aquele que se encontra mais próximo do ponto atual. Vai-se sempre seguindo esta lógica, visitando sempre o ponto mais próximo. Quando todos os pontos forem visitados, é necessário regressar ao ponto inicial.

Ao longos dos anos, várias foram as tentativas para encontrar a melhor solução para a resolução do TSP, sendo apresentadas algumas em [65–67].

3.4 LKH

O LKH é um algoritmo heurístico, utilizado para dar resposta ao TSP. Em [9] é efetuada uma descrição deste algoritmo, sendo inicialmente apresentado o algoritmo original e, de seguida, são apresentadas as melhorias efetuadas ao mesmo algoritmo. Assim, é necessário o cálculo prévio de um caminho a ser seguido. A esse caminho gerado será aplicado o algoritmo LKH que tem como objetivo eliminar ligações desnecessárias, sendo λ o número de ligações a retirar. Em cada iteração, as ligações do percurso atual são substituídas por novas ligações, sendo assim obtido um caminho mais curto, que apresente o menor custo possível.

O número de ligações a serem retiradas poderiam ser definidas pelo utilizador, sendo muitas vezes utilizado λ=2 ou λ=3, no entanto no estudo apresentado em [68] utilizaram λ=4 e λ=5. Esta definição de λ constituía um problema, uma vez que o utilizador não sabia, à *priori*, qual o melhor valor a ser utilizado. Assim, Lin and Kernighan introduziram uma melhoria ao seu algoritmo, nomeadamente a definição automática de λ. Durante a execução do algoritmo, o valor de λ vai alterando em cada iteração, de modo a dar a melhor resposta possível ao problema.

Sendo T o melhor caminho atual, o algoritmo, para cada iteração, vai encontrar os dois melhores conjuntos de ligações entre pontos, neste caso X=x_1,...,x_r e Y=y_1,...y_r. De seguida, as ligações de X serão substituidas pelas ligações de Y, sendo obtido assim o

melhor caminho possível. Esta substituição é denominada de movimento *r-opt*, e a figura 3.4 representa um movimento *3-opt*, ou seja, serão retiradas 3 ligações do caminho.

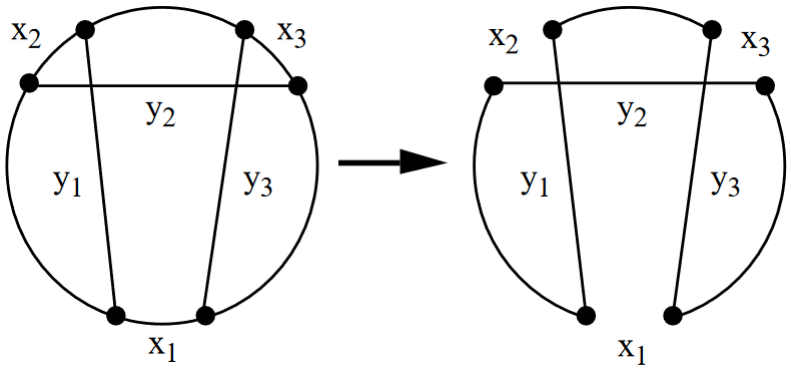


Figura 3.4: Movimento *3-opt* [9]

3.5 *Boundary Value Problem* (BVP)

Num BVP, o principal objetivo consiste em encontrar soluções para uma equação diferencial, satisfazendo também as condições de fronteira (*boundary conditions*) [69]. Estas condições são então aplicadas no inicio e no fim do intervalo considerado (*boundary values*).

Por exemplo, numa situação onde existe uma equação diferencial $y(t) = y$, sendo t a variável que representa o tempo. Para cada *boundary value*, seria necessário encontrar uma solução e uma *boundary condition*. A equação 3.1 apresenta um exemplo do que poderiam ser as *boundary conditions* para o *boundary value* t_0. As soluções encontradas para dar resposta a este problema são conhecidas por BVS.

$$y(t_0) = y_0 \qquad y'(t_0) = y'_0 \tag{3.1}$$

3.6 RRT*

O RRT é um algoritmo que tem como objetivo encontrar o melhor caminho entre dois pontos. Neste algoritmo, são gerados pontos aleatoriamente, desde um ponto inicial até um ponto final, gerando uma árvore de ligações entre pontos. Os pontos são conectados ao seu ponto mais próximo, sendo gerado assim o caminho mais curto entre dois pontos pretendidos. Sempre que é gerado um novo ponto, é necessário verificar se esse ponto

não se encontra num objeto, evitando assim colisão com o objeto. Este algoritmo acaba sempre que o ponto destino for atingido, sendo uma grande vantagem a sua rapidez de execução.

Por sua vez, o RRT* é a versão otimizada do algoritmo RRT. O principio básico do funcionamento do RRT* é o mesmo que o RRT, no entanto existem dois aspetos que foram adicionados ao RRT* que culminam em resultados diferentes entre estes dois algoritmos.

O primeiro aspeto está relacionado com o cálculo de distância entre o ponto a ser considerado e o seu ponto vizinho. Esta distância é denominada de custo, e as ligações terão sempre por base o menor custo possível.

O segundo aspeto encontra-se relacionado com a re-análise da árvore gerada anteriormente. Após ser obtida a árvore de pontos, será adicionado um novo ponto e será analisado o custo entre esse ponto e os seus pontos vizinho. Caso o custo seja inferior ao já calculado, a ligação é alterada com sucesso. Caso, contrário, a árvore será mantida. Esta re-análise permite suavizar o caminho anteriormente calculado. Um exemplo da implementação deste algoritmo encontra-se representado na figura 3.5.

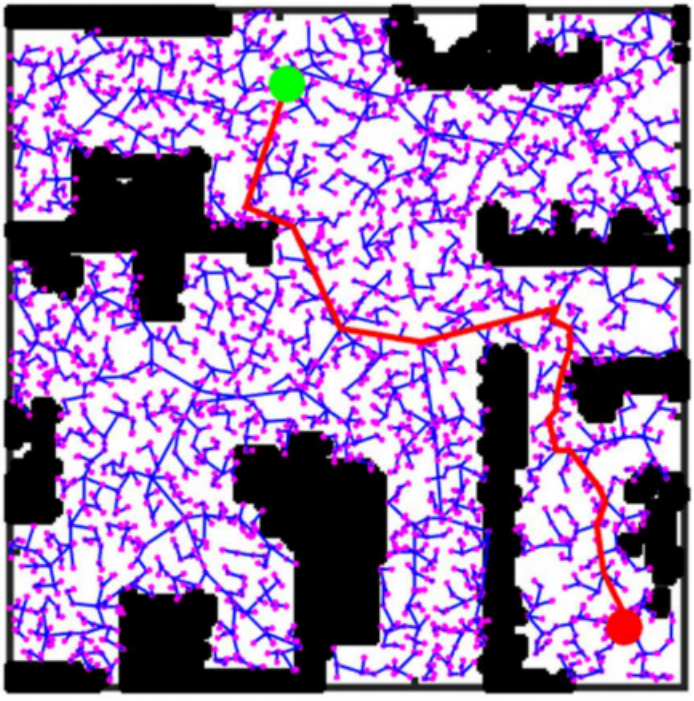


Figura 3.5: Exemplo da implementação do algoritmo RRT* [10]

Apesar de ser uma melhoria quando comparado com o RRT, o RRT* possui um tempo de computação superior ao seu antecessor. Em [10], o autor pretende comparar diferentes algoritmos utilizados para o planeamento de trajetórias. Para esta comparação, o autor utiliza cinco mapas diferentes, tendo em atenção o tempo de execução de cada algoritmo para cada mapa. Assim, a figura 3.6 apresenta os tempos de execução dos diferentes algoritmos para os diferentes mapas.

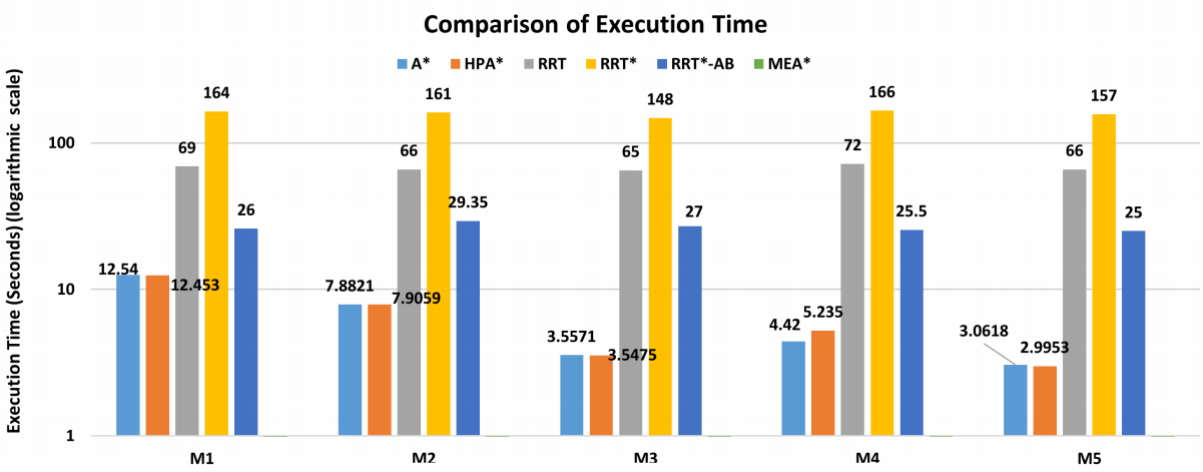


Figura 3.6: Exemplo da implementação do algoritmo RRT* [10]

Os dados apresentados na figura 3.6 permitem provar assim que o algoritmo RRT* apresenta um maior tempo para o cálculo da melhor trajetória. Para o caso desta dissertação, apenas é relevantes dos dados obtidos para o algoritmo RRT e RRT*.

Esta página foi intencionalmente deixada em branco.

Capítulo 4

Exploração do método *Structural Inspection Path Planning* aplicado ao processo de Inspeção de Eólicas

Neste capítulo será avaliado o método *Structural Inspection Path Planning* no processo de inspeção de eólicas. Com este trabalho é pretendido aumentar o conhecimento sobre o planeamento de trajetórias para a inspeção de estruturas. Tendo em conta o caso de estudo desta dissertação, qual será o desempenho do método quando aplicado no processo de inspeção de uma eólica?

O *Structural Inspection Path Planning* é um algoritmo iterativo, desenvolvido por investigadores da universidade ETH, que tem como principal objetivo encontrar um caminho viável para se efetuar uma inspeção com um robô aéreo. O algoritmo proposto tem como objetivo ser executa em tempo-real e a bordo do UAV. Este tipo de planeamento continua a ser investigado, sendo o maior foco da comunidade científica uma solução capaz de fazer um planeamento quase em tempo real. Este algoritmo foi proposto em alguns artigos, como servem de exemplo os seguintes [11,56]. É importante salientar que este algoritmo se encontra disponível online[1], sendo possível a sua utilização em ROS.

[1] `https://github.com/ethz-asl/StructuralInspectionPlanner`

4.1 Proposta apresentada

No artigo [11] é apresentada uma solução para a inspeção de uma estrutura 3D. Esta estrutura encontra-se representada por uma *mesh* definida por um determinado número de triângulos e, para cada triângulo representado, vai existir um ponto de vista (*viewpoint*) da qual esse triângulo é visto. Após a obtenção desses pontos, será calculado um caminho que garanta a total inspeção do objeto 3D desejado.

De modo a simplificar o problema, o algoritmo encontra-se implementado para ser utilizado com *fixed-wing* e *quadcopter*, sendo que, tanto o *roll* como o *pitch* são considerados nulos nos pontos de interesse. O algoritmo assume também a existência de um sistema de visão com orientação fixa relativamente ao *frame* do UAV.

É importante salientar que este algoritmo não se preocupa com a utilização do menor número de *viewpoints* possível mas sim com a escolha do melhor *viewpoint* para cada triângulo da *mesh*. Para cada ponto, é gerada ainda a orientação do veículo para a visualização do triângulo em questão.

O algoritmo 1 apresenta uma visão geral de como é feito o planeamento da trajetória, sendo este algoritmo apresentado em [11].

Algoritmo 1 *Structural Inspection path planner*

```
Carregamento da mesh
k = 0
Gera viewpoints para triângulos da mesh
Gera caminho entre os viewpoints
Calcula matriz de custo
Resolve TSP para obter caminho inicial
while k<n_interaçőes do
    Redimensiona configuraçáo dos viewpoints
    Recalcula o caminho gerado
    Recalcula a matriz de custo
    Resolve TSP para atualizar o melhor caminho de inspeçáo
    k=k+1
end while
return MelhorCaminho, MelhorCusto
```

4.1.1 Identificação de *Viewpoints*

A primeira ação efetuada pelo algoritmo é o carregamento da *mesh*. A *mesh* a ser utilizada tem que ser uma *mesh* triangular.

De seguida, para cada triângulo da *mesh*, é retirado um *viewpoint*. Tendo em atenção que não se encontram otimizados, será necessário otimizar a posição de modo a que a distância ao *viewpoint* seguinte seja a menor possível. Após a otimização da posição, será feita a otimização do *heading*. De modo a garantir a fiabilidade desta otimização, são impostas algumas restrições.

A primeira restrição é aplicada na posição de modo a que seja possível encontrar a orientação para a qual o triângulo é completamente visível. Assim, g = *[x,y,z]* corresponde à posição do *viewpoint*, x_i corresponde aos vértices do triângulo, a_N corresponde à normal ao triângulo, e d_{min} e d_{max} correspondem à distância mínima e à distância máxima de inspeção. Esta restrição encontra-se descrita na equação 4.1 e na figura 4.1(a).

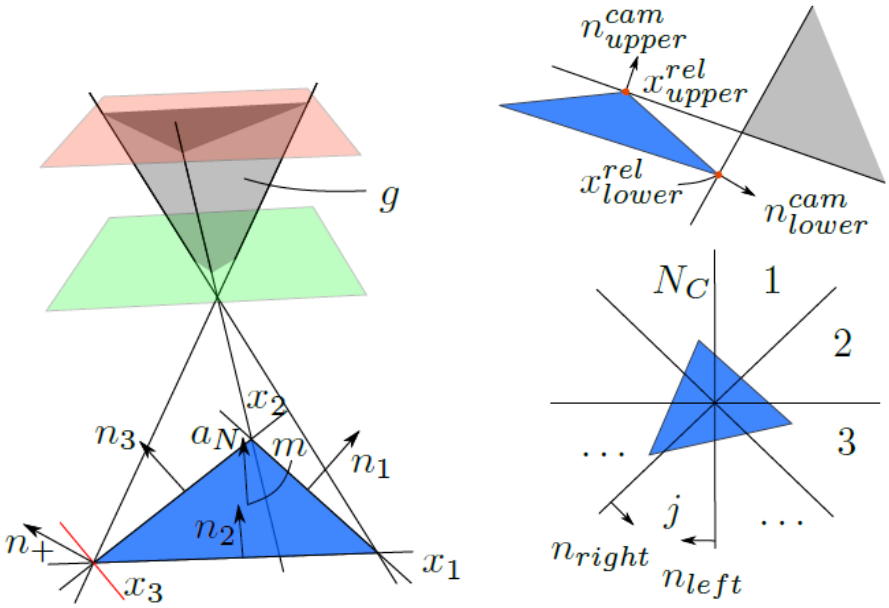
$$\begin{bmatrix} (g-x_i)^T n_i \\ (g-x_1)^T a_N \\ -(g-x_1)^T a_N \end{bmatrix} \geq \begin{bmatrix} 0 \\ d_{min} \\ -d_{max} \end{bmatrix}, i = \{1,2,3\} \tag{4.1}$$

A segunda restrição está relacionada com o campo de visão limitado (*Field-Of-View* (FOV)) apresentado pela câmara, onde é possível definir o ângulo de abertura horizontal e vertical.

Tendo em conta esta restrição, é importante referir que a lente da câmara é convexa. Para a altura da imagem da câmara, os triângulos podem não apresentar caraterísticas convexas, conduzindo à distorção do triângulo em questão. De modo a resolver este problema, o triângulo será então dividido em N_c pedaços iguais. A junção deste pedaços vai originar uma aproximação a um triângulo convexo para compensar o efeito convexo da lente.

No caso da largura da imagem da câmara, as restrições não se encontram definidas, sendo apenas considerado uma d_{min} que consiga ver o triângulo completo. Isto permite uma variação no que toca ao cálculo do *yaw*, sendo possível calcular essas restrições. Estas restrições podem ser analisadas na equação 4.2. Nesta equação, x^{rel}_{lower} e x^{rel}_{upper} correspondem aos vértices do triângulo na *mesh*, m corresponde ao ponto médio do triângulo e n^{cam}_{lower}, n^{cam}_{upper}, n_{right} e n_{left} correspondem à normal que separa os hiperplanos. Para uma melhor perceção desta restrição, a figura 4.1(b) representa a formação de um triângulo convexo a partir de um triângulo não convexo.

$$\begin{bmatrix} (g - x^{rel}_{lower})^T n^{cam}_{lower} \\ (g - x^{rel}_{upper})^T n^{cam}_{upper} \\ (g - m)^T n_{right} \\ (g - m)^T n_{left} \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.2}$$



(a) Restrição da posição de um triângulo (b) Restrição da câmara

Figura 4.1: Restrições do algoritmo [11]

Em cada iteração é necessário proceder à otimização dos *viewpoints* recalculados. Para tal, é necessário proceder à minimização da somas das distâncias quadradas ao ponto seguinte (g_p^{k-1}), ao ponto anterior (g_s^{k-1}) e ao ponto atual no caminho calculado anteriormente (g^{k-1}). Esta otimização encontra-se representada na equação 4.3.

$$min((g^k - g_p^{k-1})^T(g^k - g_p^{k-1}) + (g^k - g_s^{k-1})^T(g^k - g_s^{k-1}) + (g^k - g^{k-1})^T(g^k - g^{k-1})) \tag{4.3}$$

4.1.2 Cálculo do caminho

Na primeira iteração, é recolhida uma amostra que proporcione uma cobertura total da estrutura a ser inspecionada.

Para o planeamento do caminho são utilizados dois métodos. Entre dois pontos, é feito o cálculo do caminho recorrendo ao método BVS. No entanto, caso seja encontrado um obstáculo que se encontre posicionado entre os pontos a serem considerados, o método a ser utilizado passa a ser o RRT* [70]. Após este cálculo, e de modo a otimizar o caminho encontrado, é utilizado o método LKH [71].

4.1.3 Cálculo da matriz de custo

De modo a obter o menor custo possível, é retirada uma nova amostra em cada iteração do algoritmo, sendo calculado o melhor caminho para cada amostra recolhida.

Esta inspeção deve ser efetuada tendo um menor custo possível (no que toca a distância a ser percorrida entre pontos ou o tempo gasto durante a viagem), garantindo que, caso exista um sistema de perceção a bordo do robô (como uma câmara ou um *Light Detection And Ranging* (LiDAR)), este seja capaz de cobrir toda a área da estrutura cuja inspeção é desejada.

Para o cálculo do custo, é necessário ter em atenção que o algoritmo apenas impõe limite à velocidade do quadcopter. Assim, a velocidade de translação máxima é denominada por v_{max} e a velocidade de rotação máxima é denominada de $\dot{\psi}_{max}$. O tempo de execução (t_{ex})) da equação 4.4 vai corresponder ao custo para a execução do caminho planeado entre dois pontos, correspondendo d à distância Euclidiana entre esses dois pontos. O custo total vai corresponder então ao somatório de todos os tempos de execução calculados.

$$t_{ex} = max(d/v_{max}, ||\psi_1 - \psi_0||/\dot{\psi}_{max}) \tag{4.4}$$

4.2 Resultados apresentados pelo algoritmo

Tal como referido anteriormente, este algoritmo encontra-se disponível no GitHub. Após a sua instalação, foi possível verificar a existência de alguns exemplos disponibilizados pelos autores do trabalho e que foram apresentados no artigo publicado em [11].

De seguida, serão expostos dois dos exemplos disponíveis deste algoritmo.

4.2.1 BigBen

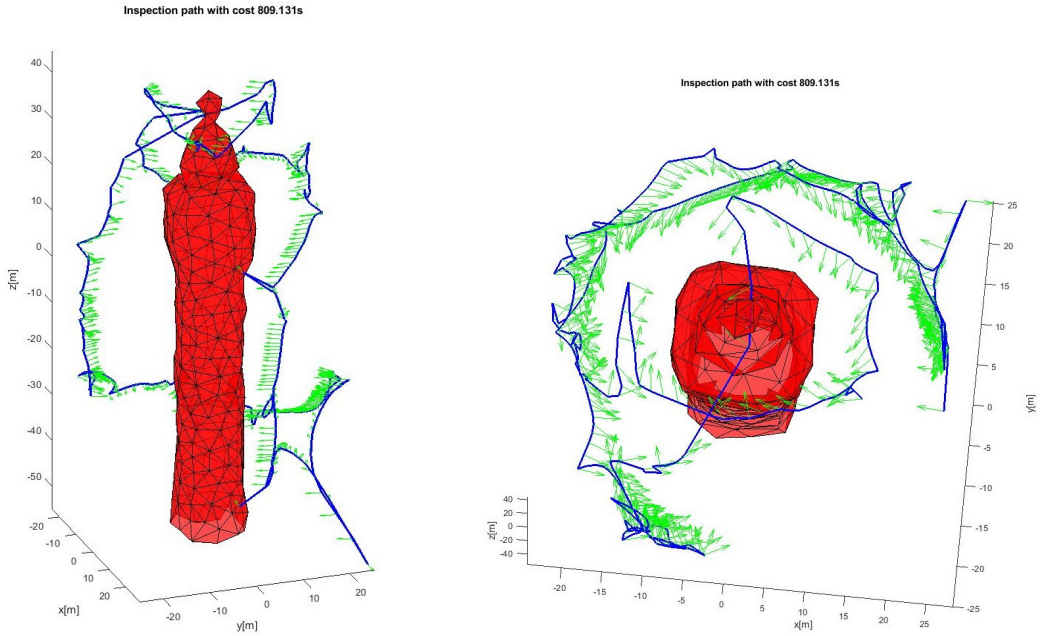
O primeiro exemplo a ser executado foi o Big Ben, o famoso relógio que se encontra em Londres. Na tabela 4.1 encontram-se representados os parâmetros que podem ser alterados pelo utilizador. Para além desses parâmetros, é importante salientar que a

posição inicial e a posição final tem de coordenadas [25;25;-55] e o número de iterações é de 20.

Tabela 4.1: Parâmetros para *mesh* Big Ben

nº triângulos	526
FOV	[120;120]
∠ incidência	30º
Pitch	5º
d_{min}	10.0
d_{max}	50.0
v_{max}	1.0
$\dot{\psi}_{max}$	0.5

Em relação ao caminho calculado, a figura 4.2 corresponde ao caminho dado como sendo o de menor custo pelo algoritmo em questão, para a inspeção do Big Ben. O custo determinado para esta inspeção é de 809.131 s.



(a) Vista lateral do caminho calculado

(b) Vista superior do caminho calculado

Figura 4.2: Visualização da *mesh* Big Ben com com a trajetória calculada pelo algoritmo de diferentes vistas.

4.2.2 Estátua Hoa Hakananai'a

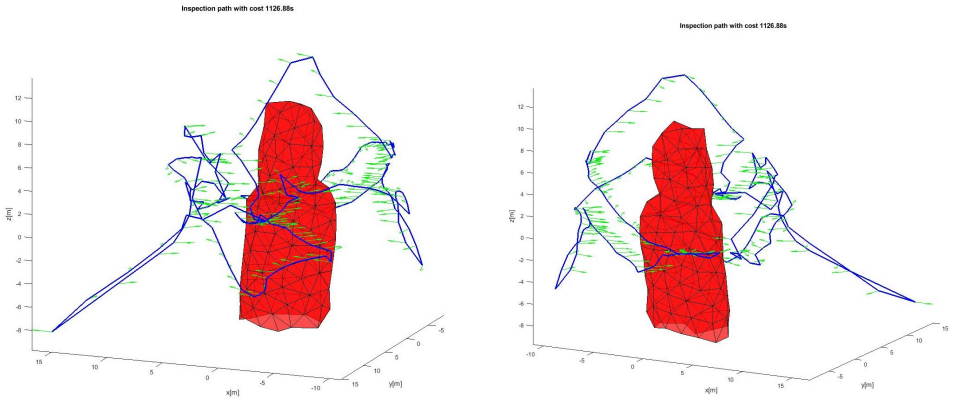
O segundo exemplo corresponde ao cálculo da trajetória ótima para a inspeção de inspeção da estátua Hoa Hakananai'a. Esta estátua é uma representação de um Moai, uma conhecida estátua gigante de pedra que se encontra na Ilha de Páscoa, no Chile.

No que toca a parâmetros, estes são ligeiramente diferentes daqueles utilizados para a inspeção do Big Ben, nomeadamente o valor de *pitch*, a distância mínima e máxima de inspeção, e e o valor definido para a velocidade máxima. Em relação à sua posição inicial e final, esta corresponde a [15;15;-8.0] e o número de iterações permanece igual, sendo este 20. A tabela 4.2 permite uma melhor visualização dos valores definidos para que a inspeção seja o mais otimizável possível possível.

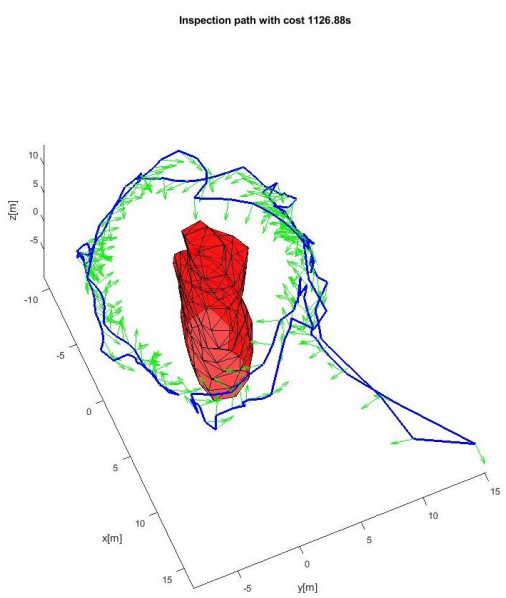
Tabela 4.2: Parâmetros para *mesh* Hoa Hakananai'a

nº triângulos	225
FOV	[120;120]
∠ incidência	30º
Pitch	25º
d_{min}	4.0
d_{max}	8.0
v_{max}	0.25
$\dot{\psi}_{max}$	0.5

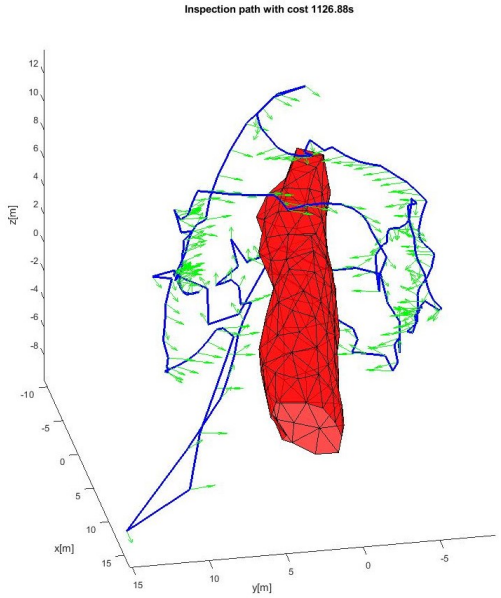
Após a conclusão da execução do algoritmo, executando o ficheiro criado onde contém a trajetória calculada encontrado obtém-se as figuras representadas na figura 4.3, onde é possível observar diferentes vistas do caminho de inspeção em relação à *mesh* utilizada. O custo determinado para esta inspeção é de 1126.88 s.



(a) Vista frontal do caminho calculado (b) Vista traseira do caminho calculado



(c) Vista superior do caminho calculado



(d) Vista inferior do caminho calculado

Figura 4.3: Representação da *mesh* Hoa Hakananai'a com caminho de inspeção calculado.

4.3 Resultados obtidos para o caso em estudo

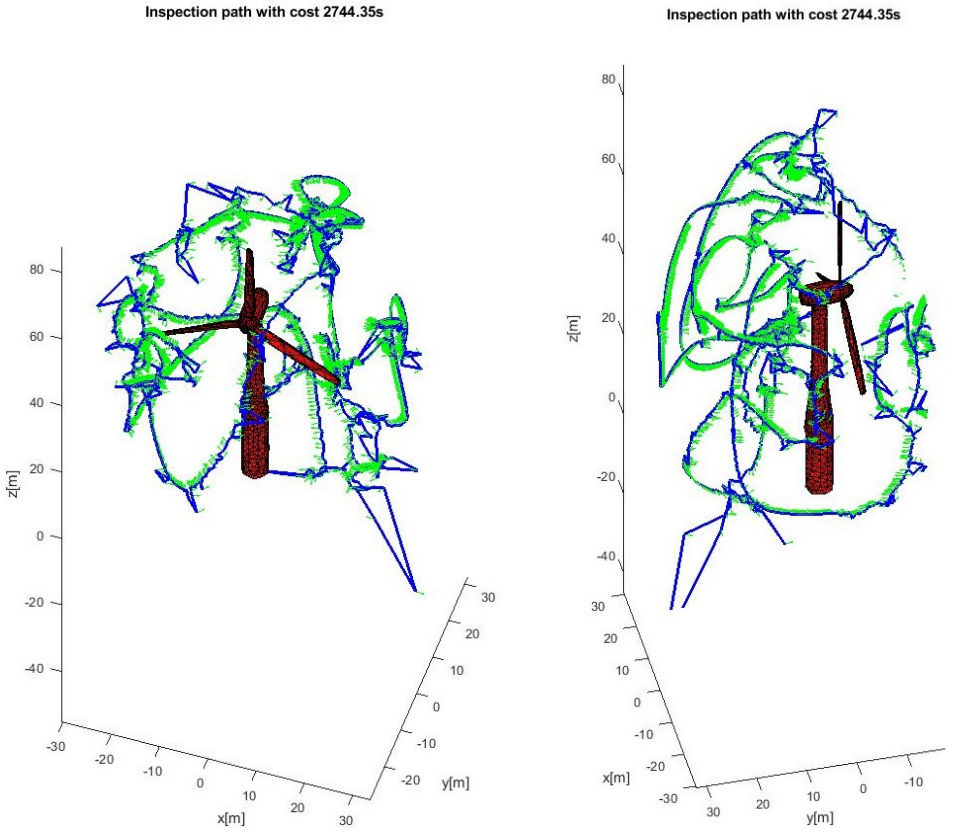
De modo a validar o algoritmo descrito em [11] com uma eólica, foi criada uma *mesh* triangular de uma eólica. Para a criação da mesh foi necessário ter em atenção o número de triângulos a serem colocados na *mesh*. Caso o algoritmo não consiga computar um *viewpoint* para um determinado triângulo, o algoritmo não será capaz de prosseguir com os seus cálculos. Nesse caso, o triângulo em questão terá que ser removido da *mesh*.

De modo a ser possível a sua comparação com os exemplos fornecidos pelo algoritmo, os parâmetros utilizados foram iguais aos parâmetros utilizados para o caso do Big Ben, sendo a única diferença entre as duas *meshes* o número de triângulos que cada uma delas contém. Esses parâmetros encontram-se representados na tabela 4.3.

Tabela 4.3: Parâmetros para *mesh* Eólica

nº triângulos	3808
FOV	[120;120]
∠ incidência	30º
Pitch	5º
d_{min}	10.0
d_{max}	50.0
v_{max}	1.0
$\dot{\psi}_{max}$	0.5

A trajetória gerada, este encontra-se representado na figura 4.4, onde é possível observar o resultados através de diferentes perspetivas do mesmo. O custo determinado para esta inspeção é de 2744.3 s.



(a) Vista superior do caminho calculado

(b) Vista inferior do caminho calculado

Figura 4.4: Visualização do caminho calculado pelo algoritmo de diferentes perspetivas.

Após este primeiro resultado, foram realizados mais alguns testes, onde se alteraram alguns parâmetros como, por exemplo, o FOV, distância mínima e máxima, e velocidade máxima. Os resultados obtidos estão detalhados na tabela 4.4.

Na eólica, a área de interesse de inspeção corresponde às pás, à nacelle e ao rotor. Nestes testes, não foi considerada a inspeção de apenas a área de interesse da eólica. Essa aplicação será abordada no capítulo de resultados.

4.3.1 Discussão dos resultados obtidos

Após terminar a execução, o algoritmo indica o tempo de algumas das ações importantes para a obtenção do melhor caminho, nomeadamente o tempo total do cálculo da inspeção, o tempo do LKH, o tempo despendido para o cálculo do RRT*, o tempo dedicado às distâncias calculadas e o tempo para a recolha dos *viewpoints*. Assim, em termos de comparação direta considerando a *mesh* Big Ben e a Eólica, foi possível concluir que a maior complexidade da *mesh* traduz-se num maior tempo despendido para gerar o melhor caminho de inspeção, tal como se pode observar na tabela 4.4. Apesar de nesta tabela constarem os tempos conseguidos durante a execução da *mesh* Hoa Hakananai'a, estes de nada servem para as comparações efetuadas.

Tabela 4.4: Principais diferenças entre *mesh* Big Ben e *mesh* Eólica

	Big Ben	Eólica	Hoa Hakananai'a
Dimensões [x;y;z] (m)	[20;20;60]	[15;60;100]	[8;4;20]
nº triângulos	526	3808	225
Tempo de cálculo (ms)	33057	416533	9222
Tempo de cálculo método LKH (ms)	28510	367075	7892
Tempo de cálculo método RRT* (ms)	5	36	1
Tempo de cálculo de distâncias (ms)	233	20698	48
Tempo de recolha de *viewpoints* (ms)	4026	20378	1212

Após os testes efetuados, existem alguns aspetos que devem ser mencionados:

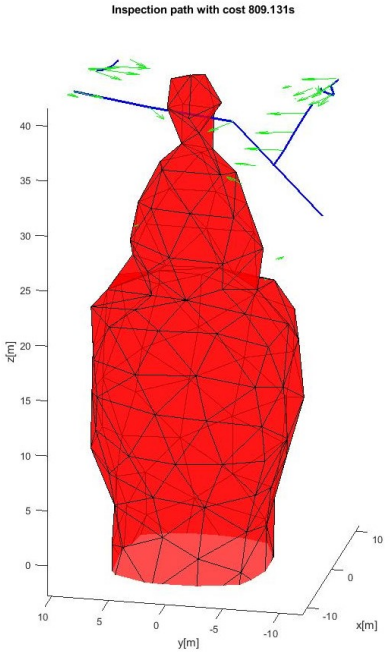
- É necessário ter atenção ao número de triângulos a utilizar, sendo que o algoritmo requer um número mínimo de triângulos. Para além disso, um número excessivo de triângulos traduz-se num maior tempo de processamento por parte do algoritmo.

- As *meshes* utilizadas nos exemplos dados pelo algoritmo não apresentam muito detalhe. Neste caso, as *meshes* apresentam, com a devida escala, as dimensões dos objetos reais, no entanto existem pormenores que são visivelmente omitidos. Com este aspeto, e comparativamente ao que aconteceu no caso da eólica, é possível

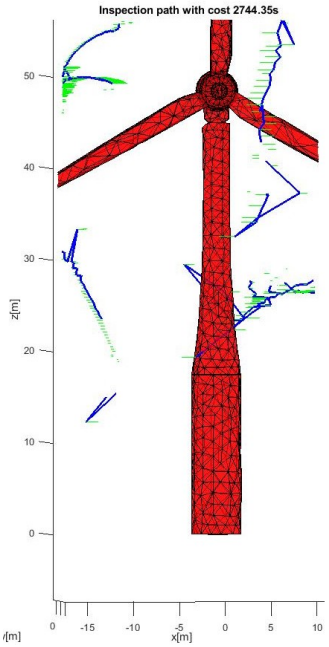
concluir que o algoritmo se comporta melhor quando a *mesh* a ser inspecionada apresenta o menor detalhe possível.

- O algoritmo apresenta uma pequena falha no que toca ao *obstacle avoidance*. Apesar de em [11] ser referido que é utilizado o método RRT* para evitar a colisão com obstáculos, nos exemplos apresentados anteriormente foi possível constatar que podem existir ligações entre pontos que atravessam a *mesh* a ser inspecionada, tal como se pode verificar na figura 4.5.

- No caso da eólica, existe o cálculo de inspeção em zonas onde não existe interesse de inspeção. As zonas de interesse são consideradas como sendo a parte superior da eólica, nomeadamente a nacelle, o rotor e as pás. Observando a figura 4.5(a), é notório o excesso de caminho criado na zona superior, tanto na parte traseira como dianteira da eólica. Nestas duas zonas, o caminho criado deveria estar mais distribuído e não tão concentrado àquelas duas regiões.



(a) Vista aproximada durante inspeção da *mesh* Big Ben, correspondendo ao topo do Big Ben



(b) Vista aproximada durante inspeção da *mesh* Eólica, correspondendo à base da eólica

Figura 4.5: Exemplos de situações onde se verifica falhas no algoritmo

Com esta análise, foi possível concluir que este algoritmo é indicado para a inspeção da eólica. Apesar do tempo de processamento levado a cabo por este algoritmo e do resultado obtido, é necessário referir que foi efetuada a inspeção da *mesh* completa da eólica. De modo a analisar de uma forma mais precisa o comportamento do algoritmo no caso da eólica, seria importante restringir a inspeção apenas para a área de interesse, as pás.

Para além disso, é importante referir que este algoritmo apresenta uma grande vantagem no que toca aos métodos de inspeção, uma vez que permite o cálculo de um caminho para uma possível inspeção utilizando apenas uma *mesh* para tal. Não é descartada a possibilidade de existirem avanços na resolução destes tipo de problemas, no entanto apenas este algoritmo se encontra disponível para os utilizadores.

Capítulo 5

Projeto

Neste capítulo serão abordados os diferentes elementos necessários para o desenvolvimento de uma solução. Serão apresentados diferentes soluções e efetuadas as respetivas escolhas, devidamente fundamentadas.

5.1 VTOL

De modo a dar resposta ao principal problema durante uma inspeção, foi estudada um possível desenvolvimento de um veículo híbrido. Assim, na disciplina *Laboratório de Sistemas Multirobóticos* (LASMU) de mestrado foram estudadas diferentes *frames* para o veículo e foi efetuado o levantamento de todos os componentes necessários para o seu bom funcionamento. Tendo em conta que era uma disciplina de mestrado, é necessário referir que este trabalho foi feito em grupo, com a colaboração do colega Rui Mendes.

Tendo em conta a dimensão deste projeto, o mesmo foi dividido em dois, sendo que nesta dissertação se abordará a parte eletrónica presente no VTOL. A figura 5.1 representa a arquitetura de alto nível do sistema, estando a cor de laranja representados os componentes estudados pela autora desta dissertação.

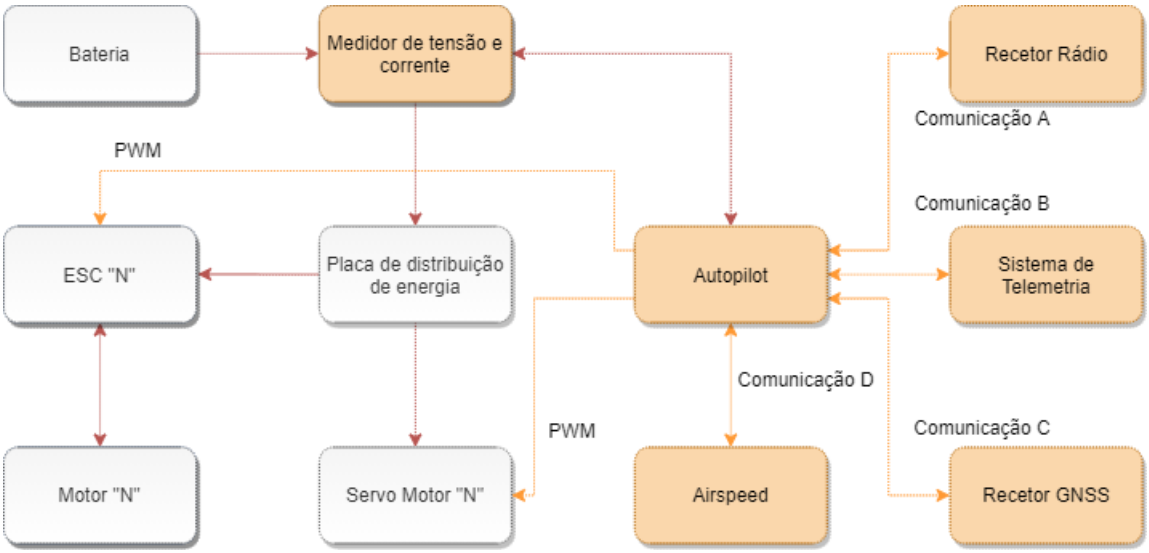


Figura 5.1: Arquitetura do sistema

5.1.1 *Autopilot*

Um *autopilot* é um controlador de baixo nível utilizado para o controlo de voo de um UAV. Para a escolha deste componente, foi necessário ter em consideração o facto de existir uma *ground station* e a necessidade da mesma comunicar com o UAV. Para que esta comunicação seja possível, é necessário que o *autopilot* suporte o protocolo de comunicação MAVLink, para que seja possível a comunicação do veículo com a *ground station*. Assim, encontraram-se dois controladores que poderiam ser aplicados, nomeadamente o Pixhawk e o APM 2.8.

O Pixhawk 1 é um controlador de voo baseado no Pixhawk-project FMUv2 com um desenho de *hardware* aberto, desenvolvido inicialmente pela 3DR. Esta placa corre PX4 no NuttX OS. Era originalmente uma plataforma *standard* para correr o PX4. No entanto, esta placa deixou de ser fabricada pela 3DR, passando a ser utilizada a mRo Pixhawk como alternativa [72]. A placa mRo Pixwahk baseia-se no Pixhawk-project FMUv3 que veio corrigir um erro que limitava o Pixhawk 1 a 1MB de memória flash [73].

(a) Pixhawk 1, desenvolvido por 3DR [72]

(b) mRo Pixhawk [73]

Figura 5.2: Pixhawk

O Pixhawk 1 é composto por um STM32F427 180 MHz ARM Cortex M4 que permite o controlo do sistema principal. Para além disso, possui 256 KB de RAM, 1 MB de flash, slot de cartão microSD e permite ligação de GPS que funcione com o protocolo U-Blox 7/8 ou U-Blox 6. Suporta conexões de *Inter-Integrated Circuit* (I2C), *Controller Area Network* (CAN), *Analog Digital Converter* (ADC), *Universal Asynchronous Receiver-Transmitter* (UART), *Pulse Width Modulation* (PWM), *General Purpose Input/Output* (GPIO), entre outras.

Para além do Pixhawk, ponderou-se a utilização do APM 2.8. O APM 2.8 é um *autopilot* baseado no Arduino Mega, da ArduPilot. Este *autopilot* é capaz de controlar aviões de asa fixa, helicópteros, VTOL, entre outros. Em termos de *firmware*, tal como o PX4, é *open source*, encontrando-se em constante desenvolvimento [12].

Figura 5.3: APM 2.8 [12]

O APM 2.8 possui 4 MB de *dataflash* para o registo de dados, contém um giroscópio

de 3 eixos, um acelerómetro e um barómetro de alta performance. É possível existir uma ligação externa a um GPS que funcione com o protocolo U-Blox. Para o processamento inclui um Atmel's ATMEGA2560 e para o controlo do *Universal Serial Bus* (USB) possui um ATMEGA32U [74].

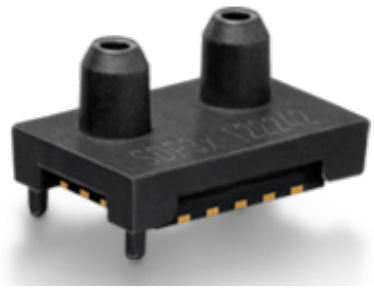
5.1.2 *Airspeed*

O *airspeed* é um sensor diferencial, utilizado em aviões e UAVs, que permite medir a velocidade do avião em relação ao ar. Este tipo de sensores pode ser analógico ou digital.

Os sensores analógicos são sensores diferenciais, medindo a velocidade do UAV recorrendo à diferença entre o ar presente no tubo e o ar que entra pelo *pitot*. Este tipo de sensores efetuam a conversão de um sinal analógico para digital. Este tipo de sensores tem como principal vantagem o facto de ser compatível com qualquer *autopilot*. No entanto, as medidas podem apresentar algum ruído dada a conversão que necessitam de efetuar.

Os sensores digitais tem como princípio de funcionamento o mesmo que o dos sensores analógicos. Para estes sensores, o protocolo de comunicação pode ser I2C, *Serial Peripheral Interface* (SPI) ou CAN. As principais vantagens na utilização destes sensores reside na sua precisão. No entanto, nem todos os *autopilots* suportam a sua utilização.

As figuras 5.4(a) e 5.4(b) contém dois possíveis sensores a serem utilizados na implementação pretendida, sendo um deles analógico e o outro digital.



(a) Sensor analógico Sensirion SDP3X [75]

(b) Sensor digital Drotek [76]

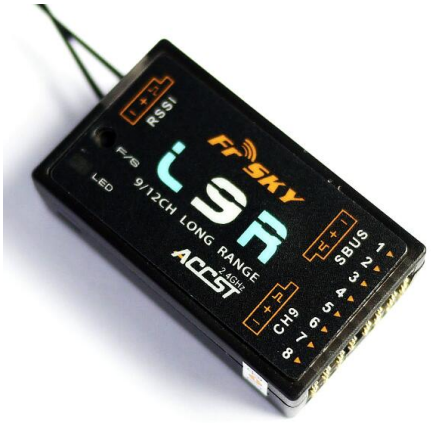
Figura 5.4: Diferentes sensores *airspeed*

5.1.3 Recetor Rádio

O rádio link consiste na comunicação rádio entre o VTOL e o comando, permitindo assim o controlo das atitudes do VTOL. Assim, é necessária a presença de um módulo recetor no VTOL e outro módulo transmissor presente no comando.

Para esta comunicação, o material possível de ser utilizado já se encontrava no laboratório. Esse material é o Frsky XJT e L9R, que permitem uma comunicação segura na banda dos 2.4 GHz.

(a) Módulo acoplado ao comando, Frsky XJT [77]



(b) Módulo presente no VTOL, Frsky L9R [78]

Figura 5.5: Recetor rádio

5.1.4 Sistema de Telemetria

A telemetria, no caso do presente projeto, é importante pois permite visualizar numa *ground station* os dados de voo do VTOL. Para tal, é necessário o acoplamento de um módulo no VTOL e de um módulo que permite a ligação, por USB, ao computador utilizado.

No caso da telemetria, o material já se encontrava no laboratório, sendo esse material um módulo genérico da Hobbyking 433 MHz.

Figura 5.6: Módulo telemetria para ligação ao computador [13]

5.1.5 Recetor GNSS

O recetor de *Global Navigation Satellite System* (GNSS) é utilizado para a obtenção de uma localização.

Para a obtenção do sinal já existiam módulos presentes no laboratório. Esse módulo é o Ubx M8T, que permite a receção de diferentes tipos de constelações, como serve de exemplo o GPS, o GALILEO, entre outros.

Figura 5.7: Recetor de GNSS [14]

5.1.6 Medidor de tensão e corrente

De modo a garantir a alimentação correta do *autopilot*, foi necessária a procura de um módulo capaz de ler a tensão e corrente da bateria, fornecendo adicionalmente tensão

para o *autopilot*. Neste caso, foi necessário verificar o tipo de bateria a ser utilizada, o máximo de corrente suportada e ainda a tensão à qual deve ser alimentado o *autopilot*.

Tal como aconteceu am alguns tópicos anteriores, este módulo já se encontrava no laboratório, tendo sido verificado se o seu enquadramento no projeto seria possível.

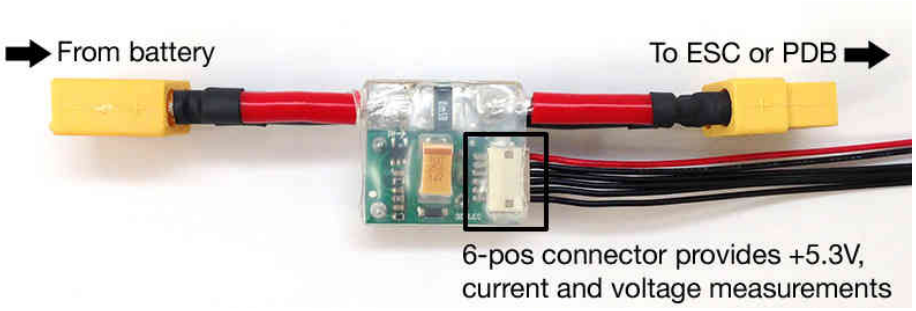


Figura 5.8: Medidor de tensão e corrente [15]

5.2 Firmware

5.2.1 Ardupilot

O Ardupilot é um *autopilot* criado em 2009 utilizado para a utilização de robôs autónomos em situações do quotidiano. É um *firmware open source* em constante desenvolvimento pela comunidade. Possui fóruns[1] onde é possível tirar dúvidas e pedir ajuda aos membros da comunidade dedicados ao desenvolvimento deste *firmware*. Não apresentam nenhum *hardware* específico, sendo possível a utilização deste *firmware* numa grande variedade de *autopilots*.

É capaz de controlar qualquer veículo, apresentando quatro *firmwares* distintos:

- **ArduPlane**: *firmware* desenvolvido para o controlo de *fixed wing*. Este é também o *firmware* utilizado para o controlo de um VTOL.
- **ArduCopter**: *firmware* desenvolvido para o controlo de multirotors e helicópteros.
- **ArduRover**: *firmware* desenvolvido para o controlo de veículos terrestres e barcos.
- **ArduSub**: *firmware* desenvolvido para o controlo de ROV e veículos subaquáticos.

[1] https://discuss.ardupilot.org/

Este *firmware* torna possível a utilização de uma *ground station* que se encontrará em comunicação, em tempo real, com o UAV. Esta utilização é possível devido à implementação de MAVLink e mavros neste *firmware*. A *ground station* a ser utilizada poderá ser o Mission Planner (figura 5.9), desenvolvida especialmente para ser utilizada com o Ardupilot, ou QGroundControl. Nestas aplicações é possível visualizar informação importante sobre o UAV tal como a sua velocidade, a sua altitude, o seu *roll*, *pitch* e *yaw*, definir uma missão de *waypoints* utilizando um mapa. Para além disso, é possível armar e desarmar o UAV, efetuar *takeoff* e *land* e alterar o modo de voo do veículo. É permitido ainda guardar os *logs* das missões efetuadas pelo UAV e proceder à sua análise caso necessário.

Figura 5.9: MissionPlanner

Em Ardupilot é possível ainda efetuar simulações, uma vez que este *firmware* contém ambiente de simulação em *Software In The Loop* (SITL) e em Hardware In The Loop (HITL). Em termos de controlo do UAV, estas simulações são válidas, no entanto este *firmware* não apresenta integração direta com o Gazebo.

Em relação a calibrações, o Ardupilot contém a calibração automática de PID, denominado de Autotune. Para tal, é necessário que o veículo seja colocado a voar, mantendo a altitude. Dependendo do eixo a calibrar, o veículo vai efetuar algumas manobras de modo a descobrir quais os valores da malha PID seriam os mais indicados. No final do

processo, o utilizador pode escolher manter ou não os valores calculados pelo Autotune.

Este *firmware* apresenta diferentes modos de voo, sendo o modo de interesse o *Guided Mode*. Este modo permite que o veículo navegue até um determinado ponto, sem ser necessário definir uma missão, permitindo assim o envio de comandos de MAVLink alterando instantaneamente o comportamento do veículo. Se a *ground station* definir o ponto para o qual o veículo se deve deslocar, o mesmo irá efetuar o voo até ao ponto pretendido, ficando a fazer um círculo com um determinado raio em torno do ponto (*loiter*).

Tendo em atenção o caso específico desta dissertação que reside na utilização de um VTOL, o *firmware* mais indicado a ser utilizado seria o ArduPlane. Com a utilização deste firmware, é possível utilizar comando MAVLink, dos quais se destacam os seguintes:

- **MAV_CMD_NAV_VTOL_TAKEOFF**: comando que permite efetuar *takeoff*, sendo este em modo drone.
- **MAV_CMD_NAV_VTOL_LAND**: comando que permite a aterragem do VTOL em modo drone.
- **MAV_CMD_NAV_WAYPOINT**: este é o único comando de navegação suportado pelon ArduPlane. O veículo irá voar para a latitude, longitude e altitude especificada. O *firmware* considera que o veículo chegou ao seu destino quando o veículo se aproximar do ponto, a um determinado raio.
- **MAV_CMD_DO_VTOL_TRANSITION**: comando que permite efetuar transições para *fixed wing* e multirotor. É necessário estar em modo AUTO para poder ser utilizado.

Existe uma lista com todos os comandos MAVLink suportados pelo *firmware*, estando a mesma disponível online[2].

5.2.2 PX4

O PX4 é um *fimware*, *open source*, que pode ser utilizado numa grande variedade de robôs, desde robôs aéreos, subaquáticos e terrestres. Encontra-se em desenvolvimento ativo por parte da comunidade, existindo um fórum[3] para que os programadores possam colocar as suas dúvidas, indicar falhas no *firmware* e ainda pedir ajuda na resolução

[2]https://ardupilot.org/dev/docs/plane-commands-in-guided-mode.html
[3]https://discuss.px4.io/

de problemas. Este *firmware* pode ser utilizado em diversos *autopilot*, sendo o mais conhecido o Pixhawk.

Tal como o Ardupilot, com a utilização do *firmware* PX4 é possível utilizar uma *ground station*, como por exemplo o QGroundControl. Para que esta funcionalidade seja possível, este *firmware* possui o protocolo de comunicação MAVLink. Nesta aplicação é possível controlar parâmetros como a velocidade, as coordenadas de GPS do veículo, distância ao ponto inicial, entre outros. É também possível alterar parâmetros *default* do veículo como, por exemplo, o raio da circunferência em modo *loiter*, efetuar o *download* do *log* do voo efetuado e ainda analisar esse mesmo *log*.

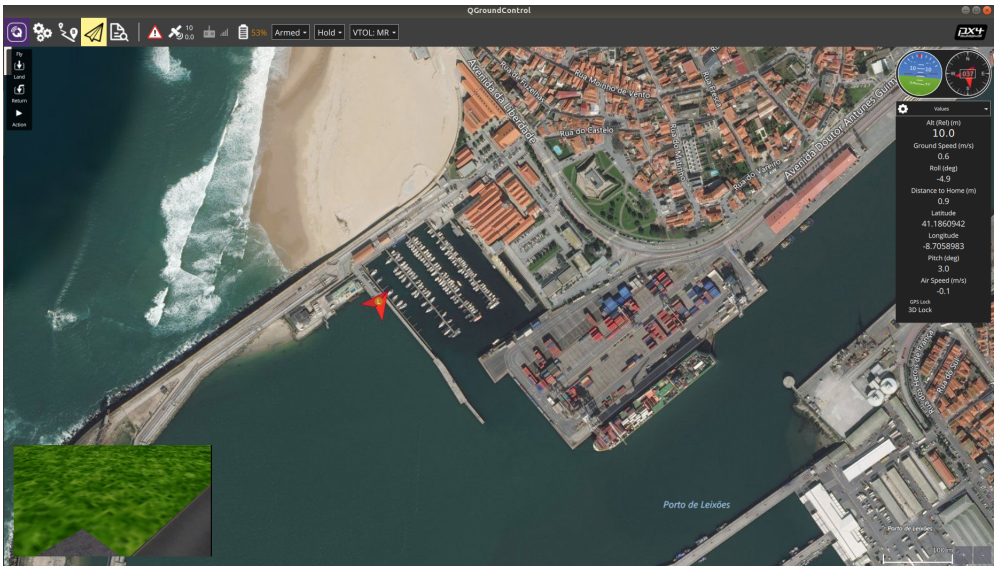


Figura 5.10: QGroundControl

Este *firmware* apresenta a possibilidade de serem efetuadas simulações, podendo estas ser SITL ou HITL. Tem ainda a vantagem de possuir integração direta com o Gazebo, sendo possível o utilizador visualizar os comportamentos do veículo durante a simulação, para além da informação mostrada no QGroundControl.

Em semelhança ao Ardupilot, este *firmware* possui também diversos modos de voo sendo o modo *Offboard* o de interesse. Este modo permite o controlo do movimento e atitude do veículo, sendo o equivalente ao *Guided Mode* do Ardupilot. Este modo requer uma conexão ativa a uma *ground station*. Caso esta conexão falhe, o veículo irá aterrar ou executar outro protocolo de segurança, sendo que este pode ser alterado pelo utilizador.

Neste *firmware* é possível a utilização de uma vasta gama de comandos MAVLink,

servindo de exemplo os comandos utilizados para o controlo em posição que poderá ser `SET_POSITION_TARGET_LOCAL_NED, SET_POSITION_TARGET_GLOBAL_INT` ou ainda `SET_ATTITUDE_TARGET`, dependendo se o utilizador pretende utilizar coordenadas globais ou coordenadas locais. Na página online do PX4 é possível encontrar alguns comandos que são suportados por este *firmware*.

5.2.3 Comparação Ardupilot e PX4

De modo a perceber qual o melhor *firmware* a ser utilizado nesta dissertação, foram estudados, instalados e testados ambos os *firmwares* mencionados anteriormente. Este estudo foi realizado na unidade curricular LASMU, em parceria com o colega Rui Mendes.

Para este estudo foi considerada a necessidade de efetuar takeoff/land, um controlo em posição, transições entre *fixed wing* e multirotor, e ainda permanecer no modo OFFBOARD/GUIDED durante o seu percurso. Assim, após alguns testes, foi possível verificar que existem diferenças entre os dois *firmwares*, estando os mesmos descritos na tabela 5.1.

Tabela 5.1: Diferenças entre PX4 e Ardupilot

PX4	**Ardupilot**
Simulador para VTOL funcional	Simulador implementado por alunos do 1º ano de mestrado, necessitando de correções
Controlo em posição funcional, encontrando-se o controlo em velocidade em desenvolvimento	Controlo em posição funcional, não existindo implementação do controlo em velocidade
Suporta vários comandos de *offboard*	Suporta poucos comandos de *offboard*
Desenvolvimento ativo para VTOL	Desenvolvimento ativo apenas para *copter* e *plane* (exceto VTOL)
Sem Autotune	Autotune
Interface e processos de calibração mais simples e intuitivo. Apresenta uma melhor explicação, existindo mais documentação	Possui uma interface gráfica mais fraca, contendo processos de calibração mais rudimentares
Sem *bugs* na integração de sensores (conhecidos)	Apresenta *bugs* na integração de alguns sensores (ex: magnetómetro)
O laboratório possui *drivers* para diferentes sensores (ex: stim)	Não existem *drivers* desenvolvidas pelo laboratório para este *firmware*

De modo a melhorar a implementação de simulador em Gazebo, seria necessário efetuar algumas melhorias, nomeadamente correção dos coeficientes de simulação da dinâmica do UAV, desligar motores de *multicopter* no voo em *fixed wing* e ainda eliminar o efeito *bang bang* proveniente do controlador de *fixed wing*.

Dadas as diferenças entre os dois *fimwares*, a escolha é clara, tendo sido utilizado o PX4.

5.3 Manobra de Inspeção

Após o estudo do algoritmo [11] descrito no capitulo 4, e tendo como referência os resultados obtidos, foram estudadas duas abordagens diferentes para a inspeção de eólicas, que serão descritas nos subtópicos seguintes.

5.3.1 Manobra de Inspeção das pás de uma eólica

O primeiro ponto a ser considerado era o facto de apenas ser relevante a inspeção das pás, pois são estas que sofrem um desgaste mais acentuado e que comprometem o funcionamento da mesma. O segundo ponto a ser considerado foi o facto de se necessitar uma inspeção mais aproximada, focada o máximo possível nas pás, rotor e nacelle da eólica. Assim, tendo em conta que o algoritmo *Structural Inspection Path Planning* oferece uma inspeção mais afastada do ativo, foi pensada uma solução que consistia em criar retas de inspeção entre o rotor e cada pá (para a parte frontal da eólica) e retas entre a nacelle e as pás (para a parte traseira da eólica). Estas retas, quando executadas a uma certa distância da eólica, permitiriam fazer uma inspeção mais precisa das pás. A figura 5.11(a) representa das retas que seriam necessárias para proceder à inspeção da parte frontal da eólica. Por sua vez, a figura 5.11(b) representa as retas necessárias para proceder à inspeção da parte traseira da eólica.

(a) Protótipo de inspeção na parte da frente da ventoinha

(b) Protótipo de inspeção na traseira da ventoinha

Figura 5.11: Protótipo de inspeção

No entanto, continuaria a faltar uma visão lateral das pás, tendo sido necessário pensar numa solução para esse problema. De modo a dar resposta a essa questão, foi pensada a criação de semi circunferências, nas laterais das pás. Estas circunferências permitiriam fazer a mudança entre a inspeção frontal e a inspeção traseira da eólica, aproveitando a deslocação do UAV para observar zonas das pás que não apareceriam apenas com a utilização das retas. Estas circunferências encontram-se representadas na figura 5.12.

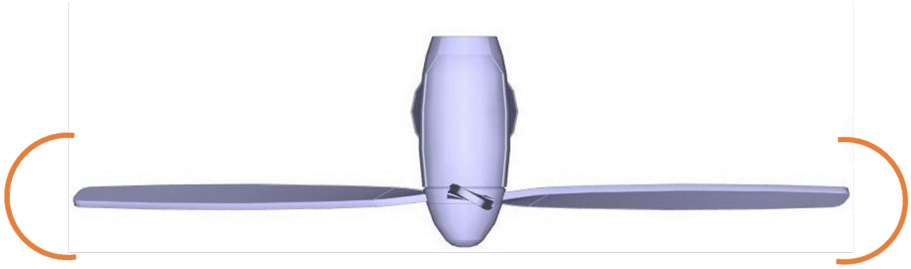


Figura 5.12: Protótipo de inspeção lateral

5.3.2 Manobra de *loiter*

Existem situações em que a inspeção não necessita de ser pormenorizada, sendo possível a passagem de um veículo apenas para verificar o aspeto geral da eólica. Este levantamento requer pouco tempo e não necessita de pormenor na informação que recolhe, podendo ser utilizada para efetuar um levantamento do número de eólicas que necessitariam de uma inspeção com mais detalhe.

Assim, de modo a dar resposta a este tipo de inspeções, foi projetada a solução de efetuar um *loiter* em torno da eólica, tal como indica a figura 5.13. Nesta manobra foram feitos círculos em torno da eólica, descendentes, a uma certa distância da mesma. Esta solução foi pensada para que seja utilizado apenas o modo *fixed wing*, aproveitando as vantagens deste veículo para tornar esta inspeção o mais rentável possível.

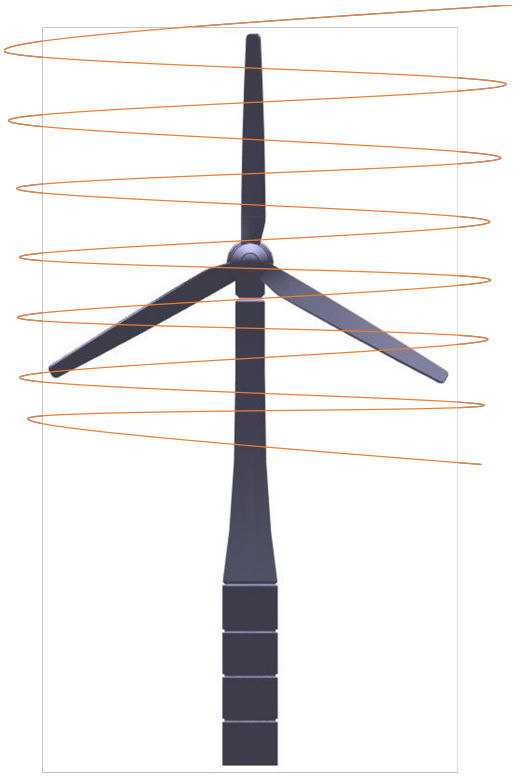


Figura 5.13: Protótipo de inspeção *loiter*

5.4 Resumo das decisões de projeto

No seguimento dos objetivos inicialmente definidos para o projeto e do estudo efetuado de cada componente (*hardware*/*software*) a ser desenvolvido, foram efetuadas as seguintes decisões:

Tendo em conta o estudo efetuado, o *autopilot* escolhido foi o Pixhawk 1. Tendo em conta a utilização deste controlador, foi necessária também a implementação de um botão de segurança para poder armar e desarmar o VTOL com segurança.

Relativamente ao *airspeed*, e de modo a facilitar a implementação deste sensor no projeto, decidiu-se utilizar um sensor digital para este efeito. Para além disso, é importante salientar que um sensor digital, em comparação com um analógico, é um sensor mais preciso, não sendo necessário proceder à sua calibração para obter resultados credíveis.

No que toca a telemetria, recetor rádio, recetor GNSS e o medidor de tensão e corrente, foram utilizados os componentes apresentados anteriormente tendo em atenção que estes já existiam em laboratório.

Assim, após a escolha de todos os componentes necessário para a construção do VTOL surgiu um novo esquema para a arquitetura de alto nível do projeto. Este novo esquema encontra-se representado na Figura 5.14.

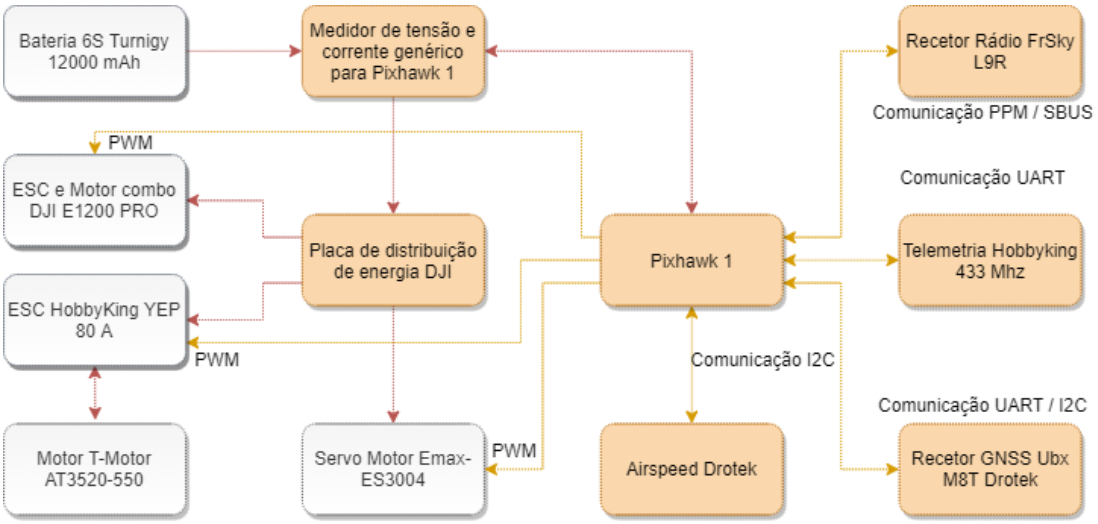


Figura 5.14: Arquitetura detalhada do sistema

Foi necessário também proceder à escolha do *firmware* a ser utilizado no *autopilot*. Tal como referido anteriormente, ambos os *firmware* foram instalados, testados e simulados de modo a ser possível obter as diferenças entre ambos. Inicialmente, os testes começaram com o Ardupilot uma vez que era um *firmware* que prometia constante desenvolvimento, sendo o mesmo muito utilizado em drones. No entanto, este desenvolvimento não se encontrou para o caso do VTOL. Este *firmware* encontra-se muito limitado ao que se consegue utilizar. As transições em VTOL apenas são permitidas em modo AUTO, os comandos de MAVLink suportados são relativamente baixos e não possui integração direta com o Gazebo. No que toca ao PX4, as coisas são diferentes. No desenvolvimento para VTOL existe uma maior diversidade de comandos MAVLink que podem ser utilizados e ainda existe integração direta com o Gazebo. Esta integração é fundamental pois permite ao utilizador perceber de forma mais rápida comportamentos atípicos do veículo. Após uma grande reflexão, acabou por se decidir que o melhor *firmware* a ser utilizado nesta dissertação seria o PX4.

Em relação às manobras de estudo, ambas merecem especial atenção uma vez que ambas são consideradas importantes para dar resposta ao problema principal desta dissertação. Para além destas duas manobras, existe ainda o algoritmo *Structural Inspection Path Planning* que será analizado para o caso de estudo em questão.

Esta página foi intencionalmente deixada em branco.

Capítulo 6

Implementação

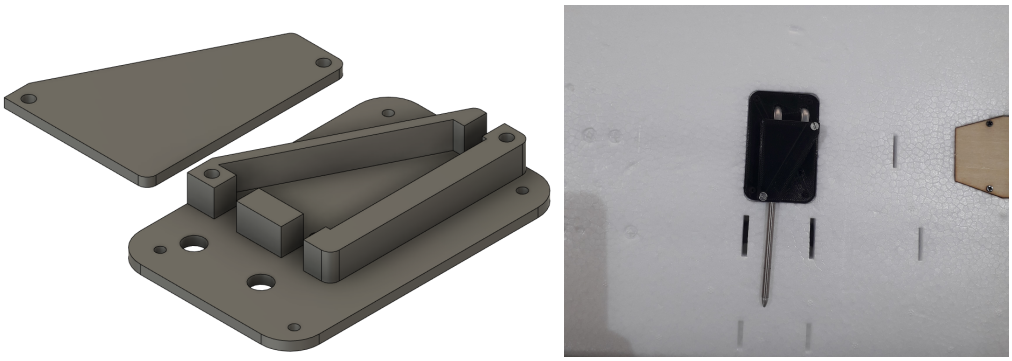
Este capítulo detalha os passos adotados para implementar as manobras descritas no capítulo anterior. Serão também abordados os blocos de *software* desenvolvidos e validados em Gazebo.

6.1 VTOL

6.1.1 Peça de suporte do *Airspeed*

O *Airspeed*, tal como evidenciado anteriormente, é um sensor diferencial que mede a velocidade do veículo em relação ao ar. Assim, este não pode ter interferências dos *propellers*, uma vez que causaria uma medida incorreta. Após análise dos túneis de ar gerados pela rotação dos 5 *propellers*, concluiu-se que o melhor posicionamento seria debaixo da asa.

De modo a ser possível acoplar este sensor à estrutura, foi necessário recorrer ao desenho tridimensional. A peça foi desenhada para encaixar num local que não seria utilizado, onde já existiam furos para se poder aparafusar esta nova peça. A Figura 6.1(a) representa o esquema da peça desenvolvida, estando representado na figura 6.1(b) a montagem deste sensor na estrutura do VTOL.

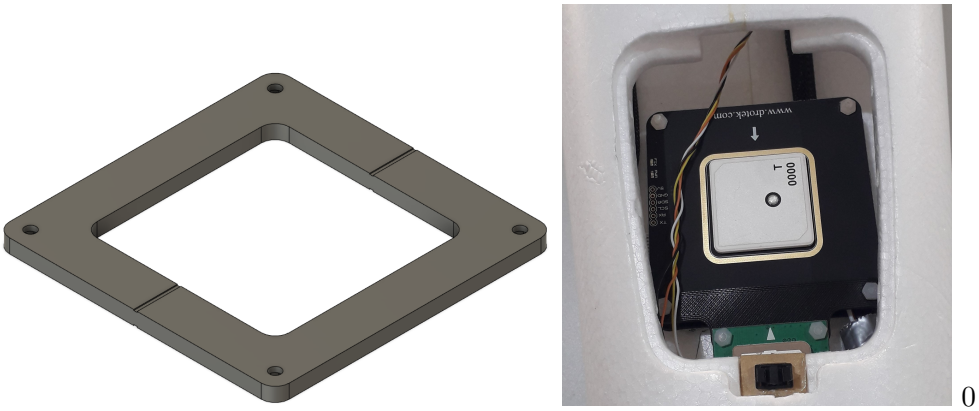


(a) Peça de suporte do *Airspeed* (b) Montagem do *airspeed* da estrutura

Figura 6.1: Peça desenhada em 3D e montagem da mesma na *frame* do VTOL

6.1.2 Suporte do recetor GNSS

A peça de suporte do recetor de GNSS, (Figura 6.2(a)), foi desenhada com o propósito de ser colada na *frame*. O recetor será colocado, com o auxílio de parafusos, a esta peça, sendo a figura 6.2(b) representante desta montagem. Isto confere segurança ao recetor e permite uma maior facilidade caso seja necessário remover o mesmo.



(a) Peça de suporte do recetor GNSS (b) Montagem do recetor GNSS na estrutura

Figura 6.2: Peça desenhada em 3D e montagem da mesma na *frame* do VTOL

6.1.3 Ligações Pixhawk

Após a colocação de todos os sensores nos respetivos sítios e da montagem integral da *frame*, foi necessário efetuar as ligações dos sensores ao *autopilot*. O sensor de corrente

e tensão foi ligado à entrada de POWER no Pixhawk. A telemetria ficou ligada à porta TELEM1, seguindo-se a ligação do recetor GNSS à entrada GPS. Tendo em conta que existiam dois sensores que utilizavam o protocolo de comunicação I2C, nomeadamente o *airspeed* e o magnetómetro, foi necessário ligar um *splitter* de I2C à entrada I2C do Pixhawk. Por último, foi necessário ligar o botão de segurança à entrada SWITCH do Pixhawk. Estas ligações encontram-se representadas na figura 6.3.

Figura 6.3: Demonstração das entradas utilizadas para a ligação dos sensores enunciados

Foi necessário também proceder à ligação do recetor rádio, motores e servo motores do veículo. Para estas ligações, o Pixhawk disponibiliza pinos na sua lateral. O recetor rádio foi ligado aos pinos RC. As superfícies de controlo de *fixed wing* foram ligados aos pinos MAIN OUT 1–4 e AUX OUT 1. Por sua vez, as superfícies de controlo de multirotor foram ligadas aos pinos MAIN OUT 5–8. A figura 6.4 demonstra as ligações enunciadas.

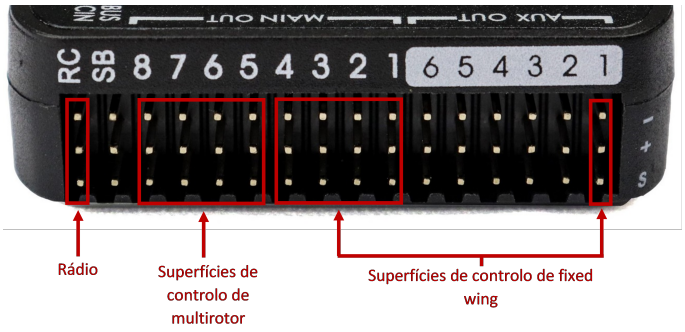


Figura 6.4: Demonstração dos pinos para ligação do sistema de controlo de *fixed wing* e multirotor

O resultado final de todas as ligações necessárias para o funcionamento do veículo podem ser analizadas na figura 6.5.

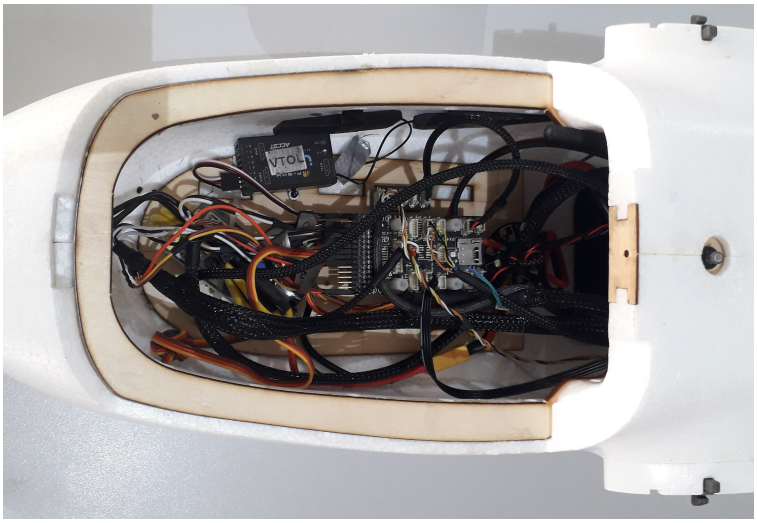


Figura 6.5: Ligações dos sensores ao *autopilot*

6.2 Algoritmo desenvolvido de Inspeção Autónoma de uma Eólica

Durante o desenvolvimento deste algoritmo foi tido em atenção o facto de ser necessário apresentar uma solução o mais generalizada possível, capaz de dar resposta a diferentes *meshes*. Assim, definiu-se que o utilizador necessita de saber algumas informações re-

lativamente ao ativo elétrico (eólica) que está a utilizar, nomeadamente as coordenadas do rotor e nacelle, o comprimento das pás, a distância a que a inspeção deverá ser feita, o número de pontos que pretende que sejam criados em cada reta e a distância entre o rotor e as pás. O algoritmo 2 apresenta uma visão geral de como é criada a manobra de inspeção.

Algoritmo 2 Wind Turbine Blades Inspection Algorithm

```
1: Set Parameters
2: Calculate wind turbine blade coordinates
3: Compute lines between nacelle/rotor and wind turbine blades
4: Calculate semi circles on the sides of the wind turbine blades
5: for all points do
6:    for all points do
7:       Compute distance between points
8:       if distance = 0 then
9:          Write inf on point distance
10:      end if
11:   end for
12:   Find minimum distance
13:   Write new point on position matrix
14: end for
15: Write information on file
```

6.2.1 Cálculo coordenadas das pás

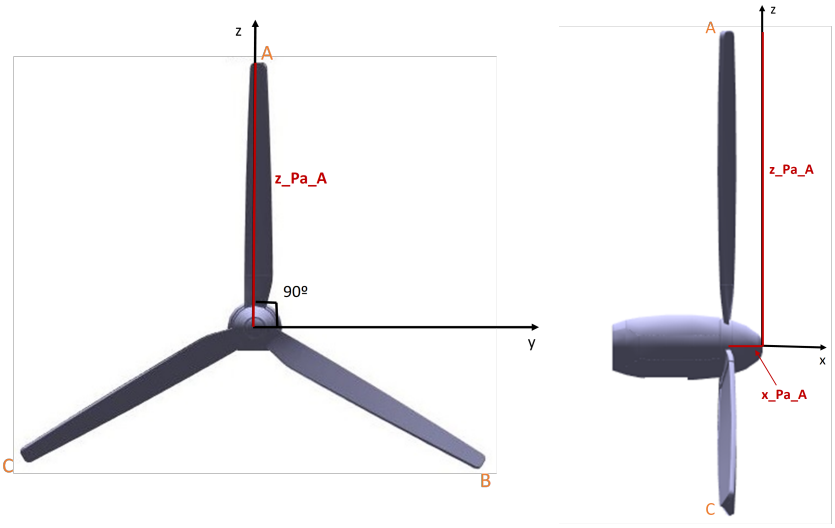
O primeiro cálculo efetuado pelo algoritmo corresponde ao cálculo das coordenadas das pás. As pás encontram-se desfasadas entre si em 120^{o}. Foi necessário então analisar cada pá individualmente. Este calculo tem em consideração que a eólica esteja com uma pá colocada a 90^{o} em relação ao rotor.

A pá A é considerado o caso mais simples para o cálculo das suas coordenadas. Esta pá encontra-se posicionada verticalmente em relação ao rotor. Neste caso, a coordenada y será igual à coordenada y do rotor. Em relação à coordenada z, a coordenada da pá será igual à coordenada z do rotor com a adição do comprimento da pá. No que toca a x, tendo em conta que existe um desfasamento entre as coordenadas do rotor e a posição da pá, a coordenada x da pá será igual à coordenada x do rotor sendo-lhe subtraída a distância entre a pá e o rotor.

De uma forma simplificada, as coordenadas da pá A serão:

$$\begin{cases} x_{pa\_A} = x_{rotor} - distancia_{pa\_rotor} \\ y_{pa\_A} = y_{rotor} \\ z_{pa\_A} = z_{rotor} + comprimento_{pas} \end{cases} \tag{6.1}$$

A figura 6.6 corresponde a uma representação esquemática da lógica aplicada para os cálculos, estando na figura 6.6(a) uma visão frontal desse esquema e na figura 6.6(b) uma visão lateral, assinalando a vermelho as coordenadas x e z da pá.



(a) Representação esquemática dos eixos y e z

(b) Representação esquemática dos eixos x e z

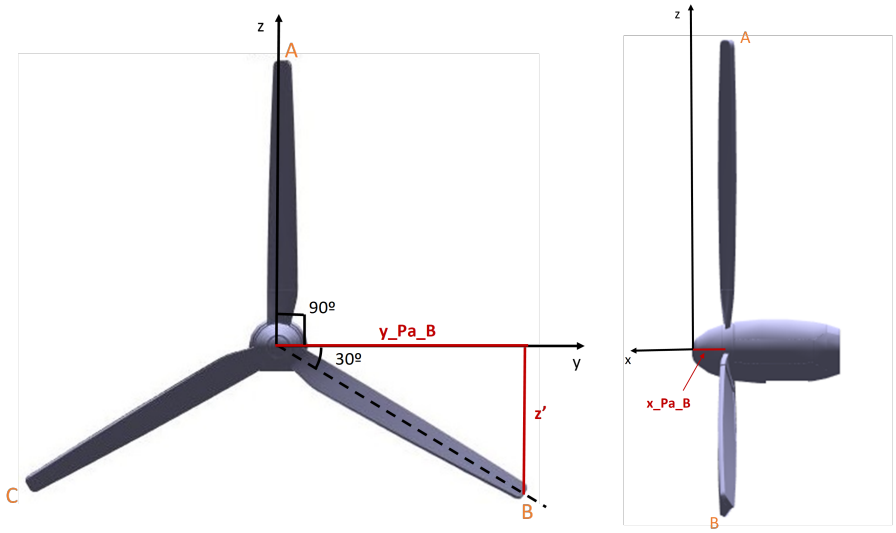
Figura 6.6: Representação esquemática da lógica para cálculo das coordenadas da pá A

No caso da pá B, o cálculo das suas coordenadas necessitou de uma maior atenção visto que a pá não se encontra alinhada com nenhum eixo, sendo necessário considerar o ângulo entre a pá e o eixo de referência. Tal como na pá descrita anteriormente, a coordenada x da pá B corresponde à diferença entre a coordenada x do rotor e a distância entre o rotor e a pá (figura 6.7(b)). No que toca à coordenada y, e tendo como base a trigonometria, esta será igual ao produto entre o comprimento da pá e o cosseno do ângulo entre a pá e o eixo y. Como o desfasamento entre pás corresponde a 120º, o ângulo formado entre a pá A e o eixo x é de 90º, então o ângulo formado entre a pá B e o eixo y é de 30º. Em relação à coordenada z da pá, esta será a diferença entre a coordenada z do rotor e z' representado na figura 6.7(a). Este z' corresponde ao produto

entre o comprimento da pá e o seno do ângulo, neste caso 30º. A equação que representa assim as coordenadas da pá B é:

$$\begin{cases} x_{pa\_B} = x_{rotor} - distancia_{pa\_rotor} \\ y_{pa\_B} = comprimento_{pas} * cos(\alpha) \\ z_{pa\_B} = z_{rotor} - (comprimento_{pas} * sin(\alpha)) \end{cases} \tag{6.2}$$

A figura 6.7 representa o enunciado anteriormente, estando a vermelho representadas as coordenadas x, y e z da pá B.



(a) Representação esquemática dos eixos y e z

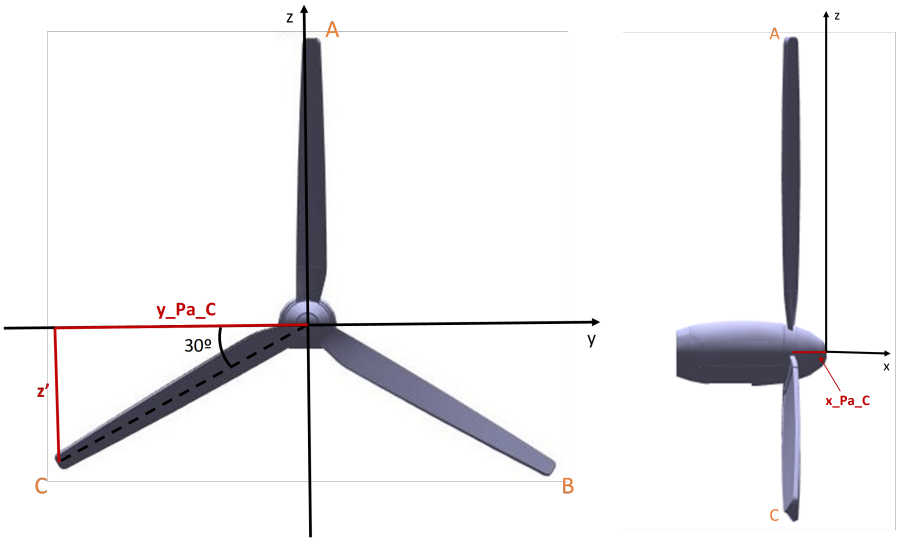
(b) Representação esquemática dos eixos x e z

Figura 6.7: Representação esquemática da lógica para cálculo das coordenadas da pá B

As coordenadas da pá C foram calculadas de forma semelhante ao cálculo efetuado para as coordenadas da pá B. A coordenada em x, tal como nas duas pás anteriores, corresponde à diferença entre a coordenada x do rotor e a distância entre o rotor e a pá (figura 6.8(b)). A coordenada y corresponde ao produto entre o comprimento da pá e o cosseno do ângulo entre a pá e o eixo yy, tendo esta coordenada que ser negativa uma vez que se encontra na parte negativa do eixo. Para o cálculo da coordenada z a linha de pensamento aplicada é a mesma que no cálculo de z da pá B, ou seja, esta será a diferença entre a coordenada z do rotor e z' representado na figura 6.8(a). Este z' corresponde ao produto entre o comprimento da pá e o seno do ângulo. As coordenadas da pá C são então representadas por:

$$\begin{cases} x_{pa\_C} = x_{rotor} - distancia_{pa\_rotor} \\ y_{pa\_C} = -comprimento_{pas} * cos(\alpha) \\ z_{pa\_C} = z_{rotor} - (comprimento_{pas} * sin(\alpha)) \end{cases} \tag{6.3}$$

Na figura 6.8 encontra-se uma representação visual do mencionado anteriormente, estando a vermelho representadas as coordenadas da pá C.



(a) Representação esquemática dos eixos y e z

(b) Representação esquemática dos eixos x e z

Figura 6.8: Representação esquemática da lógica para cálculo das coordenadas da pá C

6.2.2 Cálculo das retas entre pá e rotor/nacelle

Para cada pá foi necessário gerar duas retas, uma entre o rotor e a ponta da pá e a outra entre a nacelle e a ponta da pá. Para tal, criou-se uma função que gera a reta desejada, com um ponto inicial e um ponto final, contendo um determinado número de pontos. O número de pontos criados na reta pode ser definido pelo utilizador. O número de pontos deve ser escolhido tendo em atenção o FOV da câmara e ainda o diâmetro da pá. No caso desta dissertação, o *Horizontal Field-Of-View* (HFOV) foi considerado igual ao *Vertical Field-Of-View* (VFOV).

De modo a saber o HFOV, ou seja, o que a câmara consegue ver na horizontal, é necessário conhecer, *a priori*, dois parâmetros, sendo estes a distância a que me encontro do objeto e o ângulo de abertura da câmara, FOV. Com o conhecimento destas duas

variáveis, o HFOV pode ser calculado através da equação 6.4, onde D corresponde à distância ao objeto em milímetros e FOV corresponde ao campo de visão angular da câmara em graus. O resultado será o HFOV em milímetros. A figura 6.9 permite uma melhor perceção sobre as variáveis mencionadas.

$$HFOV = 2 \cdot D \cdot \tan\left(\frac{FOV}{2}\right) \tag{6.4}$$

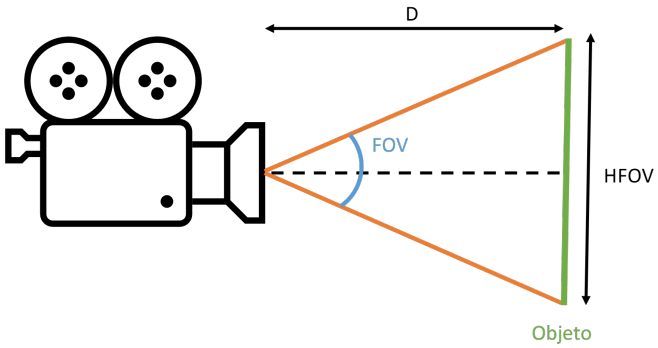


Figura 6.9: Informação importante sobre uma câmara

Assim, para este caso de estudo, foi necessário garantir que a câmara utilizada tinha um HFOV de, pelo menos, $3.5m$ que corresponde ao diâmetro das pás da ventoinha. Acabou por se optar por uma câmara com um FOV de 60^{o} e determinou-se uma distância ao objeto de $10\,m$. Aplicando a equação 6.4 obteve-se um HFOV de $11.55\,m$, demonstrando assim que a câmara seria capaz de ver a largura da pá da ventoinha na sua totalidade.

Com o valor de HFOV determinado, foi possível então determinar o número de pontos mínimo para garantir uma inspeção total da pá. Assim, o número mínimo de pontos será dado pela divisão entre o comprimento da pá e o HFOV da câmara.

$$n_{pontos} = \frac{comprimento_{pa}}{HFOV} \tag{6.5}$$

Para o caso em estudo, a pá utilizada apresenta um comprimento de aproximadamente $40\,m$ e o HFOV calculado é de $11.55\,m$. Aplicando a equação 6.5, o número mínimo de pontos obtido é 4. Isto significa que, para o sucesso do algoritmo, o número mínimo de pontos a ser utilizado na criação das retas é 4.

Para além da criação das retas, considerou-se também o cálculo de *yaw* para cada ponto criado, com o objetivo de o veículo efetuar a manobra olhando sempre para o

objeto a ser inspecionado. O cálculo do *yaw* encontra-se representado na equação 6.8, onde x_A,x_B,y_A e y_B representam as coordenadas dos pontos para os quais se pretende o cálculo do *yaw*.

$$yaw = atan2\left(\frac{y_A - y_B}{x_A - x_B}\right) \tag{6.6}$$

A figura 6.10 apresenta as retas geradas pelo algoritmo. Esta figura apresenta ainda setas coloridas, indicando o *yaw* calculado pelo algoritmo.

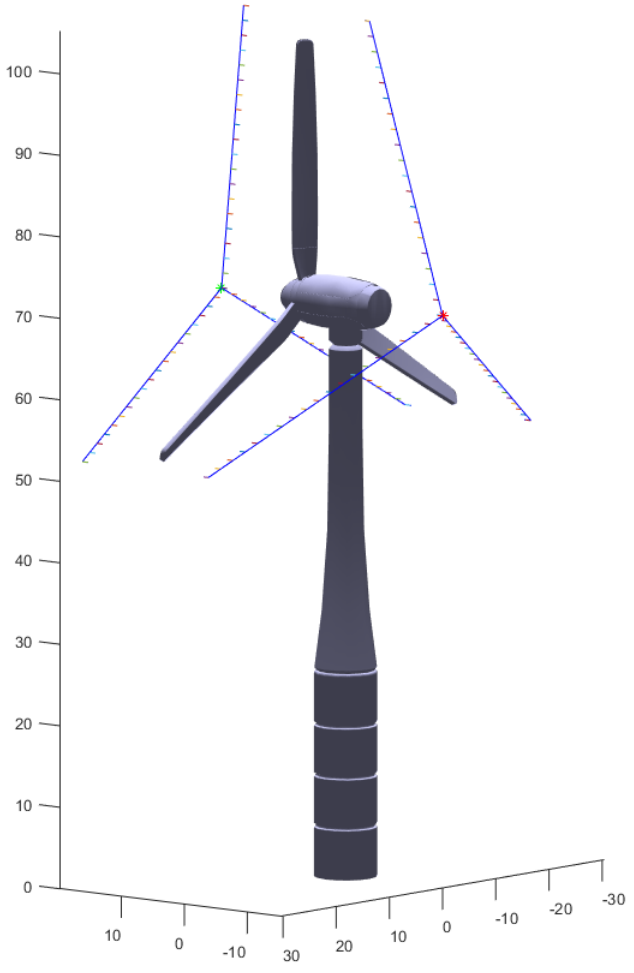


Figura 6.10: Retas nas pás geradas pelo algoritmo

6.2.3 Cálculo dos semicírculos laterais

Para a inspeção lateral foi projetada a criação de semicírculos, com origem na coordenada da pá. Estes semicírculos foram gerados utilizando uma função onde é necessário definir o inicio e final do semicírculo, entre 0 e 2π. Para além disso, é necessário definir também o número de pontos a serem gerados neste semicírculo, sendo esse número o mesmo que o definido para a criação de retas. Esta função devolve o ângulo de cada ponto, em radianos. Essa posição, em x e em y, foi depois convertida em coordenadas, como descritas na equação 6.7.

$$\begin{cases} O_{semicirculo} = [x, y, z]_{pa} \\ \alpha \in [0, 2\pi] \\ x_{semicirculo} = distancia_{inspecao} * cos(\alpha) + x_{origem} \\ y_{semicirculo} = distancia_{inspecao} * sin(\alpha) + y_{origem} \end{cases} \tag{6.7}$$

Após a obtenção dos pontos do semicírculo, procedeu-se ao cálculo do *yaw* para cada ponto calculado, seguindo a mesma abordagem descrita na equação 6.8. A figura 6.11 apresenta os semicírculos criados assim como setas coloridas que indicam o *yaw* calculado pelo algoritmo.

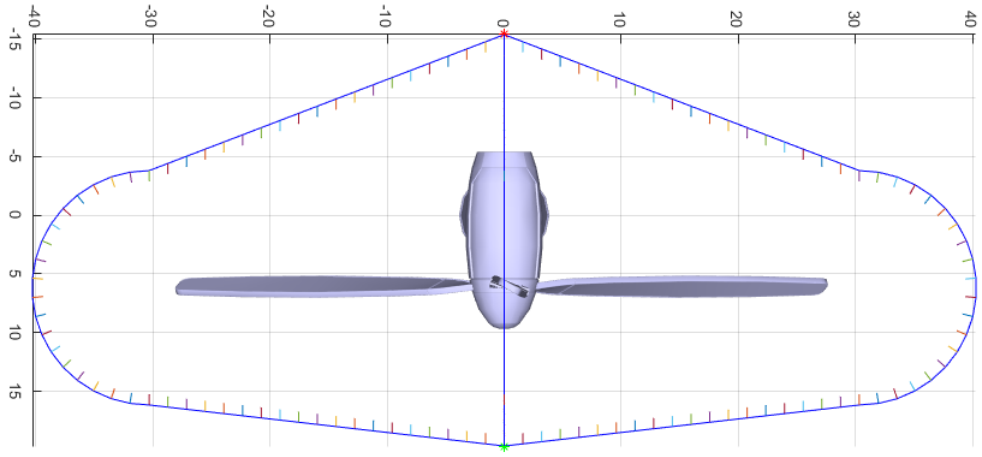


Figura 6.11: Semicírculos gerados pelo algoritmo

6.2.4 Ordenação do caminho gerado

Após a obtenção de todas a retas e semicírculos necessários para a inspeção, procedeu-se então a reorganização dos pontos de modo a que o algoritmo escolha o ponto seguinte como sendo aquele que apresente uma menor distância.

Procedeu-se à comparação de um determinado ponto com os restantes pontos existentes. Para cada comparação, foi calculada a distância e armazenada numa variável, juntamente com o id desse ponto. Após essa análise, foi identificado o ponto que apresentava menor distância ao ponto considerado, sendo esse o ponto a ser comparado no ciclo seguinte do algoritmo.

No final da reorganização, foi gerada uma trajetória que se encontra representada na figura 6.12, estando a verde representado o ponto inicial e a vermelho o ponto final.

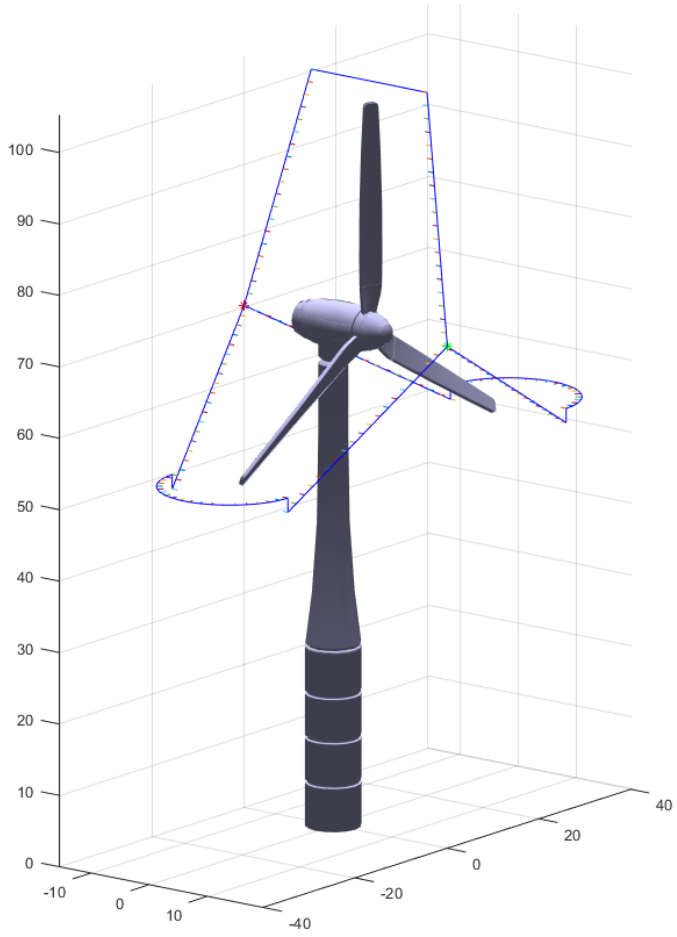


Figura 6.12: Trajetória gerada pelo algoritmo

6.2.5 Ficheiro de missão

Após a reorganização dos pontos calculados, foi necessário guardar a informação num ficheiro de texto, para que o mesmo fosse posteriormente analisado. Assim, foi aberto um ficheiro do tipo *txt* e definiu-se que seria do tipo *write*. Após a abertura, foi escrito no ficheiro cada ponto obtido, sendo escritas as coordenadas de cada ponto seguido do *yaw* correspondente. A estrutura do ficheiro encontra-se apresentada na figura 6.13.

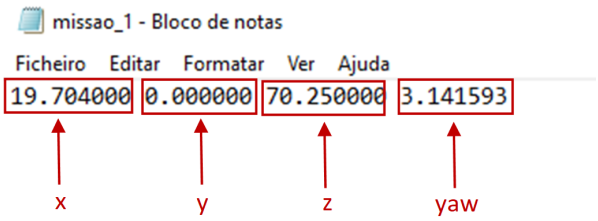


Figura 6.13: Estrutura do ficheiro de inspeção gerado

Após a escrita de todos os pontos, o ficheiro era fechado, estando assim pronto para ser analisado pelo código desenvolvido em ROS.

6.3 Módulos de *software* desenvolvidos

Tendo por base a *framework* ROS, foram desenvolvidos três nós com diferentes funcionalidades. Estes módulos serão descritos de seguida, onde serão enunciadas as principais funcionalidades de cada um deles no pipeline de planeamento de trajeto desenvolvido para a inspeção de eólicas.

6.3.1 VTOL Control

Módulo de *software* responsável pela determinação de todas as atitudes tomadas pelo veículo, utilizando para tal a função *mission()*. Dentro deste nó, são utilizados serviços, essenciais para o funcionamento do veículo, sendo estes o serviço de *arm/disarm*, definição do modo de voo e efetuar a transição do VTOL. Para além disso, este nó necessita de fazer um pedido para que o nó *Mission Generation* publique um novo ponto.

6.3.1.1 Serviços essenciais

Para armar e desarmar o veículo recorreu-se à utilização de um serviço ROS, que faz o envio de um *command_bool*, alterando o estado de *arm*. O serviço a ser utilizado pode ser definido como:

```
ros::ServiceClient arming_client = n.serviceClient
<mavros_msgs::CommandBool>("mavros/cmd/arming");
```

É também necessária a criação de uma função que recebe por parâmetro uma variável do tipo *bool* e procede à alteração do estado *arm* do serviço considerado.

Foi necessário incluir a alteração, através do código, do modo de voo do veículo. Para tal, recorreu-se à utilização do serviço *set_mode*.

```
ros::ServiceClient set_mode_client = n.serviceClient
<mavros_msgs::SetMode>("mavros/set_mode");
```

A função associada a este serviço recebe como parâmetro uma variável do tipo *string*, procedendo depois à alteração do modo de voo. Esta função permite receber qualquer modo de voo e proceder à sua alteração com sucesso.

O ROS contém ainda um serviço responsável pela alteração do modo de voo de um VTOL. Este serviço foi considerado de especial importância tendo em conta o veículo a ser utilizado neste projeto.

```
ros::ServiceClient set_vtol_transition = n.serviceClient
<mavros_msgs::CommandVtolTransition>("mavros/cmd/vtol_transition");
```

Para este serviço foi também necessária a criação de uma função que recebe por parâmetro um número inteiro que representa o modo pretendido para o veículo, sendo 3 a transição para multirotor e 4 a transição para *fixed wing*.

O último serviço a ser utilizado permitia a alteração de parâmetros. Este serviço seria necessário para a alteração do raio de *loiter* para o caso de ser efetuada uma inspeção em *loiter* ou ainda para a alteração do valor mínimo e máximo do *pitch* em *fixed wing*. Neste caso, a função associada a este serviço recebe por parâmetro uma *string* que deverá indicar o parâmetro a ser alterado e um *float* indicativo do valor do valor pretendido.

```
ros::ServiceClient set_loiter_radius = n.serviceClient
<mavros_msgs::ParamSet>("/mavros/param/set");
```

6.3.1.2 mission()

Esta função foi criada para decidir qual a atitude tomada pelo veículo de acordo com a informação proveniente do outro nó. A lógica dentro desta função encontra-se explícita no algoritmo 3. De uma forma simplificada, esta função vai estar a verificar o modo de voo que o veículo deverá ter no ponto para o qual se desloca. Isto permite um controlo automático das transições necessárias por parte do algoritmo desenvolvido.

Algoritmo 3 Mission Attitude

```
1: Save desired point into local variable
2: if Flight Mode = FW OR Flight Mode = FW_Screw then
3:    Transition to FW
4:    Change pitch min and max
5:    if Loiter Radius = 0 then
6:       check_navigation_FW()
7:    else
8:       if Flight Mode = FW then
9:          Changer loiter radius
10:         check_navigation_loiter()
11:      else
12:         Changer loiter radius
13:         Changer pitch min and max
14:         check_navigation_loiter()
15:         Publish request to record
16:      end if
17:   end if
18: else
19:   Calculate distance to point
20:   if Distance > Minimum distance Transition then
21:      Transition to FW
22:      check_navigation_FW_MC()
23:   else
24:      Transition to MC
25:      check_navigation_MC()
26:   end if
27: end if
```

Neste algoritmo, existem algumas funções que requerem uma melhor atenção, nomeadamente a função *check_navigation_FW()*, a função *check_navigation_loiter()*, a função *check_navigation_FW_MC()* e a função *check_navigation_MC()*.

A função *check_navigation_FW()* serve para efetuar o voo, entre pontos, em *fixed wing*. Nesta função, o controlo em posição é efetuado recorrendo à função *nav_local_pos_setpoint()*.

Sempre que é atingido a região de aceitação de um ponto gerado, será publicado um novo ponto de missão para o UAV. Para tal, considera-se que o veículo se encontra próximo do ponto quando a sua posição atual se encontra no intervalo $[pontodesejado - distânciaFW; pontodesejado + distânciaFW]$. Esta distância FW é uma variável que pode ser alterada e que define a distância mínima em relação ao ponto à qual é considerado oportuno o pedido de um novo ponto.

A função *nav_local_pos_setpoint()* tem como objetivo a publicação, para o tópico de ROS *setpoint_position*, do ponto para o qual se pretende ir, assim como ao cálculo da rotação associada a esse ponto.

```
ros::Publisher pub_pos_setpoint_local = n.advertise
<geometry_msgs::PoseStamped>("/mavros/setpoint_position/local", 1);
```

A *check_navigation_MC()* é utilizada para a viagem entre pontos em multirotor. Desde modo, a função utilizada para o controlo será *correct_pid_pos()*. Assim que o multirotor chega ao ponto pretendido, é efetuado um pedido para que seja atribuído um novo ponto. Enquanto este pedido não é atendido, o multirotor manterá a sua posição no ponto anterior. Assim que chega um novo ponto, são analisados os valores de x, y e z. Caso o novo ponto seja [0,0,0], o multirotor inicia o processo de aterragem. Caso contrário, irá proceder à viagem até ao próximo ponto, utilizando para tal a função *correct_pid_pos()*.

Esta função de controlo, *correct_pid_pos()*, tem como finalidade o controlo da velocidade com o recurso à malha PID. Neste caso, o tópico ROS utilizado para a publicação das velocidades obtidas será *setpoint_velocity*.

```
ros::Publisher pub_vel_setpoint_local = n.advertise
<geometry_msgs::TwistStamped>("/mavros/setpoint_velocity/cmd_vel", 1);
```

A função *check_navigation_FW_MC()* foi desenvolvida para que seja possível uma mistura entre voo em *fixed wing* e multirotor. Assim, é suposto que o veículo efetue o voo em *fixed wing* até que atinja uma determinada distância ao ponto pertendido. Quando essa distância é alcançada, o veículo transita para multirotor e desloca-se até ao ponto neste modo de voo. Nesta função, o controlo é efetuado recorrendo à função *nav_local_pos_setpoint()* e o pedido de novo ponto é efetuado quando o veículo atinge o ponto desejado.

Em *check_navigation_loiter()* é possível circundar o ponto desejado num número máximo de duas voltas. Assim que se entra nesta função, é iniciado um *loiter* em

torno do ponto desejado, utilizando para tal a função de controlo *nav_loitter_mode()*. À medida que vai fazendo *loiter* em torno do ponto, vai sendo calculada a distância entre o ponto atual e o ponto desejado. Se a distância calculada se encontrar no intervalo $[raio - 1; raio + 1]$, o ponto para o qual isto acontece é marcado como sendo um ponto de referência. Com esta marcação, o veículo terá que dar duas voltas ao ponto, passando duas vezes pelo ponto de referência.

A função *nav_loitter_mode()* é a responsável pela manobra de *loiter*. Para que a manobra seja efetuada, é necessário publicar para o tópico ROS *setpoint_raw* as coordenadas do ponto desejado. Com esta função é possível executar *loiters* descendentes, simulando um parafuso em torno de um ponto.

```
ros::Publisher pub_target_attitude = n.advertise
<mavros_msgs::PositionTarget>("/mavros/setpoint_raw/local", 1);
```

6.3.2 Mission Generation

Este nó foi criado com o principal objetivo de abrir os ficheiros contendo os *waypoints* das missões e proceder à publicação dos pontos sempre que fosse recebido um pedido proveniente do nó *Vtol Control*.

Este nó começa por fazer a abertura do ficheiro principal de missão, sendo a função *read_mission()* a responsável por esta ação. Este ficheiro é escrito pelo utilizador, e nele define-se, pela seguinte ordem:

- **x, y, z**: coordenadas do ponto para o qual o veículo deve viajar. Para o caso do modo de voo igual a 2, ou seja, para o parafuso, estas coordenadas serão correspondentes às coordenadas finais do *loiter*.

- ***yaw***: *yaw* que o veículo deve apresentar naquele ponto, em radianos. Só será considerado em graus quando o parâmetro "modelo"for preenchido ou na situação em que se pretende fazer um parafuso. Quando o parâmetro "modelo"é preenchido, o valor de *yaw* corresponde à rotação da *mesh* utilizada. Para um modo de voo igual a 2, ou seja, manobra do parafuso, o valor de *yaw* introduzido será o valor de *pitch* assumido pelo algoritmo.

- **Modo de voo**: modo de voo que o veículo deve apresentar no ponto desejado. Os valores admitidos são 0, 1 e 2, correspondendo a multirotor, *fixed wing* e *fixed wing* em parafuso, respetivamente. Este valor deverá ser definido pelo utilizador.

- **Raio do *loiter***: define o raio que o veículo terá que fazer no modo *loiter*, em metros. Para que o algoritmo considere este parâmetro como válido, é necessário que o modo de voo escrito no parâmetro anterior seja 1 (*fixed wing*).

- **Modelo**: define-se o ficheiro txt, de missão, que deverá ser aberto pelo nó. Caso não seja necessária a abertura de nenhum ficheiro, este parâmetro deverá ser considerado 0.

A escrita do ficheiro principal de missão tem que respeitar a ordem descrita anteriormente, sendo que em caso de o utilizador não querer definir algum parâmetro, o valor a colocar nesse parâmetro deverá ser 0.

Caso o ficheiro principal indique que é necessário efetuar a abertura de um ficheiro de missão, figura 6.14, ou seja, o parâmetro "modelo"indicar um valor diferente de zero, é chamada a função *read_file_model()*. Nesta situação, as coordenadas x, y e z definidas no ficheiro principal são consideradas como sendo as coordenadas do centro do modelo, o *yaw* considerado passa a ser a rotação aplicada ao modelo em questão. Nesta situação, o *yaw* terá que se encontrar em graus. Esta caraterística do algoritmo permite rodar os pontos da missão consoante a rotação apresentada pela *mesh* utilizada. Para todos os outros ângulos seguiu-se a lógica apresentada na equação 6.8, retirada de [79]:

$$\begin{cases} x_{ponto\_rodado} = x\_centro + (x_{ponto\_original}) * cos(yaw\_centro\_rad) - \\ \qquad\qquad (y_{ponto\_original}) * sin(yaw\_centro\_rad) \\ y_{ponto\_rodado} = y\_centro + (y_{ponto\_original}) * cos(yaw\_centro\_rad) + \\ \qquad\qquad (x_{ponto\_original}) * sin(yaw\_centro\_rad) \\ yaw_{ponto\_rodado} = yaw_{ponto\_original} + yaw\_centro\_rad \end{cases} \tag{6.8}$$

*mission_planner - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

0	0	105	0	0	0	0
500	500	105	0	0	0	0
500	500	0	0	0	0	missao_1
0	0	105	0	0	0	0
0	0	0	0	0	0	0
#X	Y	Z	YAW	MODE(MC=0, FW=1, FW_P=2)	LOITER RAD (FW ONLY)	MODEL

Figura 6.14: Ficheiro principal preenchido para uma inspeção em multirotor, *fixed wing* ou multirotor/*fixed wing*

De seguida, o novo ponto é publicado num tópico ROS criado pela autora deste projeto.

```
ros::Publisher pub_point = n.advertise<vtol_control::vtol>
("mission_info", 1);
```

Caso o ficheiro principal indique que não existe modelo a ser seguido 6.15, a função *read_mission()* continuará a analisar a informação contida no ficheiro principal da missão. Quando todos os pontos forem publicados, o ficheiro é fechado e o nó é encerrado.

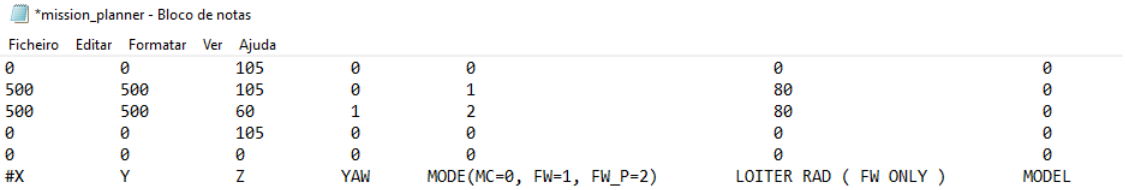


Figura 6.15: Ficheiro principal preenchido para uma inspeção em parafuso

6.3.3 Registo de dados da missão

Este nó foi criado com o intuito de fazer a gravação, para um ROSBAG, das imagens recolhidas pela câmara. Este nó foi criado de modo a ser possível definir o ponto exato onde seria necessário iniciar e terminar as gravações das imagens, para posterior comparação de algoritmos. Para tal, é feita a subscrição ao tópico *rec_instruction* para saber quando será necessário começar a gravação. Para além disso, é feita a subscrição ao tópico ROS *standard_vtol/usb_cam/image_raw* para obter as imagens capturadas pela câmara. Sempre que é recebida uma nova imagem, é colocada uma *flag* a *true*.

```
ros::Subscriber record_instruction_sub = n.subscribe<std_msgs::Int8>
("/rec_instruction", 1, rec_instruction_cb);
```

```
ros::Subscriber vtol_image_sub = n.subscribe<sensor_msgs::Image>
("/standard_vtol/usb_cam/image_raw", 1, image_sub_cb);
```

Assim que o *status* de gravação passar a 1 no *subscriber record_instruction_sub*, é aberto um ficheiro do tipo bag. Sempre que for recebida uma nova imagem e o *status* de gravação permanecer a 1, essa imagem será escrita no bag. Caso o *status* do bag passe para 0, a gravação é concluída e o bag é guardado.

6.4 Criação de pontos com *Structural Inspection Path Planning*

De modo a poder comparar este algoritmo com o algoritmo desenvolvido em MATLAB foi necessário criar as mesmas condições de teste para ambos os casos. Neste caso, o algoritmo *Structural Inspection Path Planning* permite que sejam alterados alguns parâmetros como pontos iniciais e finais da inspeção, parâmetros da câmara e o tamanho da *bounding box*. A *bounding box* é uma caixa que delimita as extremidades do algoritmo, ou seja, o caminho criado por este algoritmo apenas é criado dentro da *bounding box*.

O primeiro ficheiro a ser alterado seria o ficheiro *bigBenParam.yaml*, que contém as definições da câmara. Este permite a definição do valor de *pitch*, o FOV vertical e horizontal. Para os testes efetuados, o ângulo de *pitch* foi considerado 0, o FOV vertical e horizontal foram considerados iguais, apresentando um valor de 60^{o}.

De seguida foi necessário proceder à alteração de alguns parâmetros no nó responsável pelo carregamento da *mesh* e criação do caminho de inspeção. O nó utilizado para a simulação foi o *bigBen*, nó já existente, sendo necessário efetuar algumas alterações de modo a que a inspeção fosse efetuada à *mesh* pretendida. A primeira alteração efetuada consistiu na alteração do início e fim do cálculo do caminho, sendo o início no rotor e o fim na nacelle. Para além disso, é possível também alterar a distância máxima e mínima de inspeção, assim como o número de iterações do RRT*. Esta alteração pode ser observada nas seguintes linhas de código.

```
  geometry_msgs::Pose reqPose;

/* starting pose*/
  reqPose.position.x = 19.7;
  reqPose.position.y = 0.0;
  reqPose.position.z = 70.25;

  /* final pose*/
  reqPose.position.x = -15.4;
  reqPose.position.y = 0.0;
  reqPose.position.z = 70.25;

  /* parameters for the path calculation */
  srv.request.minDist = 10.0;
```

```
srv.request.maxDist = 50.0;
srv.request.numIterations = 20;
```

Foi necessário também redefinir a *bounding box* de acordo com as dimensões da *mesh* utilizada. Para além do tamanho, foi necessário também a definição do centro dessa caixa. Estas edições são efetuadas no nó *bigBen*, estando abaixo representadas as linhas de código que permitem estas alterações.

```
koptplanner::inspection srv;
srv.request.spaceSize.push_back(60);
srv.request.spaceSize.push_back(80);
srv.request.spaceSize.push_back(100);
srv.request.spaceCenter.push_back(0);
srv.request.spaceCenter.push_back(0);
srv.request.spaceCenter.push_back(79);
```

Após as alterações efetuadas, o algoritmo faz o cálculo do melhor caminho, estando o mesmo representado na figura 6.16.

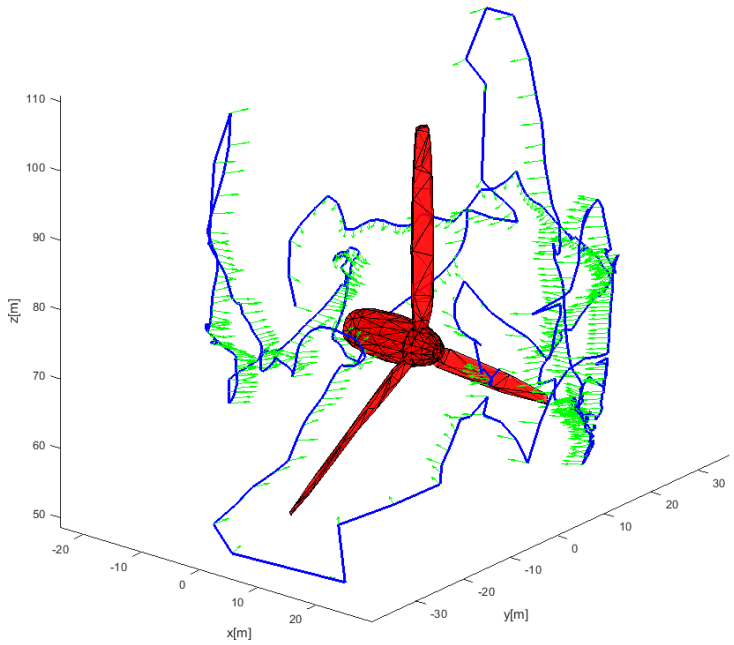


Figura 6.16: Caminho calculado, para as condições descritas, com o algoritmo *Structural Inspection Path Planning*

6.5 Gazebo

De modo a analisar o comportamento do veículo para as missões geradas, foi necessário recorrer à utilização do simulador Gazebo. Neste simulador, dado o caso de estudo deste projeto, foi necessário proceder à criação de um mundo que contivesse todos os elementos necessários para a representação do caso a ser estudado. Assim, nos tópicos seguintes, serão abordados os diferentes componentes adicionados ao mundo.

6.5.1 Ambiente de simulação

O primeiro passo necessário seria a criação de um modelo de uma ventoinha eólica. Por defeito, este tipo de modelo não se encontra implementado no Gazebo. Assim, foi necessário fazer a criação dos ficheiros necessários para a definição do modelo no simulador. O modelo desenvolvido encontra-se representado na figura 6.17.

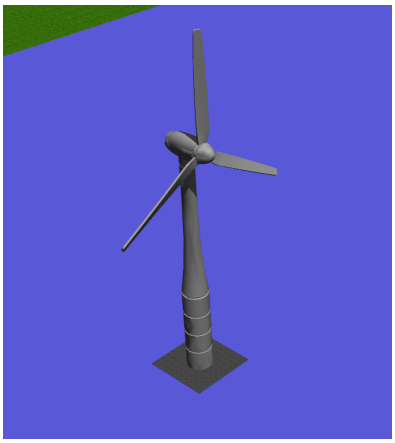


Figura 6.17: Modelo da eólica desenvolvido para Gazebo

No ambiente de simulação foram colocadas quatro eólicas no meio da água, desfasadas no eixo do y de 600 metros. A primeira eólica apresenta de coordenadas [500,500,0], a segunda eólica [500,1100,0], a terceira eólica [500,1700,0] e, a quarta e última eólica, [500,2300,0].

Para a simulação da costa utilizou-se o modelo *grass_plane* já existe no Gazebo. O ambiente aquático foi simulada com a inserção do modelo *water_level* ao Gazebo. Para além destes modelos, foi inserido também o modelo *asphalt_plane* na base de cada eólica e no local de *takeoff* do veículo. O resultado do ambiente construído encontra-se representado na figura 6.18.

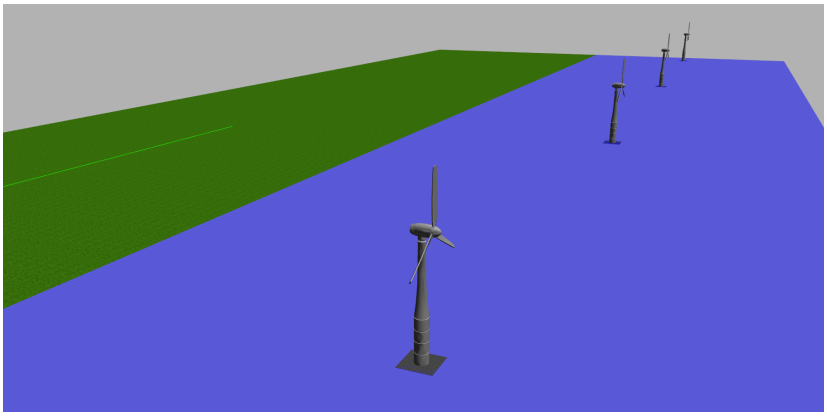


Figura 6.18: Mundo criado em Gazebo

6.5.2 Integração de uma câmara no UAV

De modo a ser possível a obtenção de imagens durante a inspeção, foi necessário proceder à integração de uma câmara ao modelo de VTOL utilizado. Para o caso desta simulação foi utilizado o modelo *standard_vtol*, já disponível no Gazebo. Assim, a introdução da câmara no modelo considerado passou pela adição das seguintes linhas ao modelo de VTOL.

```
<include>
    <uri>model://fpv_cam</uri>
    <pose>0.4 0 -0.1 0 -0.0872 -1.5707</pose>
</include>

<joint name="fpv_cam_joint" type="fixed">
    <child>fpv_cam::link</child>
    <parent>standard_vtol::base_link</parent>
    <axis>
        <xyz>0 0 1</xyz>
        <limit>
          <upper>0</upper>
          <lower>0</lower>
        </limit>
    </axis>
</joint>
```

No ficheiro XML apresentado é possível constatar a integração de uma câmara e a sua posição em relação ao *frame* do veículo. A figura 6.19 representa o VTOL no ambiente de simulação. Nesta figura é ainda possível verificar que existe uma pequena janela, que corresponde à visão da câmara na simulação.

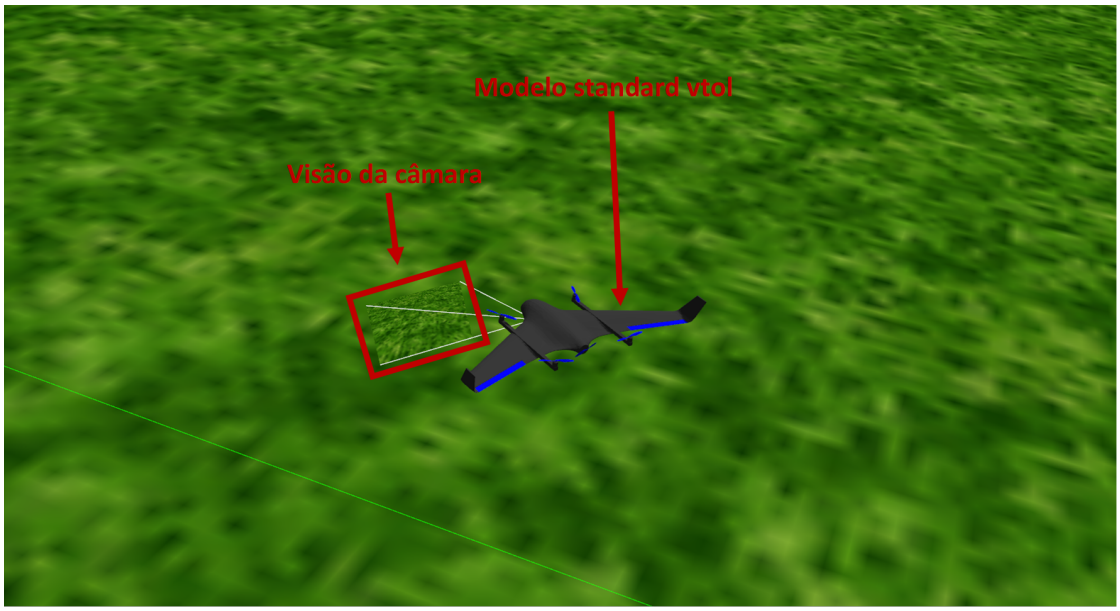


Figura 6.19: Modelo VTOL utilizado em simulação

6.5.3 Integração do vento no ambiente simulado

Um dos desafios inerentes ao processo de inspeção de eólicas em alto mar e a existência de ventos fortes que poderão afetar a qualidade de inspeção do UAV. Nesse sentido, introduzimos essa componente na simulação de modo a validar o comportamento do veículo nestas condições.

O Gazebo já possuía um *plugin* para a simulação de vento, sendo possível neste *plugin* definir a velocidade do vento (m/s), alterando o parâmetro *windVelocityMean*. Este *plugin* permitia ainda a alteração da direção do vento, sendo necessário alterar o parâmetro *windDirectionMean*. Apesar de ser possível a alteração de mais parâmetros, estes foram os considerados importantes para o caso de estudo.

O *plugin* necessário para esta simulação encontra-se representado nas seguintes linhas de código, sendo necessária a sua adição ao mundo a utilizar.

```
<plugin name='wind_plugin' filename='libgazebo_wind_plugin.so'>
      <frameId>base_link</frameId>
      <robotNamespace/>
      <xyzOffset>1 0 0</xyzOffset>
      <windDirectionMean>0 1 0</windDirectionMean>
      <windVelocityMean>0.0</windVelocityMean>
      <windGustDirection>0 0 0</windGustDirection>
      <windGustDuration>0</windGustDuration>
      <windGustStart>0</windGustStart>
      <windGustVelocityMean>0</windGustVelocityMean>
      <windPubTopic>world_wind</windPubTopic>
    </plugin>
```

6.5.4 Alteração da massa do veículo

De modo a entender a diferença do peso do veículo no seu comportamento, e especialmente quando existe vento, foi necessário procurar o parâmetro necessário para esta alteração. Assim, no ficheiro que define o modelo de Gazebo, existe um parâmetro denominado de *mass*, que corresponde à massa do veículo. Este parâmetro encontra-se em kg, vindo por *default* o valor de 5 kg. Esta massa será mantida num valor de 5 kg para os testes finais uma vez que é a massa aproximada da estrutura VTOL assemblada.

Esta página foi intencionalmente deixada em branco.

Capítulo 7

Resultados

Neste capítulo serão detalhados os resultados obtidos nas simulações efetuadas. Começarão por ser abordados os testes experimentais efetuados com a plataforma VTOL assemblada, seguindo-se os resultados das simulações do algoritmo implementado.

7.1 Testes experimentais efetuados com a plataforma VTOL

Os testes de voo que serão apresentados correspondem ao modo de voo em multirotor, sendo os mesmos testes reais. Nos primeiros voos, foi detetado que o veículo apresentava uma rotação indesejada em *yaw*[1] [2] e uma vibração em *pitch*. De modo a resolver este problema, foi necessário a calibração da malha PID. Esta calibração permitiu obter um veículo mais estável[3] no seu voo.

A análise dos *logs* de voo permitiu uma melhor compreensão sobre a interferência desta calibração na estabilidade do veículo. A figura 7.1 e 7.2 permite verificar o estado de *pitch* antes e após a calibração de PID.

[1] `https://www.youtube.com/watch?v=9moH8nvL68s`
[2] `https://www.youtube.com/watch?v=jMZzGwYfvHk`
[3] `https://www.youtube.com/watch?v=px3XkDsV84k`

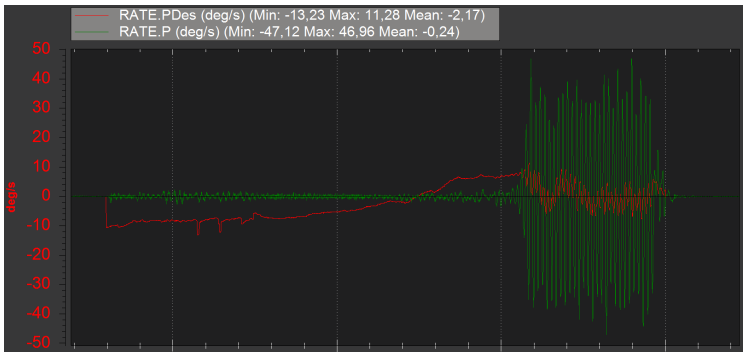


Figura 7.1: *Pitch* antes da calibração de PID

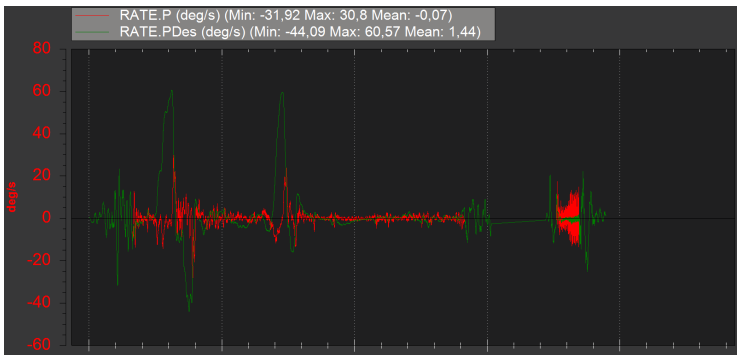


Figura 7.2: *Pitch* após calibração de PID

Em *roll* foi também efetuada calibração. No entanto, tendo em conta a semelhança de gráficos antes e após a calibração, só será demonstrado o gráfico após a calibração, estando este representado na figura 7.3.

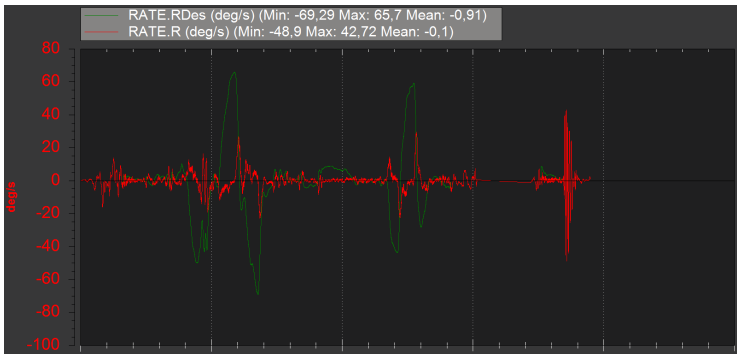


Figura 7.3: *Roll* após calibração de PID

Por último, foi necessário proceder à calibração de *yaw*. Para além da calibração de PID, foi necessário proceder à calibração da bússola, uma vez que o *log* obtido apresentava algumas anomalias. Estas anomalias podem ser verificadas na figura 7.4, que representa o *yaw* antes das calibrações. Na figura 7.5 pode ser observado o *yaw* do veículo, após as calibrações descritas anteriormente.

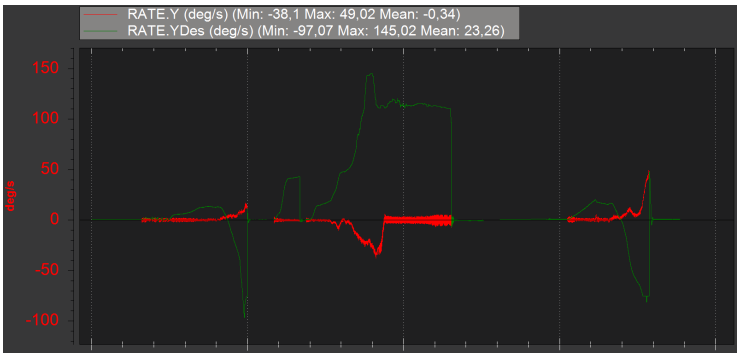


Figura 7.4: *Yaw* antes da calibração de PID e da calibração da bússola

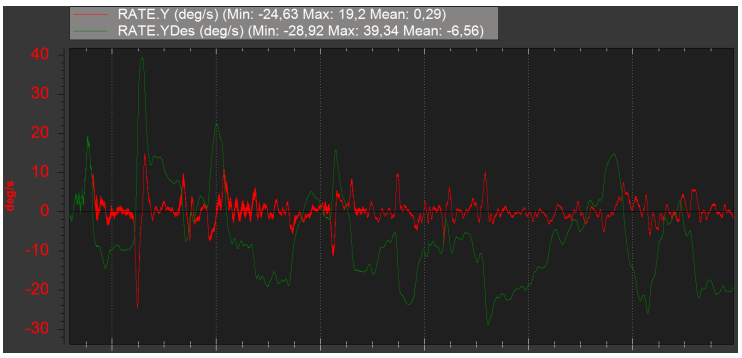


Figura 7.5: *Yaw* após da calibração de PID e da calibração da bússola

Em termos de voos reais estes só puderam ser efetuados em multirotor. Isto deveu-se ao facto de ter existido a interferência da pandemia COVID-19 e ainda a falta de um local apropriado para a realização desses testes.

7.2 Relação peso/vento

Os primeiros testes a serem efetuados consistiram na adição de vento para determinados pesos da estrutura. Estes testes foram realizados de modo a verificar o comportamento do veículo em situações de vento e perceber a influência do peso da estrutura.

O primeiro teste a ser efetuado consistiu na mudança da massa do veículo para 1 kg. Tendo a massa definida, foi-se alterando os valores do vento, sendo estes de 0 m/s, 5 m/s, 10 m/s, 15 m/s e 20 m/s. Para cada velocidade, procedeu-se à aplicação de vento em y, tendo sido feitos voos horizontais (eixo x, figura 7.6(a)) e longitudinais (eixo y, figura 7.6(b)). Nestas figuras, a azul, encontra-se representado o caminho percorrido pelo veículo.

(a) Exemplo de um voo horizontal (b) Exemplo de um voo longitudinal

Figura 7.6: Caminho percorrido pelo veículo, para voo horizontal e longitudinal

A velocidade no vento é definida em metros por segundo. A tabela 7.1 permite observar a conversão de m/s para km/h.

Tabela 7.1: Velocidade do vento

m/s	km/h
0	0
5	18
10	36
15	54
20	72

Os testes de voo simulados tinham como destino o mesmo ponto em todas as situações consideradas. Para a comparação de resultados, foram tidos em consideração alguns parâmetros, nomeadamente o tempo total de voo, a distância percorrida, a diferença de altitude, estando os resultados apresentados na tabela 7.2, para uma massa de 1 kg.

Tabela 7.2: Resultados obtidos para massa de 1 kg

ID	Vel. Vento (m/s)	Tipo Voo	Tempo Voo (s)	Dist.(km)	Dif. Alt (m)
1	0	Horizontal	215,604797	2.17	6.2
2	5	Horizontal	218,098986	2.16	5.7
3	10	Horizontal	221,008598	2,15	5.3
4	15	Horizontal	225,999104	2.29	7.2
5	20	Horizontal	234,720173	2.14	11.0
6	0	Longitudinal	218,139599	2.23	6.9
7	5	Longitudinal	235,371787	2,28	8.3
8	10	Longitudinal	316,904192	2,31	16.7
9	15	Longitudinal	365,914112	2.36	27.4
10	20	Longitudinal	395,757056	2,96	54.7

Após a análise dos resultados obtidos, foi possível concluir que quanto maior a velocidade do vento, maior o tempo de voo do veículo. Tendo em atenção a diferença de altitude obtida, é possível concluir que esta se encontra relacionada com a distância total percorrida. Quanto maior for a diferença de altitude, maior será a distância percorrida pelo veículo. O voo com ID 10 foi considerado instável, uma vez que apresenta uma grande oscilação nas variações da sua altitude, como pode ser observado na figura 7.7. Isto acontece devido ao facto de o veículo possuir pouca massa e se encontrar a ser exposto a uma velocidade de vento de 20 m/s.

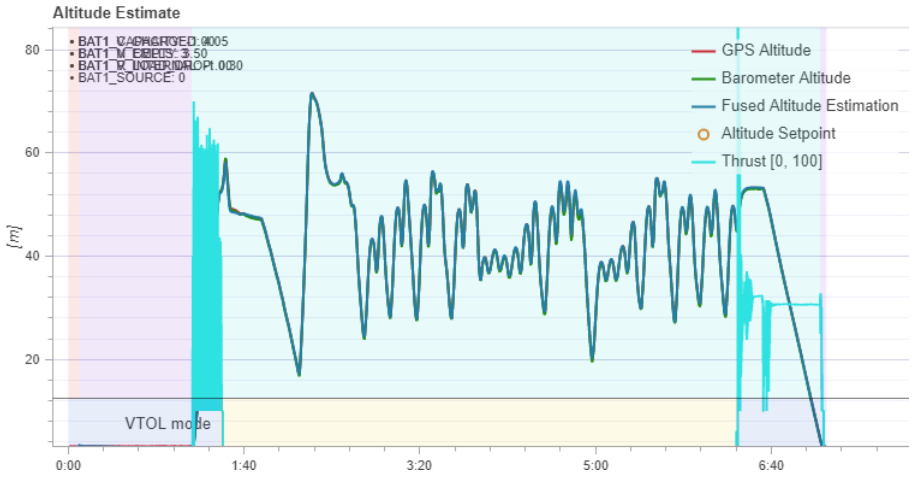


Figura 7.7: Gráfico da diferença de altitude para o ID 10

O segundo teste efetuado consistiu no aumento da massa do veículo para 5 kg, valor *default* da estrutura. Para esta massa, os testes efetuados foram iguais aos apresentados anteriormente, tendo sido acrescentado o tipo de voo diagonal (figura 7.8). Este acrescento foi considerado importante unicamente para esta massa uma vez que esse valor será o utilizado para as simulações finais. Os resultados obtidos para estes testes encontram-se apresentados na tabela tabela 7.3.

Tabela 7.3: Resultados obtidos para massa de 5 kg

ID	Vel. Vento (m/s)	Tipo Voo	Tempo Voo (s)	Dist.(km)	Dif. Alt (m)
11	0	Horizontal	211,808862	2,17	6.4
12	5	Horizontal	218,527253	2,17	7.5
13	10	Horizontal	219,900393	2,16	6.3
14	15	Horizontal	229,194559	2,15	6.5
15	20	Horizontal	238,832367	2,15	10.4
16	0	Longitudinal	220,818037	2.21	6.2
17	5	Longitudinal	229,569817	2,24	6.5
18	10	Longitudinal	222,920525	2,28	27.2
19	15	Longitudinal	227,725307	2,31	33.5
20	20	Longitudinal	244,911877	2.58	54.2
21	0	Diagonal	264,988822	2,99	5.8
22	5	Diagonal	290,565649	3,02	7.3
23	10	Diagonal	272,717367	3,05	14.1
24	15	Diagonal	271,646176	3.06	38.4
25	20	Diagonal	266,529681	3,07	41.4

Figura 7.8: Caminho percorrido pelo veículo no voo diagonal

De uma forma geral, com a análise da tabela 7.3, é possível concluir que mesmo para uma massa de 5 kg, o vento continua a influenciar o desempenho do UAV. Assim, quanto maior a velocidade do vento, maior será o tempo de inspeção. A adição de vento influencia também a variação de altitude do veículo, ou seja, quanto maior a velocidade do vento, maior vai ser a dificuldade para a manutenção da mesma altitude. O que acontece nestes casos, é que o veículo vai alterando a sua altitude de modo a conseguir efetuar a missão com sucesso.

Nesta tabela é ainda possível identificar que o teste com ID 20 foi considerado instável uma vez que, tal como aconteceu com o ID 10, apresentou mudanças de altitude muito significativas, estando o gráfico da mesma apresentado na figura 7.9.

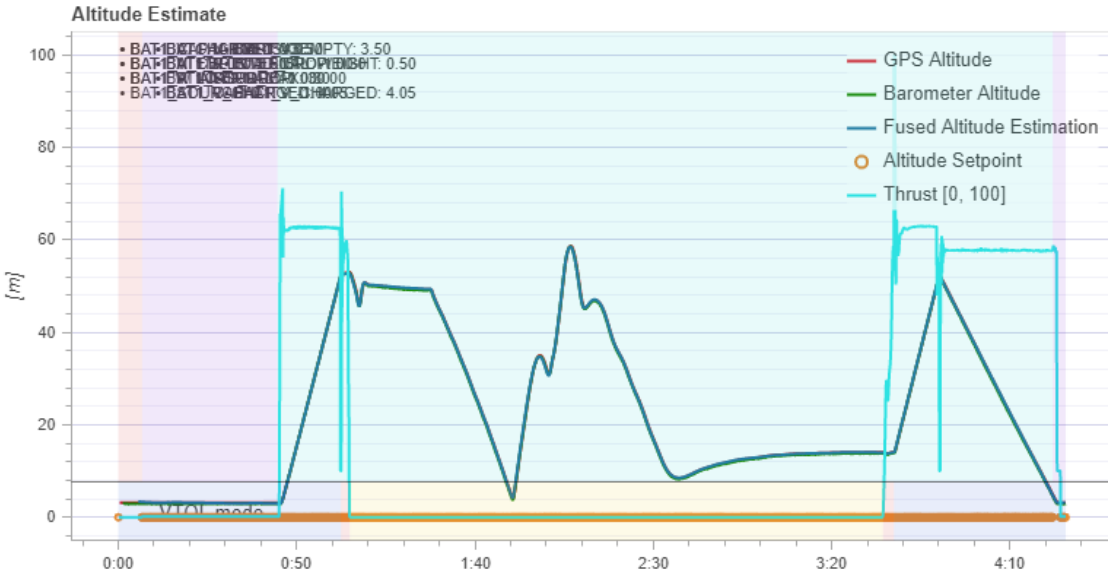


Figura 7.9: Gráfico da altitude durante o voo para o ID 20

7.3 Simulação das manobras de inspeção

7.3.1 Simulação do algoritmo desenvolvido

Após a obtenção do ficheiro com a missão em MATLAB, este foi analisado nos módulos de código desenvolvido em ROS. Em ROS, foi efetuada a manobra de inspeção obtida, sendo aplicados diferentes modos de voo, nomeadamente:

- **Voo MC**: a missão foi efetuada toda em multirotor.

- **Voo FW/MC**: o veículo voa em modo *fixed wing* até se encontrar perto da ventoinha. Quando isso acontece, o veículo transita para multirotor, efetuando a manobra de inspeção neste modo de voo. Após a conclusão da missão, o veículo retorna ao ponto de partida, no modo *fixed wing*.

- **Voo P**: o veículo viaja até à eólica em modo *fixed wing*. Quando chega à eólica, efetua a manobra parafuso em torno da mesma, a um raio de 50 metros. Após a conclusão desta manobra, o veículo regressa à origem em modo *fixed wing*.

Estes testes foram realizados de modo a demonstrar a importância da utilização de um VTOL na inspeção.

Foram efetuados testes para a situação em que não existia vento e para a situação onde existia vento. O vento foi aplicado no eixo y, a uma velocidade de 10 m/s. A câmara utilizada apresentava um *pitch* de 0^{o} para todos os voos efetuados. Para o voo em MC e FW/MC, o *yaw* utilizado era de 0^{o}, no entanto, para o modo de voo loiter, introduziu-se um *yaw* na câmara de 90^{o} para que fosse possível a captação de imagens. Os resultados obtidos com estas simulações encontram-se apresentados na tabela 7.4.

Tabela 7.4: Resultados obtidos para os diferentes modos de voo

Tipo de Voo	Vel. Vento (m/s)	Ficheiro txt	Tempo Missão(s)
MC	0	missao_MATLAB	1147.5825
FW/MC	0	missao_MATLAB	761.9453
P	0	-	314.9932
MC	10	missao_MATLAB	1008.5101
FW/MC	10	missao_MATLAB	601.3162
P	10	-	297.1683

Para o caso de estudo, e após a análise dos resultados apresentados na tabela anterior, é possível concluir que, mesmo com a aplicação de vento no mundo, a solução híbrida é a que apresenta melhores resultados quando se pretende uma inspeção pormenorizada da ventoinha. O voo em *fixed wing* até ao objeto a ser inspecionado permite não só uma poupança de energia por parte do veículo mas também uma diminuição significativa do tempo total de missão.

No que toca à inspeção efetuada em parafuso, e quando comparada com as outras duas, apresenta tempos de simulação mais reduzidos. Isto acontece uma vez que a missão é toda efetuada em modo avião. No entanto, este tipo de manobra permite apenas uma inspeção mais superficial, não sendo possível a verificação de falhas de tamanho reduzido.

É importante referir que a missão desenvolvida em MATLAB apresenta um custo de 306.90 metros, que corresponde à distância total percorrida durante a missão.

7.3.2 Simulação do algoritmo *Structural Inspection Path Planning*

Para a obtenção do melhor caminho de inspeção, foram alterados alguns parâmetros deste algoritmo de modo a obter o menor custo possível. Assim, nas diferentes simulações, foram alterados número de iterações do RRT*, o tamanho da bounding box e o ângulo de incidência.

A *bounding box* foi definida tendo em conta as medidas da área a ser inspecionada da ventoinha. Em x apresenta uma medida de 15 metros, em y 60 metros e em z 60 metros. A figura 7.10 corresponde a uma representação das medidas da área a ser inspecionada nos eixos x e y.

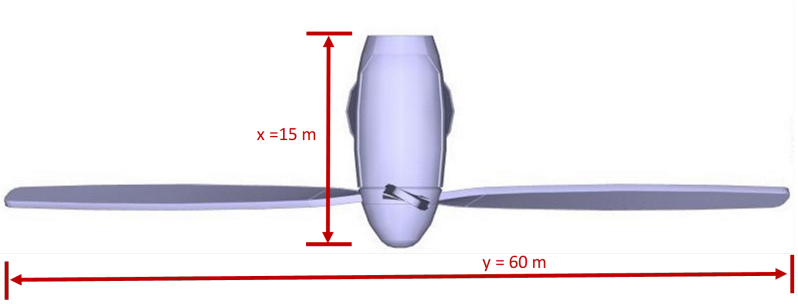


Figura 7.10: Medidas da área a ser inspecionada nos eixos x e y

De modo a garantir uma manobra de inspeção viável, começou por se definir uma *bounding box* de [50,80,80]. No entanto, os resultados obtidos não foram satisfatórios, tendo os valores sido alterados para [60,80,100]. Para estes valores de *bounding box* foram encontrados resultados satisfatórios para um número de iterações de 20. De modo a compreender quais os melhores valores a serem utilizados, foram-se fazendo testes para uma *bounding box* de [60,80,100], alterando-se o número de iterações. Os resultados encontram-se na tabela 7.5.

A análise dos valores apresentados acima permitiu perceber que o melhor caminho gerado seria aquele que apresentaria menor custo, tendo sido esse obtido com uma *bounding box* de [60,80,100] e 20 iterações. Tendo em conta o sucesso na obtenção do caminho, foi ainda aumentado o valor da *bounding box* para [80,80,100], no entanto o caminho encontrado não se tornou uma melhor opção.

Tabela 7.5: Resultados obtidos para as diferentes simulações do algoritmo *Structural Inspection Path Planning*

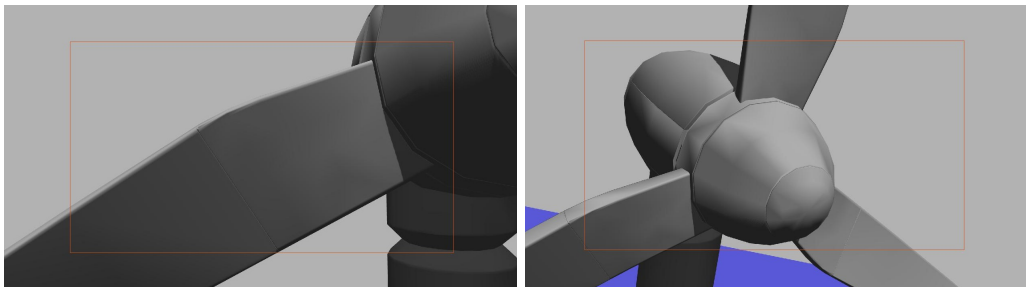
Nº Iterações	*Bounding Box* (m)	Custo(m)
20	[60,80,100]	824.12
50	[60,80,100]	965.20
10	[60,80,100]	943.14
20	[80,80,100]	851.31

7.3.3 Manobra desenvolvida VS *Structural Inspection Path Planning*

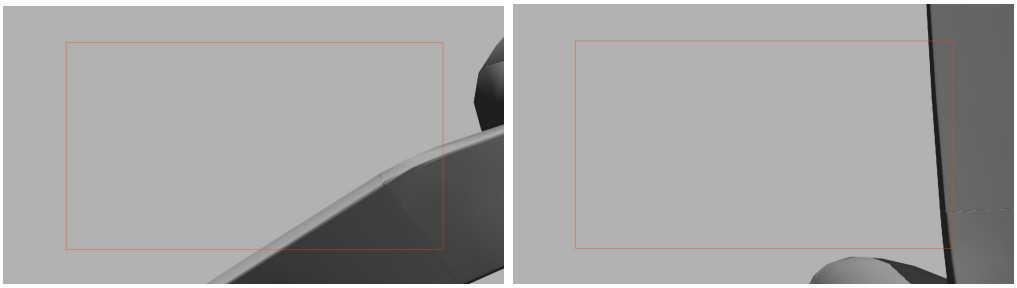
De modo a classificar as imagens, foi desenvolvido um pequeno código em MATLAB que efetua a abertura do ficheiro bag, efetuando assim a abertura das imagens. Em cada imagem foi introduzida uma caixa, representada na figura 7.11 a cor de laranja, que permite selecionar as imagens consideradas como bons resultados. Ou seja, se o ativo aparecer dentro dessa caixa, a imagem é considerada válida. Caso contrário, a imagem é descartada. Esta validação tem que ser efetuada pelo utilizador.

Figura 7.11: Caixa utilizada para diferenciar imagens válidas de imagens inválidas

No entanto, existem casos onde o ativo aparece dentro da caixa cor de laranja mas a imagem foi desconsiderada. Para o caso da inspeção de uma pá, é considerada válida uma imagem onde seja possível verificar a existência de duas extremidades do elemento de inspeção. Na figura 7.12 encontram-se exemplos de figuras consideradas válidas (figuras 7.12(a) e 7.12(b)) e de figuras consideradas inválidas (figuras 7.12(c) e 7.12(d)).



(a) Imagem considerada válida para a inspeção (b) Imagem considerada válida para a inspeção



(c) Imagem descartada (d) Imagem descartada

Figura 7.12: Exemplos de imagens consideradas válidas e inválidas

De modo a ser possível uma comparação entre os algoritmos de inspeção, foi necessário proceder a testes iguais, com parâmetros de câmara iguais e sem vento no mundo. Esta comparação só será abordada para o tipo de voo MC, ou seja, para uma missão efetuada apenas em multirotor. Os resultados obtidos para os algoritmos encontram-se apresentados na tabela 7.6, sendo A a manobra desenvolvida e B o *Structural Inspection Path Planning*.

Tabela 7.6: Resultados obtidos para os algoritmos de inspeção

	Img. Obtidas	Img. Válidas	Img. Válidas (%)	Tempo Insp (s)	Custo (m)
A	1375	1106	80.4	290.27	306.90
B	3518	2036	57.9	732.27	824.12

O ficheiro *bag* gerado com o método B teve uma dimensão elevada, tendo sido difícil a análise das imagens recolhidas em MATLAB. Este algoritmo apresenta um elevado tempo na inspeção e um elevado custo quando comparado com o algoritmo desenvolvido. Para além disso, a percentagem de imagens válidas apresenta uma grande diferença, sendo essa percentagem de 80.4% para o método A e 57.9% para o método B. Isto significa que o algoritmo desenvolvido apresenta uma melhor performance quando comparado com o

algoritmo *Structural Inspection Path Planning*.

Esta comparação apenas foi efetuada para o modo de voo MC devido ao tempo de processamento levado a cabo pela simulação, pelo tamanho do bag gerado e ainda pelo número de imagens obtidas com o algoritmo *Structural Inspection Path Planning*.

7.3.4 Avaliação da cobertura de área

Dada a diferença percentual de imagens válidas dos dois métodos, foi necessário perceber se, apesar desta diferença, os métodos conseguiriam obter imagens suficientes para uma reconstrução da eólica, ou seja, recriar uma imagem panorâmica.

Para tal, utilizou-se um *software* onde é necessário introduzir as diferentes imagens obtidas durante a inspeção, sendo o seu *output* uma imagem panorâmica da eólica. Para esta reconstrução, apenas foram consideradas as imagens que permitem visualizar a parte frontal da eólica.

A figura 7.13 corresponde à imagem panorâmica obtida para o método A e a figura 7.14 corresponde à imagem panorâmica obtida para o método B.

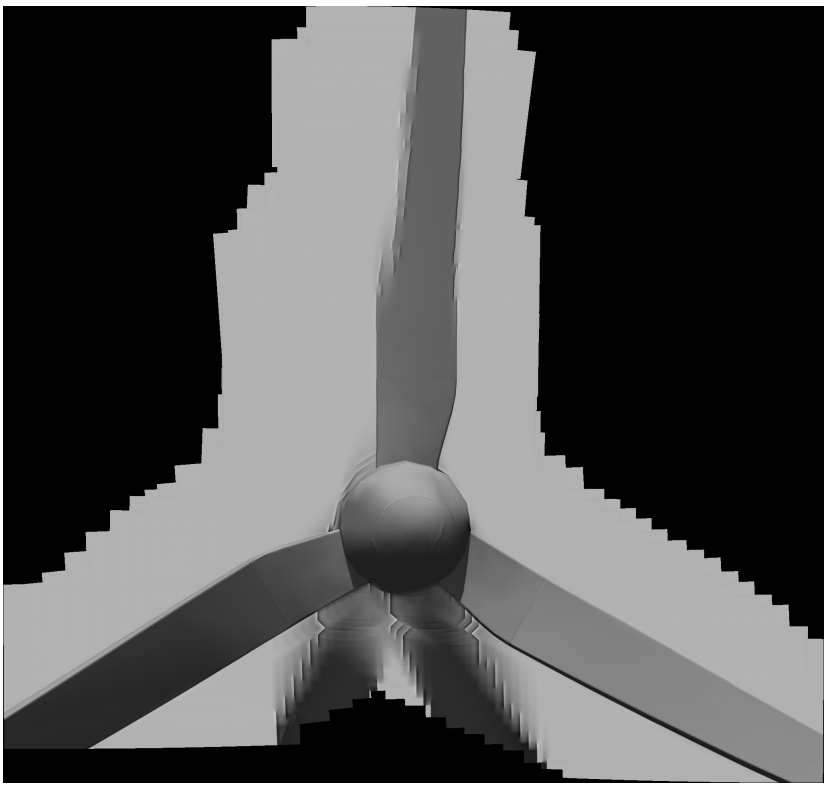


Figura 7.13: Imagem obtida após processamento das imagens de inspeção do método A

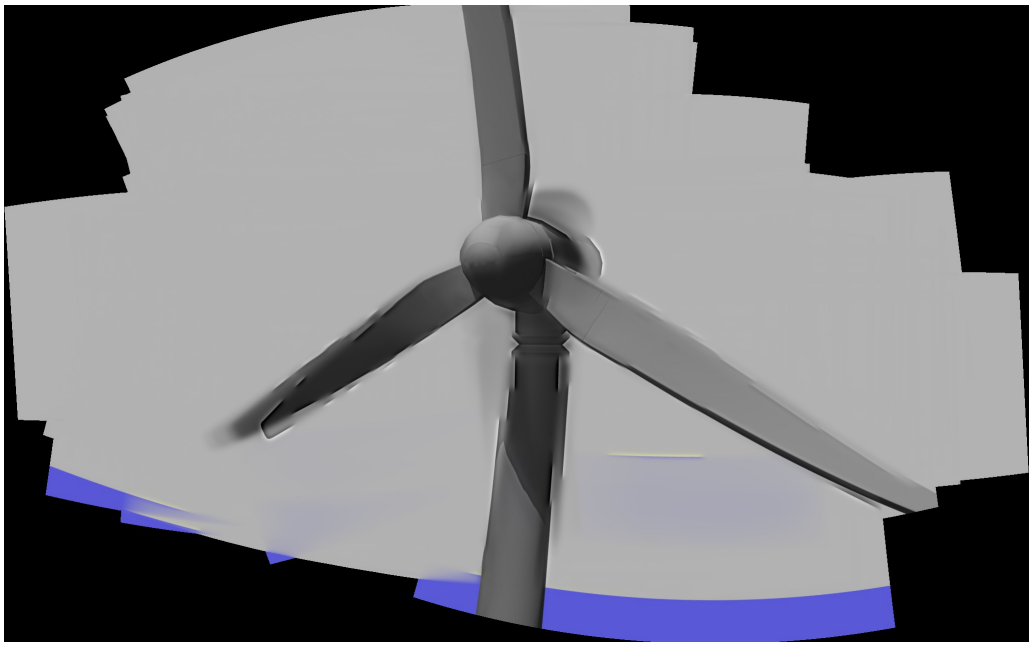


Figura 7.14: Imagem obtida após processamento das imagens de inspeção do método B

Após esta análise, foi possível verificar que ambos os métodos permitem uma inspeção viável da eólica. Apesar da diferença no número de imagens e do tempo de processamento dos dois métodos, é notória a capacidade de ser efetuada uma inspeção válida em ambos os casos.

Esta página foi intencionalmente deixada em branco.

Capítulo 8

Conclusões e Trabalho Futuro

A presente dissertação contribuiu para a consolidação de conhecimentos relativos a métodos de planeamento de trajetória que podem ser adaptados para o processo de inspeção de ativos elétricos como é o caso particular das eólicas.

O avanço tecnológico vai permitir no futuro a mitigação dos perigos associados ao processo de inspeção efetuado por Humanos, aumentando assim o nível de segurança e diminuindo significativamente o tempo despendido numa inspeção. Nos dias de hoje, muitas são as empresas que recorrem à utilização de drones para efetuar este tipo de manobras mas ainda de forma manual (teleoperação). No entanto, tem-se vindo a comprovar que a utilização de drones apesar de efetuar uma inspeção com menor tempo e com melhor qualidade (imagens com elevada resolução e possibilidade de análise em *backoffice*) quando comparado com o ser humano, apresenta uma autonomia baixa, sendo necessária a troca de bateria se se pretende inspecionar um parque eólico. Assim, de modo a a mitigar essa limitação durante o processo de inspeção, foi proposto nesta dissertação a utilização de um VTOL para a realização de inspeções e o desenvolvimento de um algoritmo optimizado de inspeção de ativos elétricos, no caso particular das eólicas. Foi efetuado um estudo sobre os diferentes tipos de veículo existentes e foi implementada uma solução.

De seguida, foi necessário efetuar um estudo relacionado com os *firmwares* existentes para se poder efetuar o controlo do veículo. Assim, estudou-se o Ardupilot e o PX4, tendo sido utilizado o PX4 nesta dissertação. Após uma breve comparação entre os dois, foi possível verificar que o desenvolvimento do Ardupilot, para VTOL, se encontra parado, estando limitadas as funções que se podem utilizar.

Foram também estudadas diferentes abordagens para o planeamento de uma trajetória. Foi estudado, de uma forma aprofundada, o algoritmo desenvolvido pela uni-

versidade ETH, o *Structural Inspection Path Planning*. Esta universidade pretendia desenvolver um algoritmo de fácil utilização por parte do utilizador. No entanto, a sua utilização não foi considerada fácil. O primeiro problema encontrado foi o tipo de *mesh* que o algoritmo aceitava. Esta *mesh* teria que ser constituída por triângulos, onde seria necessário ter em atenção o número total dos mesmos. Uma *mesh* com um elevado número de triângulos ou triângulos não visíveis resultava no não funcionamento do algoritmo. Após se conseguir obter uma *mesh* aceite pelo algoritmo, foi verificado que o mesmo fazia a criação de demasiados pontos para se efetuar a inspeção. Isto resultou num elevado número de imagens obtidas por este algoritmo, elevando também o tempo despendido para efetuar essa inspeção e o custo obtido. Assim, concluiu-se que este algoritmo não seria o mais indicado para o caso de uma estrutura como uma eólica.

Tendo em contas estes resultados, foi desenvolvido um algoritmo, que consistia na criação de retas entre o rotor/nacelle e as pás da ventoinha. Com este algoritmo conseguiu-se obter uma inspeção detalhada, realizado num curto espaço de tempo e obtendo um número considerável de imagens válidas. Este algoritmo permitiu também comprovar que a utilização de um VTOL para uma inspeção é a solução ideal, tendo em conta os tempo de voo efetuados pelo veículo em simulação.

Em suma, considera-se que os objetivos propostos para esta dissertação foram cumpridos com sucesso, faltando apenas um teste real de modo a validar o algoritmo desenvolvido. Esta validação não foi possível tendo em conta a situação atual vivida no país, o COVID 19.

8.1 Trabalho Futuro

Tendo em atenção os resultados obtidos nas simulações, o próximo passo considerado importante seria a realização de testes reais de modo a validar o algoritmo desenvolvido nesta dissertação.

Para além disso, seria interessante a adição de um *gimbal* à estrutura. Neste momento, para que o veículo efetue uma manobra de parafuso em torno da ventoinha, é necessário alterar o valor de *yaw* da câmara no modelo do veículo. Com um *gimbal* a câmara poderia estar constantemente a olhar para o ativo elétrico, sem ser necessário a alteração de parâmetros na definição da câmara.

Como linha de investigação, o trabalho de planeamento de trajetória deverá, no futuro, ser realimentado com a informação sensorial existente no veículo de modo a que o bloco de planeamento seja recalculado em tempo-real.

Bibliografia

[1] EDP. Parques eólicos. https://www.edp.com/pt-pt/parques-eolicos. Accessed: 2020-06-02.

[2] S. Verling, B. Weibel, M. Boosfeld, K. Alexis, M. Burri, and R. Siegwart. Full attitude control of a vtol tailsitter uav. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3006–3012, 2016.

[3] K. Benkhoud and S. Bouallègue. Model predictive control design for a convertible quad tilt-wing uav. In *2016 4th International Conference on Control Engineering Information Technology (CEIT)*, pages 1–6, 2016.

[4] Flickr. Canberra uav. https://www.flickr.com/photos/145621445@N03/30002865291/in/pool-canberrauav/. Accessed: 2020-06-19.

[5] sUAS News. Canberra rescue drone wins $10,000 prize. https://www.suasnews.com/2012/10/canberra-rescue-drone-wins-10000-prize/. Accessed: 2020-06-19.

[6] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, volume 114, page 649–666, 2016.

[7] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon "next-best-view"planner for 3d exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468, 2016.

[8] clipartkey. Yaw pitch roll. https://www.clipartkey.com/view/xwbiRh_yaw-pitch-roll/. Accessed: 2020-07-16.

[9] Keld Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, October 2000.

[10] Iram Noreen, Amna Khan, Khurshid Asghar, and Zulfiqar Habib. A path-planning performance comparison of rrt*-ab with mea* in a 2-dimensional environment. *Symmetry*, 11(7):945, Jul 2019.

[11] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart. Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6423–6430, May 2015.

[12] ArduPilot Mega. Ardupilot mega. `https://www.ardupilot.co.uk/`, 2020. Accessed: 2020-10-23.

[13] HobbyKing. Hkpilot 500mw transceiver radiotelemetria set v2 (433mhz). `https://hobbyking.com/pt_pt/hkpilot-500mw-transceiver-telemetry-radio-set-v2-433mhz.html`, 2020. Accessed: 2020-10-23.

[14] Drotek. Dp0303 (neo-m8t gps + lis3mdl compass xxl). `https://store-drotek.com/884-DP0303.html`, 2020. Accessed: 2020-10-23.

[15] Ardupilot. Common power module. `https://ardupilot.org/copter/docs/common-3dr-power-module.html`, 2020. Accessed: 2020-10-23.

[16] A. Kroll, W. Baetz, and D. Peretzki. On autonomous detection of pressured air and gas leaks using passive ir-thermography for mobile robot application. In *2009 IEEE International Conference on Robotics and Automation*, pages 921–926, 2009.

[17] N. Metni and T. Hamel. A uav for bridge inspection: Visual servoing control law with orientation limits. In *Automation in Construction*, volume 17, pages 3–10, November 2007.

[18] A. Banaszek, S. Banaszek, and A. Cellmer. Possibilities of use of uavs for technical inspection of buildings and constructions. *IOP Conference Series: Earth and Environmental Science*, 95:032001, December 2017.

[19] J. Choi, C. M. Yeum, S. J. Dyke, M. R. Jahanshahi, F. Pena, and G. W. Park. Machine-aided rapid visual evaluation of building façades. In *9th European Workshop on Structural Health Monitoring, Manchester, United Kingdom*, July 2018.

[20] J. M. Teixeira, R. Ferreira, M. Santos, and V. Teichrieb. Teleoperation using google glass and ar, drone for structural inspection. In *2014 XVI Symposium on Virtual and Augmented Reality*, pages 28–36, 2014.

[21] A. Davids. Urban search and rescue robots: from tragedy to technology. *IEEE Intelligent Systems*, 17(2):81–83, 2002.

[22] A. Jacoff, E. Messina, B. A. Weiss, S. Tadokoro, and Y. Nakagawa. Test arenas and performance metrics for urban search and rescue robots. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 4, pages 3396–3403 vol.3, 2003.

[23] Types of drones – explore the different models of uav's. `http://www.circuitstoday.com/types-of-drones`. Accessed: 2020-06-01.

[24] Flying model aircraft comes under scrutiny after fatal accident in brooklyn park. `https://www.nytimes.com/2013/09/07/nyregion/flying-model-aircraft-comes-under-scrutiny-after-fatal-accident-in.-brooklyn-park.html`. Accessed: 2020-06-01.

[25] Aerial mapping - ardupilot. `https://ardupilot.org/index.php/casestudies/case-aerial-mapping`. Accessed: 2020-06-01.

[26] Pictures of a thundertiger raptor e700 - ardupilot. `https://ardupilot.org/index.php/slider/Ardupilot-Helis/luis`. Accessed: 2020-06-01.

[27] Vtol search rescue - ardupilot. `https://ardupilot.org/index.php/casestudies/case-vtol-search-rescue`. Accessed: 2020-06-01.

[28] NOCTULA Consultores em Ambiente. Um dia na vida de um técnico de manutenção de aerogeradores. `https://noctula.pt/vida-um-tecnico-manutencao-aerogeradores/`. Accessed: 2020-06-02.

[29] Great Big Story. Climbing 300-foot wind turbines for a living. `https://www.youtube.com/watch?time_continue=332&v=xUjCD-fFU9k&feature=emb_title`, January 2017.

[30] Force Technology. Drone inspection of wind turbines – on- and offshore. `https:https://forcetechnology.com/en/services/drone-inspection-of-wind-turbines-onshore-and-offshore`. Accessed: 2020-06-03.

[31] EDP. Eólicas em alto mar. `https://www.edp.com/pt-pt/partilha-do-conhecimento/eolicas-em-alto-mar`. Accessed: 2020-10-20.

[32] H. Ferreira, C. Almeida, A. Martins, J. Almeida, N. Dias, A. Dias, and E. Silva. Autonomous bathymetry for risk assessment with roaz robotic surface vehicle. In *OCEANS 2009-EUROPE*, pages 1–6, 2009.

[33] C. Almeida, T. Franco, H. Ferreira, A. Martins, R. Santos, J. M. Almeida, J. Carvalho, and E. Silva. Radar based collision detection developments on usv roaz ii. In *OCEANS 2009-EUROPE*, pages 1–6, 2009.

[34] H. Ferreira, A. Martins, A. Dias, C. Almeida, J. M. Almeida, and E. Silva. Roaz autonomous surface vehicle design and implementation. January 2006.

[35] Strongmar. Turtle. `http://www.strongmar.eu/site/album/turtle-819`. Accessed: 2020-06-03.

[36] LSA. Falcos. `http://lsa.isep.ipp.pt/falcos_home.html`. Accessed: 2020-06-03.

[37] A. Martins, J. M. Almeida, C. Almeida, A. Figueiredo, F. Santos, D. Bento, H. Silva, and E. Silva. Forest fire detection with a small fixed wing autonomous aerial vehicle. In *International Autonomous Vehicles Conference, Toulouse, France*, September 2007.

[38] Strongmar. Roaz ii. `http://www.strongmar.eu/site/album/roaz-ii-783`. Accessed: 2020-06-03.

[39] LSA. Falcos. `http://lsa.isep.ipp.pt/~aprender/pagina_lsa/falcos/index.html`. Accessed: 2020-06-03.

[40] P. Sousa, A. Ferreira, M. Moreira, T. Santos, A. Martins, A. Dias, J. Almeida, and E. Silva. Isep/inesc tec aerial robotics team for search and rescue operations at the eurathlon challenge 2015. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 156–161, 2016.

[41] X. Lyu, H. Gu, Y. Wang, Z. Li, S. Shen, and F. Zhang. Design and implementation of a quadrotor tail-sitter vtol uav. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3924–3930, 2017.

[42] S. Verling, T. Stastny, G. Bättig, K. Alexis, and R. Siegwart. Model-based transition optimization for a vtol tailsitter. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3939–3944, 2017.

[43] H. Gu, X. Lyu, Z. Li, S. Shen, and F. Zhang. Development and experimental verification of a hybrid vertical take-off and landing (vtol) unmanned aerial vehicle(uav). In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 160–169, 2017.

[44] G. Flores, I. Lugo, and R. Lozano. 6-dof hovering controller design of the quad tiltrotor aircraft: Simulations and experiments. In *53rd IEEE Conference on Decision and Control*, pages 6123–6128, 2014.

[45] C. Papachristos, K. Alexis, and A. Tzes. Design and experimental attitude control of an unmanned tilt-rotor aerial vehicle. In *2011 15th International Conference on Advanced Robotics (ICAR)*, pages 465–470, 2011.

[46] Y. O. Aktas, U. Ozdemir, Y. Dereli, A. F. Tarhan, A. Cetin, A. Vuruskan, B. Yuksek, H. Cengiz, S. Basdemir, M. Ucar, M. Genctav, A. Yukselen, I. Ozkol, M. O. Kaya, and G. Inalhan. A low cost prototyping approach for design analysis and flight testing of the turac vtol uav. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1029–1039, 2014.

[47] J. Zhang, Z. Guo, and L. Wu. Research on control scheme of vertical take-off and landing fixed-wing uav. In *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pages 200–204, 2017.

[48] D. Orbea, J. Moposita, W. G. Aguilar, M. Paredes, R. P. Reyes, and L. Montoya. Vertical take off and landing with fixed rotor. In *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–6, 2017.

[49] J. Roberts, D. Frousheger, B. Williams, D. Campbell, and R. Walker. How the outback challenge was won: The motivation for the uav challenge outback rescue, the competition mission, and a summary of the six events. *IEEE Robotics Automation Magazine*, 23(4):54–62, 2016.

[50] I. Noreen, A. Khan, and Z. Habib. Optimal path planning using rrt* based approaches: A surveyand future directions. In *(IJACSA) International Journal of Advanced Computer Science and Applications*, volume 7, 2016.

[51] P. Pharpatara, B. Hérissé, and Y. Bestaoui. 3-d trajectory planning of aerial vehicles using rrt*. *IEEE Transactions on Control Systems Technology*, 25(3):1116–1123, 2017.

[52] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1476–1483, 2016.

[53] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. In *Auton Robot 34*, page 189–206, 2013.

[54] X. Luo, X. Li, Q. Yang, F. Wu, D. Zhang, W. Yan, and Z. Xi. Optimal path planning for uav based inspection system of large-scale photovoltaic farm. In *2017 Chinese Automation Congress (CAC)*, pages 4495–4500, 2017.

[55] B. E. Schäfer, D. Picchi, T. Engelhardt, and D. Abel. Multicopter unmanned aerial vehicle for automated inspection of wind turbines. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 244–249, 2016.

[56] K. Helsgaun. Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. In *Autonomous Robots, Springer US*, page 1059–1078, November 2015.

[57] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, and TaeHoon Lim. *ROS Robot Programming. From the basic concept to practical programming and robot application.* ROBOTICS Co., Ltd., 2017.

[58] Lentin Joseph. *Robot Operating System for Absolute Beginners. Robotics Programming Made Easy.* Apress, 2018.

[59] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, January 2009.

[60] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. October 2006.

[61] Wolfram Math World. Euler angles. `https://mathworld.wolfram.com/EulerAngles.html`. Accessed: 2020-07-16.

[62] Routific. Understanding the travelling salesman problem (tsp). `https://blog.routific.com/travelling-salesman-problem`. Accessed: 2020-07-20.

[63] Martin Grötschel and Olaf Holland. Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, 51:141–202, July 1991.

[64] David Applegate, Robert Bixby, Vasek Chvátal, and Barrie Cook. Finding cuts in the tsp (a preliminary report). August 1994.

[65] V. J. Sudhakar and V. Navaneetha Kumar. A new approach to solve the classical symmetric traveling salesman problem by zero suffix method. volume 6, pages 1111 – 1120, 2011.

[66] Hongwei Mo and Lifang Xu. Biogeography migration algorithm for traveling salesman problem. volume 4, pages 311–330, 2011.

[67] C. Yang and K. Y. Szeto. Solving the traveling salesman problem with a multi-agent system. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 158–165, 2019.

[68] Nicos Christofides and Samuel Eilon. Algorithms for large-scale travelling salesman problems. *Journal of the Operational Research Society*, 23:511–518, December 1972.

[69] Paul's Online Notes. Section 8-1 : Boundary value problems. `https://tutorial.math.lamar.edu/classes/de/BoundaryValueProblem.aspx`. Accessed: 2020-08-04.

[70] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. In *Robotics:Science and Systems 2010*, volume 30, pages 846–894, June 2011.

[71] A. Bircher, M. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart. An effective implementation of the lin-kernighan traveling salesman heuristic. In *European Journal of Operational Research*, volume 126, pages 106–130, October 2000.

[72] PX4 Dev Team. 3dr pixhawk 1 flight controller. `https://docs.px4.io/master/en/flight_controller/pixhawk.html`, 2020. Accessed: 2020-10-23.

[73] PX4 Dev Team. mro pixhawk flight controller. `https://docs.px4.io/master/en/flight_controller/mro_pixhawk.html`, 2020. Accessed: 2020-10-23.

[74] Unmanned Tech. Ardupilot apm 2.8 flight controller board. https://www.unmannedtechshop.co.uk/product/ardupilot-apm-2-8-flight-controller-board/, 2020. Accessed: 2020-10-23.

[75] Sensirion. Differential pressure sensor sdp3x. https://www.sensirion.com/en/flow-sensors/differential-pressure-sensors/worlds-smallest-differential-pressure-sensor/, 2020. Accessed: 2020-10-23.

[76] Drotek Electronics. Digital differential airspeed sensor kit. https://store-drotek.com/793-digital-differential-airspeed-sensor-kit-.html, 2020. Accessed: 2020-10-23.

[77] Frsky. Xjt. https://www.frsky-rc.com/product/xjt-2/, 2020. Accessed: 2020-10-23.

[78] Frsky. Lr9. https://www.frsky-rc.com/product/l9r/, 2020. Accessed: 2020-10-23.

[79] Academo. 2d rotation about a point. https://academo.org/demos/rotation-about-point/, 2020. Accessed: 2020-10-03.