INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia Electrotécnica

ISEL

Sistema de Visão para Aterragem Automática de UAV

NUNO ALEXANDRE ANTUNES MARTINS PESSANHA SANTOS

(Mestre)

Dissertação para obtenção do grau de Mestre em

Engenharia Electrotécnica – Ramo de Automação e Eletrónica Industrial

Orientadores:

Prof. Doutor Victor José de Almeida e Sousa Lobo
Prof. Doutor Fernando Manuel Fernandes Melício

Júri:

Presidente: Prof. Doutor José Manuel do Valle Cardoso Igreja
Vogais:
Prof. Doutor Fernando Manuel Fernandes Melício
Prof. Doutor Alexandre José Malheiro Bernardino

Setembro de 2014

Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia Electrotécnica

Área Departamental de Engenharia de
Sistemas de Potência e Automação

Sistema de Visão para Aterragem Automática de UAVs

Abordagem baseada num sistema localizado em terra

Versão Final

AGRADECIMENTOS

Neste espaço vou agradecer a todos aqueles que considero que tiveram um contributo essencial na elaboração desta tese, não vou agradecer por ordem de importância porque para mim todos foram importantes de alguma forma – todas as peças de um puzzle são essenciais para o poder terminar.

Devo aqui agradecer à oficina de Máquinas da Escola Naval por todo o apoio prestado, demonstrando sempre uma enorme prontidão nos trabalhos elaborados.

Ao Eng. Pedro Felicidade por ter estado sempre disponível para me apoiar, demonstrando uma enorme compreensão durante todo o período de elaboração deste trabalho.

Ao Prof. Doutor Fernando Melício por ter aceitado a orientação deste trabalho, respondendo positivamente a um desafio lançado por parte do Centro de Investigação Naval da Marinha de Guerra Portuguesa.

Ao Prof. Doutor Alexandre Bernardino por todo o apoio na área da visão artificial que me proporcionou, o que me levou sem dúvida a elaborar um trabalho mais completo. Sendo quase que um "pai" para mim nesta área, tendo-me ensinado bastante. Faço votos para que continuemos a trabalhar em conjunto num futuro próximo.

Ao Eng. Mário Marques gestor do projeto (Marinha de Guerra Portuguesa) em que esta tese se inseriu por ter sempre procurado garantir que tudo corresse bem, e nos prazos estabelecidos, o que tendo em conta o contexto existente a tarefa não foi (e não é) nada fácil.

Ao Eng. Tiago Ramalho por toda a vontade que sempre demonstrou em que tudo corresse bem, e por todo o apoio prestado.

Por fim, mas não menos importante, queria agradecer ao Prof. Doutor Victor Lobo por todo o apoio demonstrado (que mais uma vez orienta um trabalho elaborado por mim), pois mais uma vez abriu-me as "portas" necessárias e deu-me todas as ferramentas que precisei para trabalhar, sendo também nele (mais uma vez) que está a origem do tema desta dissertação. Aqui fica mais uma vez um sentido muito obrigado e votos de enorme sucesso pessoal e profissional.

Nuno Alexandre Antunes Martins Pessanha Santos, Lisboa, Setembro de 2014

RESUMO

Neste estudo é proposto um sistema de visão para aterrar automaticamente um avião não tripulado (*Unmanned Aerial Vehicle* - UAV) comercialmente existente chamado AR4 num navio, sendo este sistema composto por uma simples câmara RGB (espectro visível). A aplicação prevê a sua colocação no convés de um navio para estimar a pose do UAV (posição 3D e orientação) durante o processo de aterragem. Ao utilizar um sistema de visão localizado no navio permite a utilização de um UAV com menos poder de processamento, reduzindo assim o seu tamanho e peso.

O método proposto utiliza uma abordagem baseada no modelo 3D do objeto em que é necessária a utilização do modelo CAD 3D do UAV. A pose é estimada utilizando uma arquitetura baseada num filtro de partículas. A implementação utilizada é baseada nas estratégias de evolução presentes nos algoritmos genéticos, evitando assim perda de diversidade nas possibilidades criadas. Também é implementada filtragem temporal entre *frames* - filtro de Kalman *unscented* - por forma a obter uma melhor estimativa de pose.

Os resultados mostram erros angulares e de posição compatíveis com o sistema de aterragem automática. O algoritmo é apropriado para aplicações em tempo real em *standard workstations*, com unidades de processamento gráfico.

O UAV vai operar de um navio patrulha pertencente à Marinha de Guerra Portuguesa, o que implica a capacidade de aterrar num navio de 27 metros de comprimento, 5,9 metros de boca, com uma zona de aterragem pequena e irregular de 5x6 metros localizada na proa do navio.

A implementação de um sistema completamente autónomo é muito importante em cenários reais, uma vez que estes navios têm uma guarnição limitada e os pilotos de UAV nem sempre se encontram disponíveis. Além disso, um sistema de visão é mais robusto em ambientes onde pode ocorrer empastelamento ao sinal GPS.

Palavras-Chave: Visão artificial, Estimação de pose 3D, Veículos Autónomos, Sistemas Militares, Filtro de Partículas.

ABSTRACT

In this study a vision system for autonomous landing of an existing commercial aerial vehicle (UAV) named AR4 aboard a ship, based on a single standard RGB digital camera is proposed. The envisaged application is of ground-based automatic landing, where the vision system is located on the ship's deck and is used to estimate the UAV pose (3D position and orientation) during the landing process. Using a vision system located on the ship makes it possible to use an UAV with less processing power, decreasing its size and weight.

The proposed method uses a 3D model based pose estimation approach that requires the 3D CAD model of the UAV. Pose is estimated using a particle filtering framework. The implemented particle filter is inspired in the evolution strategies present in the genetic algorithms avoiding sample impoverishment. Temporal filtering is also implemented between frames – unscented Kalman filter – in order to get a better pose estimation.

Results show that position and angular errors are compatible with automatic landing system requirements. The algorithm is suitable for real time implementation in standard workstations with graphical processing units.

The UAV will operate from the Portuguese Navy fast patrol boats (FPB), which implies the capability of landing in 27 m length, 5.9 m breadth vessels, with a 5x6 m small and irregular landing zone located at the boat´s stern.

The implementation of a completely autonomous system is very important in real scenarios, since this ships have only a small crew and UAV pilots are not usually available. Moreover a vision based system is more robust in an environment where GPS jamming can occur.

Keywords: Computer Vision, Model Based Pose Estimation, Autonomous Vehicles, Military Systems, Particle Filters.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS

et al. (*et aliae*):	E outros (para pessoas)
e.g. (*exempli gratia*):	Por exemplo
etc.(*et cetera*):	E outros (para coisas)
i.e. (*id est*):	Isto é
sui generis	Único no seu género
UAV	*Unmanned Aerial Vehicle*
VANT	Veículo Aéreo Não tripulado
RPAS	*Remotely Piloted Aircraft System*
MGP	Marinha de Guerra Portuguesa
EN	Escola Naval
3D	Três dimensões
2D	Duas dimensões
HD	*High Definition*
VTOL	*Vertical Take-Off and landing*
GPU	*Graphics Processing Unit*
CUDA	*Compute Unified Device Arquitecture*
Esq.	Esquerda
Dir.	Direita
DRAM	*Dynamic Random Access Memory*
PCIe	*Peripheral Component Interconnect Express*
COTS	*Commercial off-the-shelf*
GPF	*Gaussian Particle Filter*
PF	*Particle Filter*

KF	*Kalman Filter*
EKF	*Extended Kalman Filter*
UKF	*Unscented Kalman Filter*
UPF	*Unscented Particle Filter*
BB	*Bounding Box*
AR	*Aspect Ratio*
GbE	*Gigabit Ethernet*
LEA	Licença especial de aeronavegabilidade
AAN	Autoridade Aeronáutica Nacional
CAD	*Computer aided design*
ZEE	Zona Económica Exclusiva
CAD	*Computer Aided Design*
IR	*Infrared*
GPS	*Global Positioning System*

"*Se puderes olhar, vê.*

Se puderes ver, repara"

José Saramago

***(Livro "Ensaio sobre a cegueira", 1995*)**

CAPÍTULO I
INTRODUÇÃO

1.1. ENQUADRAMENTO

Nos últimos anos, a pesquisa sobre veículos não tripulados tem aumentado o número de aplicações civis e militares existentes. O requisito base para a maioria destes sistemas, e especialmente em ambiente militar, é a fiabilidade garantindo assim uma taxa de falhas muito baixa em condições normais de operação.

Portugal tem a décima maior zona económica exclusiva (ZEE) do Mundo o que faz com que seja difícil controlar os 41335 km^2 de águas territoriais que nos pertencem, com recursos limitados.

Nos dias de hoje, as lanchas de fiscalização desempenham um papel muito importante neste contexto, podendo o seu desempenho ser claramente aumentado através da utilização de veículos aéreos não tripulados (UAV[1]). Estes permitem a extensão das capacidades operacionais existentes, ao e.g. enviar imagens e vídeo georreferenciado em tempo real para e.g. uma lancha de fiscalização. Este tipo de navios tem uma tripulação pequena e os pilotos de UAVs nem sempre se encontram disponíveis, sendo a automação de todas as operações que necessitem de intervenção humana (e.g. aterragem) uma prioridade. A área disponível para aterragem neste tipo de navios é pequena e irregular com um tamanho de aproximadamente 5x6 metros (Figura 1).

Devido à especificidade da aplicação o UAV utilizado deve ser simples, fiável, pequeno e resistente sendo preferencialmente de baixo custo devido ao grande número necessário e risco operacional elevado existente na sua aplicação. Este além da capacidade de operar em condições atmosféricas adversas[2], deve também ter a capacidade de descolar e aterrar

[1] Designado também por VANT – Veículo aéreo não tripulado – ou pela expressão anglo-saxónica *Remotely Piloted Aircraft System* (RPAS).
[2] Comummente designado por sistemas para todo o ambiente - *all environment*.

numa pequena área - lancha de fiscalização. O sistema desenvolvido não se procurou centrar em nenhum tipo de UAV específico podendo ser utilizado com qualquer tipo de plataforma (bastando que tenhamos o seu modelo CAD[3] disponível), sendo que as características gerais do UAV utilizado (modelo AR4) durante este estudo (Figura 3 e Figura 4) se encontram descritas na Tabela 1. A aplicação que se pretende desenvolver está maioritariamente vocacionada para as lanchas de fiscalização existentes da classe Argos (27 m de comprimento, 5,9 m de boca máxima, 2,8 m de calado e com um deslocamento de 97 toneladas) e da classe Centauro (28,4 m de comprimento, 5,95 m de boca máxima, 2,8 m de calado e com um deslocamento de 98 toneladas) pertencentes à Marinha de Guerra Portuguesa (MGP).

ALCANCE	**Vídeo (Tempo real)**	10 km (linha de vista)
	Telemetria	20 km (linha de vista)
SENSORES		Câmara RGB[4] ou IR[5]
AUTONOMIA		2 a 3 horas
DESEMPENHO	**Velocidade mínima**	37 km/h
	Velocidade cruzeiro	55 km/h
	Velocidade máxima	90 km/h
	Ângulo de subida (máx)	30 graus
	Ângulo de descida (máx)	10 graus
PESO MÁXIMO À DESCOLAGEM		5 kg
LIMITES DE OPERAÇÃO	**Altitude máxima**	4000 m (nível médio do mar)
	Temperatura mínima	-10°C
	Temperatura máxima	$+55$°C
	Vento máximo	40 km/h
DIMENSÕES	**Envergadura**	180 cm
	Comprimento (nariz – cauda)	150 cm

Tabela 1 – Características gerais do UAV utilizado – AR4.

Em 2005 (Gonçalves-Coelho, Veloso et al. 2007) a MGP inicia diversos testes a um UAV de custo reduzido (normalmente designado por *Commertial-off-the-shelf*[6] - COTS), procurando assim desenvolver uma solução que reúna as condições necessárias para cumprir com os requisitos expostos anteriormente. Tendo sido estudado quais os requisitos necessários para o UAV poder aterrar com sucesso num espaço tão confinado e móvel.

[3] Abreviatura para a designação anglo-saxónica – *Computer aided design* – comummente traduzido para desenho assistido por computador.
[4] Sigla que designa *Red, Green and Blue* – Vermelho, verde e azul – e que neste contexto é utilizada para designar uma câmara do espectro visível.
[5] Sigla que designa *Infrared* ou infravermelhos.
[6] Nome atribuído ao material - *hardware* ou *software* - já desenvolvido e com venda livre ao público em qualquer loja física ou virtual (e.g. loja na internet).

Simultaneamente ao desenvolvimento do UAV de baixo custo foi também estudado qual o melhor modo de aterragem[7] (testado a configuração de: cabo horizontal, vertical, rede com diversas configurações, entre outras), tendo sido demonstrado que uma simples rede de polipropileno[8] (Figura 2) proporciona a elasticidade e resistência necessária obtendo altos níveis de sucesso para este caso (100% de sucesso em 24 testes efetuados com a configuração final encontrada).

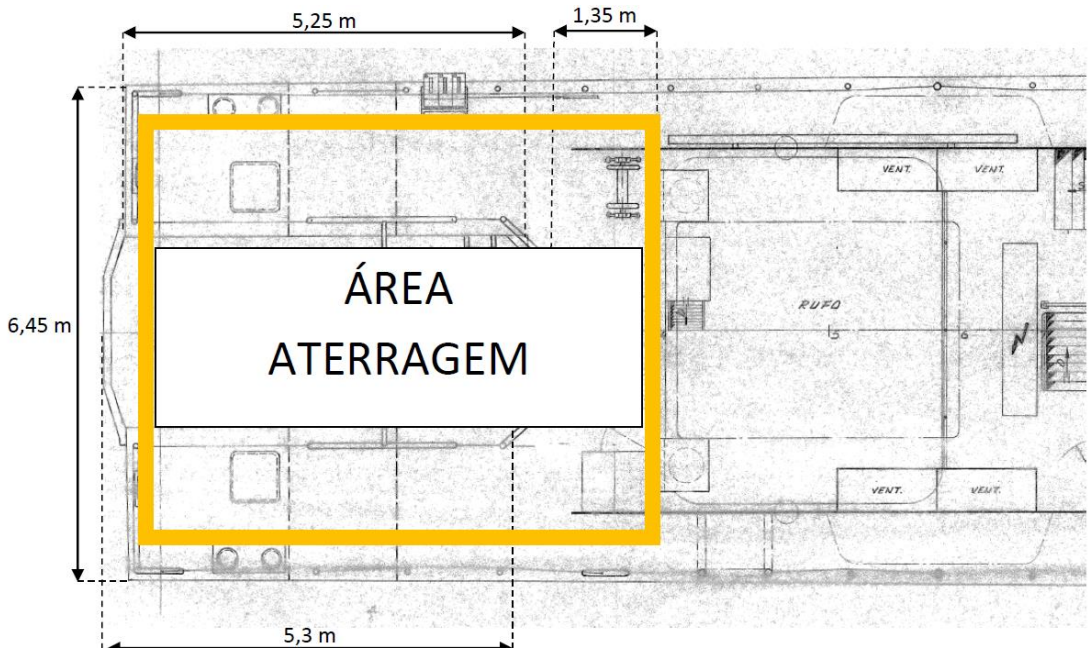


Figura 1 – Área de aterragem disponível.

Figura 2 – Testes de aterragem efetuados pela MGP.

A maior parte dos sistemas desenvolvidos nesta área é baseado em sistemas localizados a bordo dos UAVs (Shakernia, Ma et al. 1999, Saripalli, Montgomery et al. 2003, Merz,

[7] Um dos momentos mais perigosos no emprego de UAVs é a fase de recolha, pois é muito passível a erro e consequentemente acidente.
[8] Polímero derivado do propeno de forma molecular - C_3H_6.

Duranti et al. 2006, Williams and Crump 2012), não sendo normalmente contemplados os sistemas baseados em terra (Lange, Sunderhauf et al. 2009, Zhang, Shen et al. 2013). Ao usar um sistema localizado no navio é possível cumprir os requisitos necessários de peso e tamanho (ao não ser necessário adicionar poder de processamento extra – *hardware* -, o que iria aumentar o seu peso e dimensões).

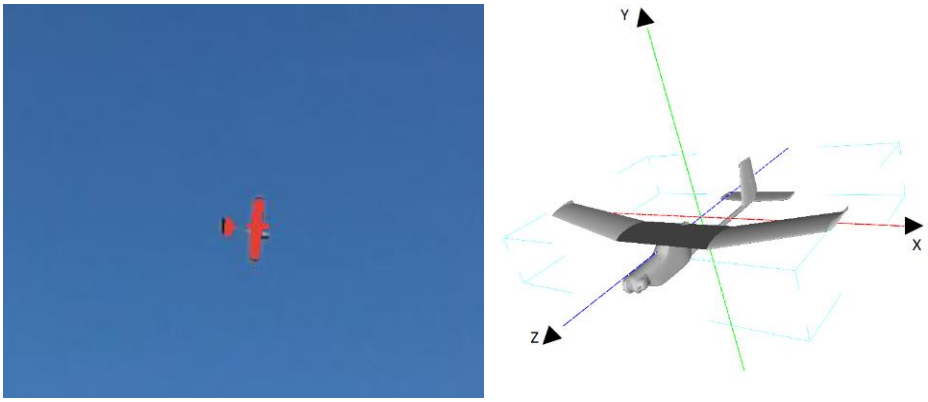


Figura 3 – Estimação de pose num cenário real (esquerda) e o modelo do UAV utilizado (direita).

Para o sistema de visão utilizado foi adotada uma abordagem baseada no modelo 3D do objeto (Figura 3 e Figura 4), sendo que nos dias de hoje todos os UAVs têm o seu modelo CAD disponível possibilitando assim a correta estimativa de pose (posição 3D e orientação) recorrendo a métodos de *tracking*[9] 3D baseados em filtros de partículas. Estes representam a distribuição da pose de um objeto como um conjunto de hipóteses com um determinado peso (partículas) (Doucet, de Freitas et al. 2001, Haug 2012). Estas hipóteses são testadas ao projetar explicitamente o modelo do objeto na imagem, com uma certa rotação e translação, comparando a informação de *pixels* obtida na imagem. O filtro de partículas em contraste com outros métodos como o filtro de Kalman pode ser utilizado em abordagens não-lineares (existem alguma variações como o filtro de Kalman estendido – e*xtended Kalman filter* – ou o filtro de Kalman *unscented* – *Unscented* Kalman *filter* - que preveem a sua utilização neste tipo de situações), ou em situações em que o sistema apresente distribuições multimodais típicas em estimação de pose (Lepetit and Fua 2005, Challa 2011, Forsyth and Ponce 2011).

A inicialização é um problema comum nas abordagens por filtro de partículas (Figura 6) quando nenhuma informação *a priori* sobre a localização inicial do objeto é disponibilizada,

[9] O termo *tracking* vai ser usado muitas vezes ao longo da presente dissertação, em detrimento da palavra seguimento ou outra semelhante.

sendo que neste caso normalmente as partículas são difundidas aleatoriamente no espaço em redor de uma posição específica (Forsyth and Ponce 2011). Este método é usualmente lento a convergir, afetando assim o desempenho e robustez do filtro. Uma outra solução é aumentar o número de partículas necessárias, mas isto necessita de maior poder de processamento o que é normalmente limitado. A informação proveniente do sistema de telemetria do UAV não foi utilizada para fazer inicialização do sistema de deteção pois este apresenta um erro elevado em altitude (por vezes da ordem das dezenas de metros – de acordo com testes práticos efetuados), optando nesta tese por uma abordagem simples e eficiente através da utilização de uma base de dados de poses do objeto conhecidas.

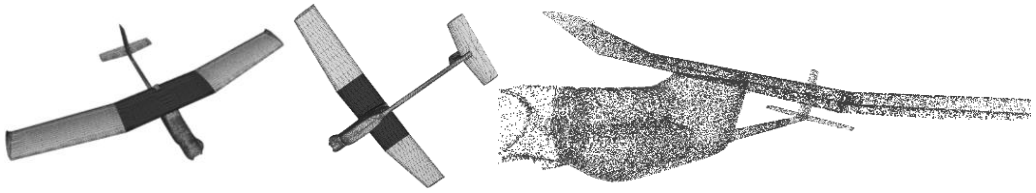


Figura 4 – Representação do modelo do UAV utilizado.

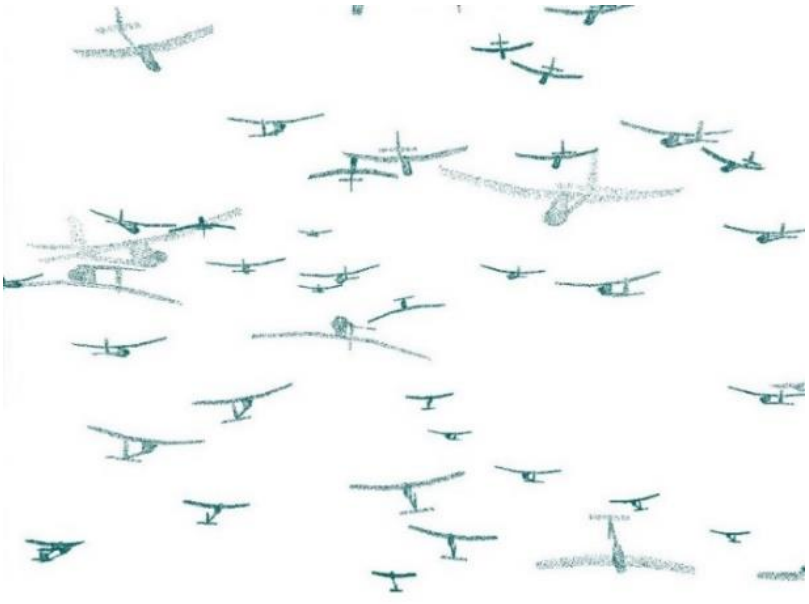


Figura 5 – Exemplo de múltiplas hipóteses (partículas) de pose (Rotação e Translação).

As estratégias de reamostragem utilizadas para as partículas geradas são baseadas nas estratégias de evolução presentes nos algoritmos genéticos (Boli, Djuri et al. 2004, Kwok, Fang et al. 2005, Park, Hwang et al. 2009), procurando assim evitar o empobrecimento da reamostragem (perda de diversidade) (Simon 2006).

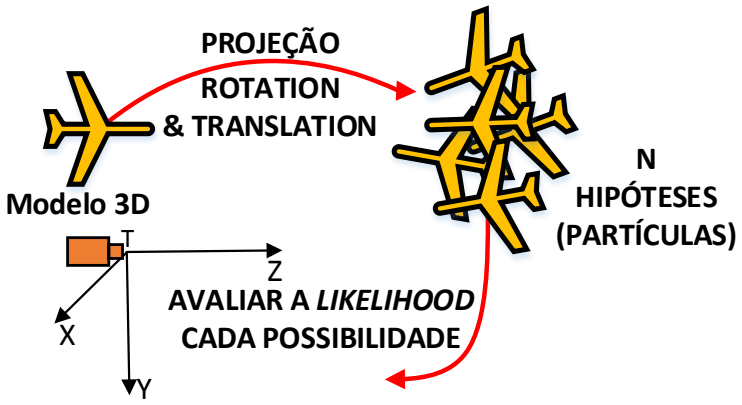


Figura 6 – Simplificação do funcionamento do filtro de partículas neste contexto de estudo.

Por forma a conseguir gerar partículas de forma eficiente foi também estudada a arquitetura *Compute Unified Device Arquitecture* – CUDA – que permitiu uma diminuição do tempo necessário por partícula. Foi também adaptada uma arquitetura de deteção de pose para fazer seguimento do UAV entre *frames* sucessivas – utilizando um filtro de Kalman *unscented* - permitindo assim uma melhor estimativa da pose apresentada entre *frames*. A Figura 7 representa um sistema Geral simplificado do sistema desenvolvido e que será explicado em detalhe ao longo da presente tese.

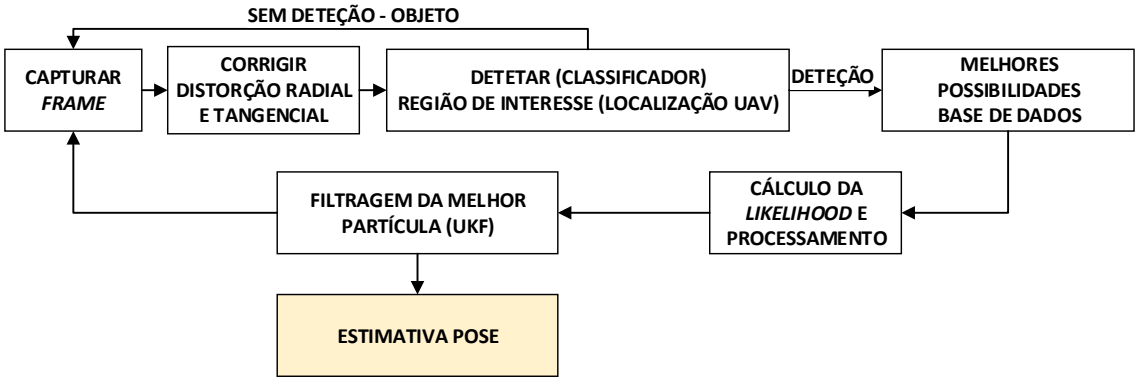


Figura 7 – Sistema desenvolvido – Diagrama Geral simplificado.

1.2. ORGANIZAÇÃO DA TESE

Esta tese está organizada em cinco capítulos distintos, que procuram evidenciar todo o trabalho que foi elaborado acerca desta temática.

No capítulo I além de ser efetuado um enquadramento ao tema de estudo é delineada a linha de ação deste estudo, referindo assim as diversas fases que foram seguidas neste estudo. É também referido o *software* e bibliotecas (ferramentas computacionais) mais importantes utilizadas, bem como as contribuições desta tese.

No capítulo II é efetuada uma revisão do estado da arte sobre o modelo geométrico da câmara, algoritmos de deteção de cantos, classificadores, filtro de partículas, filtro de partículas vs. Algoritmos genéticos, filtragem temporal (filtro de Kalman *unscented*), arquitetura CUDA e sistemas de aterragem atualmente existentes.

No capítulo III é totalmente dedicado a descrever a arquitetura do sistema utilizada, dividindo a sua explicação por fases para tornar a sua compreensão mais fácil.

No capítulo IV são abordados os testes efetuados para quantificar qual o desempenho apresentado pelos diversos constituintes do sistema e do sistema como um todo chegando às conclusões que se encontram descritas no capítulo V e último. Neste último capítulo é efetuada uma conclusão a todo o trabalho desenvolvido, indicando também as limitações encontradas e são apresentadas algumas sugestões para trabalho futuro.

1.3. METODOLOGIA UTILIZADA

O problema central deste estudo baseia-se na necessidade de estimar em tempo real a pose de um UAV, por forma a possibilitar que a fase de aterragem seja feita de uma forma automática. Assim definiu-se um método científico que garantisse a maior fiabilidade e conclusões válidas, estruturado de forma a obter um sistema com a maior fiabilidade possível, que seja capaz de cumprir a tarefa para que se destina.

Figura 8 – Esquema demonstrativo do método científico.

O método científico utilizado divide-se em três fases diferentes tendo cada uma delas uma diferente função no estudo (Azevedo 2006, Sarmento 2008, Martins Junior 2013). A primeira fase designou-se como explorativa, no seu âmbito definiu-se o problema em estudo, as questões de investigação, os objetivos, os conhecimentos e competências a

adquirir, as hipóteses a estudar e as metodologias a seguir nas diferentes fases do estudo. Na segunda fase - fase analítica -, procedeu-se à recolha de dados, através de uma metodologia de investigação explorativa no que concerne aos fundamentos teóricos e com uma metodologia de investigação experimental, que se fundamenta na experimentação. Na terceira fase, a fase final, designada por conclusiva, procedeu-se à confirmação das hipóteses e verificação dos objetivos alcançados que culminaram nas conclusões e recomendações.

1.3.1. FASE EXPLORATIVA

Nesta fase, como já foi referido estruturou-se a questão em estudo nesta dissertação de mestrado, definindo o problema, as questões relevantes e os objetivos. Foram, ainda, definidos os conhecimentos e competências a adquirir, bem como as hipóteses a considerar (Azevedo 2006, Sarmento 2008, Martins Junior 2013).

Este estudo surge na sequência de um projeto do Quadro de Referência Estratégico Nacional (QREN), denominado AUTOLAND (número de projeto - 23260) onde é contemplado o desenvolvimento de um sistema de visão (Figura 9) para estimar a pose de pequenos UAVs de asa fixa efetuando assim uma aterragem automática em plataformas móveis (para ser aplicado neste caso concreto de estudo a navios).

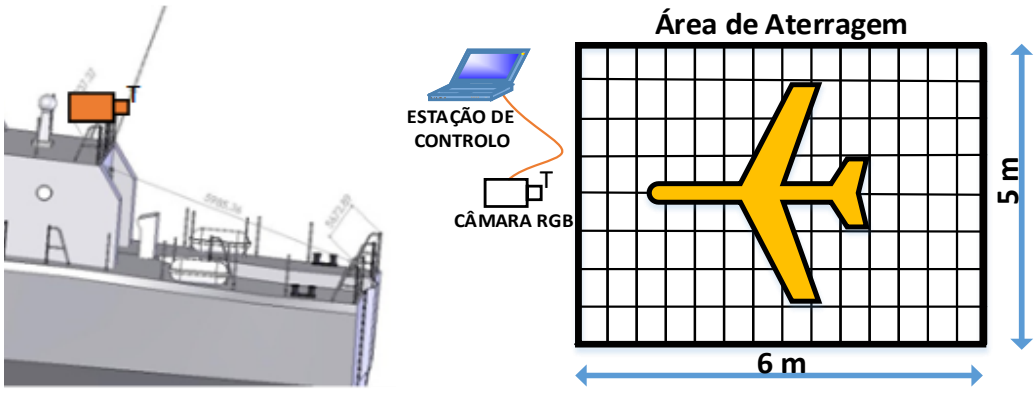


Figura 9 – Ilustração do sistema final a desenvolver.

Assim, para responder ao objetivo principal desta dissertação de mestrado foram colocadas diversas questões, entre elas:

- É possível efetuar a aterragem automática recorrendo a um sistema de visão?

- Que tipos de câmara (Infravermelhos, visível, entre outras) vão ser utilizadas?
- Qual o protocolo de comunicação a utilizar pela câmara?
- Qual a melhor abordagem: um sistema de visão localizado no navio ou na aeronave?
- Qual a resolução da câmara a utilizar?
- Qual a *frame rate*[10] mínima necessária?
- A câmara utilizada é adequada para aplicações no exterior?
- Que tipos de abordagens a este tipo de problemas existem?
- Quais as tolerâncias de erro em rotação e translação permitidas?
- É garantido um erro de translação máximo de 1 metro a 5 metros?
- Quais as dimensões do UAV a utilizar? A dimensão é adequada para uma deteção eficaz a pelo menos 50 metros de distância?
- Os métodos a aplicar são robustos o suficiente para garantir uma elevada fiabilidade no processo?
- Conseguimos detetar o objeto com diferentes rotações, translações e condições de iluminação?
- Os métodos aplicados são adequados (tempo de processamento) para podermos obter resultados em tempo real?
- Qual a melhor arquitetura a utilizar para o sistema?

Definiu-se, ainda, os conhecimentos e competências a adquirir para responder da melhor forma aos objetivos definidos anteriormente:

- Modelo geométrico da câmara – parâmetros intrínsecos e extrínsecos, distorção e calibração;
- Métodos de deteção de cantos;
- Classificador;
- Métodos de estimação de pose;
- Filtros de partículas;
- Métodos de filtragem temporal (sistemas não-lineares);
- Visão estereoscópica;
- Filtragem temporal;
- Visão monocular utilizando modelo 3D;

[10] Designado também por cadência, sendo esta uma medida do número de imagens que um determinado dispositivo ótico ou eletrónico processa por unidade de tempo.

- Sistemas atualmente existentes;
- Arquitetura CUDA;
- Conhecimentos de programação em C/C++ gerais, bem como a utilização de bibliotecas específicas para o efeito (conforme descrito no subcapítulo 1.4.).

Na revisão do estado da arte - no âmbito dos tópicos anteriores - de forma a retirar conclusões válidas, definiu-se uma metodologia qualitativa, onde se pretende conhecer o processo, através de análise documental de registos organizacionais, relatórios, normativos, estudos e outras publicações (Azevedo 2006, Sarmento 2008, Martins Junior 2013). Podendo assim construir uma base teórica que sustente as hipóteses, que de seguida se apresentam.

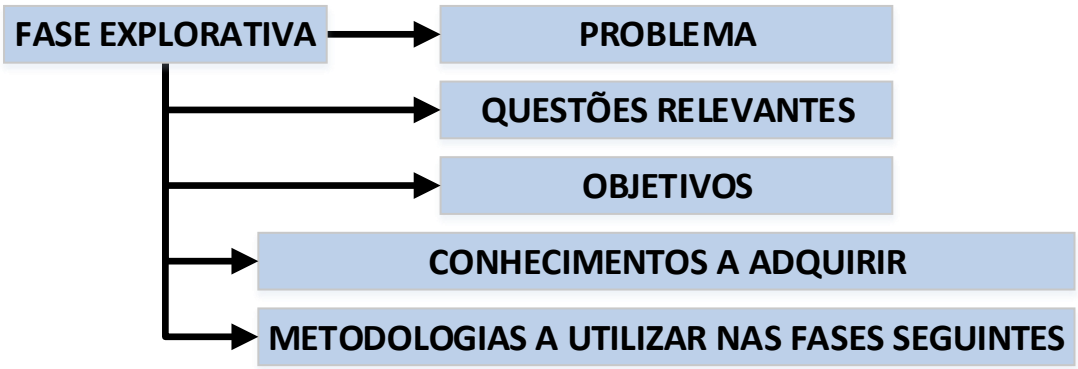


Figura 10 – Esquema demonstrativo da fase explorativa.

Nesta fase inicial de estruturação, definiram-se hipóteses a validar através de testes efetuados em imagens (*frames*[11]) e vídeos sendo que estes foram obtidos em cenário real ou gerados de forma sintética), recorrendo a uma metodologia demonstrativa, foi possível verificar se a arquitetura adotada para o sistema se adequa ao cenário em questão, através da exploração das ferramentas computacionais e da câmara a utilizar:

- Verificar se é possível adquirir imagens com a resolução necessária e em ambiente naval;
- A resolução da câmara adquirida é suficiente;
- O *software* disponível é adequado para a resolução da situação de estudo?;
- Utilização do sistema desenvolvido em imagens sintéticas, ou seja, analisar a performance do sistema desenvolvido em "laboratório";

[11] O termo *frame* é utilizada neste contexto para designar uma imagem capturada por uma câmara.

- Utilização do sistema desenvolvido em cenário real, ou seja, analisar a performance do sistema desenvolvido num caso real;
- Determinar o erro quantitativo obtido em translação e rotação, bem como dos diversos constituintes das arquiteturas desenvolvidas;
- Determinar se as arquiteturas desenvolvidas permitem a *frame* rate necessária, para poder estimar a pose do UAV em tempo real.

1.3.2. FASE ANALÍTICA

Nesta segunda fase do estudo, designada como analítica, concretizou-se o definido na fase anterior, no que concerne à revisão do estado da arte e à confirmação das hipóteses através de testes à arquitetura desenvolvida com base nas competências adquiridas durante a revisão do estado da arte (Azevedo 2006, Sarmento 2008, Martins Junior 2013).

Após a concretização da revisão do estado da arte e já com competências adquiridas, procedeu-se à escolha do tipo de sistema a desenvolver e quais as ferramentas necessárias. Escolheu-se então um sistema de visão monocular, baseado numa simples câmara *Gigabit Ethernet* – GbE - (com proteção IP67, devido ao ambiente extremamente adverso a que está sujeita) e um computador com uma placa gráfica que permitisse o processamento recorrendo à tecnologia CUDA[12]. Assim, baseou-se a escolha na contraposição entre os parâmetros presentes nas características técnicas e os que se sabiam necessários para o sistema funcionar no contexto pretendido.

Para testes utilizando imagens sintéticas foi utilizado o *software Visual Studio*, e recorrendo ao modelo do objeto foram gerados vídeos e frames com a localização pretendida. Todos os testes efetuados procuraram avaliar o desempenho quantificando o erro obtido (rotação e translação) e a *frame* rate obtida, avaliando assim se o sistema é adequado para a situação em estudo.

Finalmente, procedeu-se à interpretação dos resultados experimentais verificados nos testes efetuados. Pôde concluir-se então que com a execução destes testes se garantiu neste estudo uma melhor interpretação do desempenho do sistema, tendo em conta a

[12] *Computer Unified Device Arquitecture* - é uma plataforma para processamento paralelo na GPU, criada pela empresa NVIDIA. A GPU - *Graphics Processing Unit* ou Unidade de processamento gráfico - é comummente designado por placa gráfica num computador.

situação de estudo e o *hardware* utilizado, conseguindo identificar falhas possibilitando uma maior compreensão do sistema.

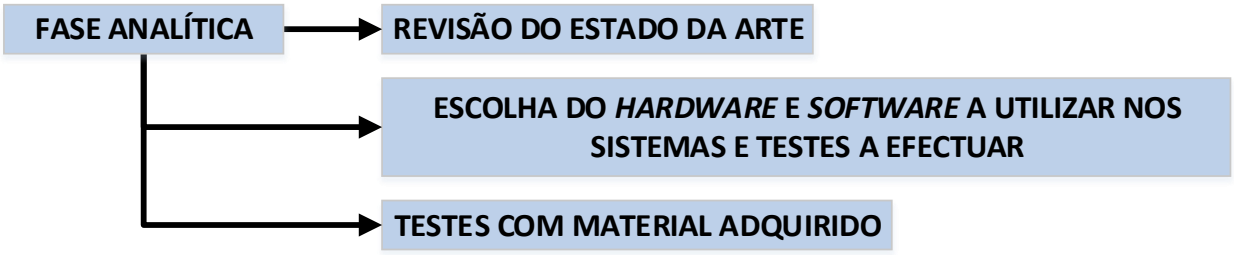


Figura 11 – Esquema demonstrativo fase analítica.

1.3.3. FASE CONCLUSIVA

Nesta última fase, confrontou-se os conhecimentos e competências adquiridos durante a revisão do estado da arte com os resultados dos testes práticos efetuados. Assim, através da discussão dos resultados, confirmaram-se as hipóteses colocadas na fase explorativa e verificaram-se os objetivos alcançados, dando então origem às conclusões e recomendações deste estudo, bem como a sugestões para futuras investigações (Azevedo 2006, Sarmento 2008, Martins Junior 2013).

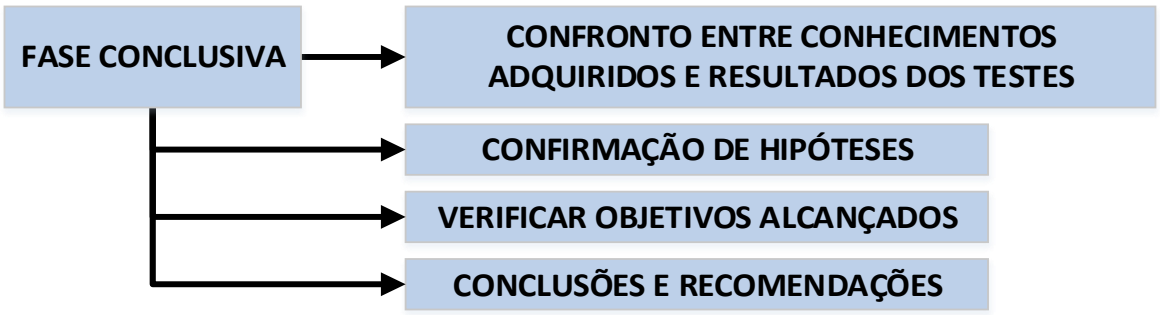


Figura 12 – Esquema demonstrativo fase conclusiva.

1.4. *SOFTWARE* E BIBLIOTECAS UTILIZADAS

O sistema foi implementado e desenvolvido em ambiente Windows (*Visual Studio* 2012 e 2013), usando a linguagem C e C++. A não familiaridade com muitas destas ferramentas, levou a que a curva de aprendizagem em certos casos fosse relativamente lenta. Adicionalmente, o *software* e bibliotecas mais importantes que foram utilizadas foram:

- ***OpenCV*** (http://opencv.org/) – Biblioteca *open source*[13] para processamento de imagem, calibração, entre outros;
- ***OpenGL*** (http://opengl.org/) – Biblioteca utilizada para efetuar a projeção (*rendering*) 2D e 3D pela GPU do modelo CAD utilizado;
- ***MeshLab*** (http://meshlab.sourceforge.net/) – *Software* utilizado para visualizar e editar o modelo CAD utilizado;
- ***Cloud Compare*** (http://www.danielgm.net/cc/) – Utilizado para editar e visualizar o modelo CAD utilizado, estando mais vocacionado para a manipulação dos pontos do modelo;
- ***MatLab*** (http://www.mathworks.com/products/matlab/) – *Software* utilizado para cálculo auxiliar e para verificação inicial de resultados;
- ***CGAL*** (https://www.cgal.org/) – Biblioteca utilizada para a execução de alguns algoritmos geométricos (e.g. *convex hull* ou *alpha shape*);
- ***Eigen*** (http://eigen.tuxfamily.org/index.php?title=Main_Page) – Biblioteca utilizada para álgebra linear, matrizes e vetores em alguns casos concretos;
- ***OpenTBB*** (http://www.threadingbuildingblocks.org/faq) – Biblioteca utilizada para paralelização na CPU;
- ***OpenMP*** (http://openmp.org/wp/) – Biblioteca utilizada para paralelização na CPU;
- ***CUDA Toolkit*** (https://developer.nvidia.com/cuda-toolkit) – Biblioteca utilizada para poder utilizar a GPU para efetuar processamento;
- ***GML Camera Calibration Toolbox*** (http://graphics.cs.msu.ru/en/node/909) – *Software* com interface gráfica para obter os parâmetros de distorção;
- ***3DF Lapyx*** (http://www.3dflow.net/products/3df-lapyx-camera-calibration-made-simple/) – *Software* semelhante ao do ponto anterior (utilizado para comparação de resultados).

[13] O conceito de *open source* está associado a uma filosofia de partilha global em que todo o conteúdo pode ser visualizado e alterado sem qualquer restrição, não devendo no entanto esquecer de referenciar os direitos de propriedade intelectual de terceiros.

1.5. CONTRIBUIÇÕES DA TESE

O trabalho efetuado procurou colmatar, como foi referido anteriormente, uma necessidade da Marinha de Guerra Portuguesa. Esta levou a que um sistema de visão para aterragem automática para aviões não tripulados fosse desenvolvido.

A principal contribuição da tese foi o aumento da capacidade em tempo real de estimar a pose (posição 3D e atitude) de um UAV, podendo utilizar a arquitetura desenvolvida estimar a pose de qualquer objeto bastando que para isso esteja disponível o seu modelo CAD 3D.

O desenvolvimento deste trabalho contribuiu também para a divulgação do nome do Instituto Superior de Engenharia de Lisboa e da Marinha de Guerra Portuguesa – Centro de investigação naval - através de participações em conferências, *workshops* e colóquios, em que foi exposto o trabalho desenvolvido nesta área.

CAPÍTULO II
CONCEITOS GERAIS

2.1. INTRODUÇÃO

Nos dias que correm, em que vivemos claramente numa era de informação, os dados são adquiridos e tratados muitas vezes em tempo real. O tratamento de dados em tempo real reveste-se de grande importância mas nem sempre foi possível, sendo o grande avançar tecnológico existente o grande impulsionador desta enorme capacidade. O presidente da *Intel Corporation*[14] Gordon E. Moore em 1965 fez mesmo uma previsão sobre a evolução do número de transístores presentes nos circuitos integrados produzidos pela sua empresa – lei de *Moore*- afirmando que o seu número iria aumentar em 100% a cada 18 meses. Muito curiosamente essa clara tendência de evolução tem-se mantido até aos dias de hoje (Bell, Gray et al. 2006). Este aumento da capacidade de processamento disponível ao público geral e não apenas a pequenas elites permite a execução de tarefas em tempo real pelo utilizador comum, o que veio aumentar o interesse nestas temáticas. Isto permitiu que o campo da visão artificial fosse alargando o número de seguidores, pois a maior parte das funções são computacionalmente complexas mas já acessíveis no poder de processamento que existe atualmente disponível ao utilizador comum.

Nos próximos subcapítulos vão ser abordados os conceitos relacionados com o modelo geométrico da câmara (abordando o seu funcionamento, os parâmetros intrínsecos, a distorção, a calibração e os parâmetros extrínsecos), visão estereoscópica (definindo o conceito de matriz fundamental), classificador (utilizado na segmentação de imagem), algoritmos de deteção de cantos, visão monocular e filtro de partículas (abordando também a relação entre os filtros de partículas e os algoritmos genéticos), filtragem temporal (filtro

[14] *Integrated Electronic Corporation* é uma empresa multinacional de origem Americana, normalmente designada apenas por *Intel.*

de Kalman *unscented*), a arquitetura CUDA e os sistemas de aterragem atualmente existentes procurando assim explorar os conceitos fundamentais das temáticas em estudo.

2.2. MODELO GEOMÉTRICO DA CÂMARA

É muito importante definir como o Mundo 3D é projetado no plano da imagem (plano 2D) obtido por uma câmara, sendo o modelo apresentado puramente geométrico (seguindo as regras da projeção perspetiva). Dado que a posição de um objeto no plano da imagem depende da sua posição no Mundo real, é necessário representar esta relação de forma sistemática Existem outros modelos para representar outros dispositivos óticos mais complexos como e.g. *fisheye lens* (lente olho de peixe) para obter um campo de visão amplo, sendo este estudo restrito ao modelo *pinhole* que melhor descreve o dispositivo de aquisição (câmara) que vamos utilizar neste contexto de estudo.

Este modelo geométrico é designado por câmara *pinhole* ou estenopeica[15], que fisicamente corresponde a uma caixa fechada com um pequeno buraco (Figura 14). As câmaras *pinhole* foram inventadas no século XVI, utilizando as leis da perspetiva descobertas um século antes por *Brunelleschi*. Em 1550 estas ganharam novas lentes (mais evoluídas), permitindo também gravar a luz que batia no seu plano da imagem (Figura 13) (Lepetit and Fua 2005, Forsyth and Ponce 2012). Foi assim possível estabelecer uma correspondência entre a área 2D no plano da imagem e a área no Mundo real (3D).

Os raios de luz refletidos ou emitidos pelo objeto passam por este buraco e formam uma imagem invertida (projeção[16], devido à maneira como os raios de luz atingem o plano da imagem) na parte de trás da caixa ou plano da imagem (2D). Para simplificação do conceito de projeção o modelo da imagem (representado como uma imagem virtual) é normalmente representado à frente do *pinhole*[17] (chamado de centro ótico – *optical center* – ponto onde os raios de luz convergem). Originando assim uma imagem não invertida (Figura 15) face à realidade, fisicamente não é possível construir uma câmara que opere desta forma mas a sua análise é em tudo equivalente ao modelo *pinhole* (Jähne, Haussecker et al. 1999, Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Bigun 2006,

[15] Nesta tese vai-se utilizar a designação *pinhole* em detrimento desta.
[16] Deriva da palavra em latim *projicere,* originada pela junção de *pro* – para a frente – e *jacere* – atirar.
[17] Designado nesta tese como modelo *pinhole* adaptado.

Hornberg 2007, Bradski and Kaehler 2008, Wöhler 2009, Cyganek and Siebert 2011, Forsyth and Ponce 2012, Prince 2012, Radke 2012).

Figura 13 – Formação de imagem na parte de trás de uma câmara (Pe and Carson 1969).

No modelo *pinhole* adaptado (Figura 15) o plano da imagem é criado ao longo do eixo *w* ou eixo ótico (ainda designado também por eixo principal), sendo o ponto de intersecção entre o eixo ótico e a imagem denominado por ponto principal. A distância entre o centro ótico (também designado por centro da câmara) e o ponto principal é denominado por distância focal.

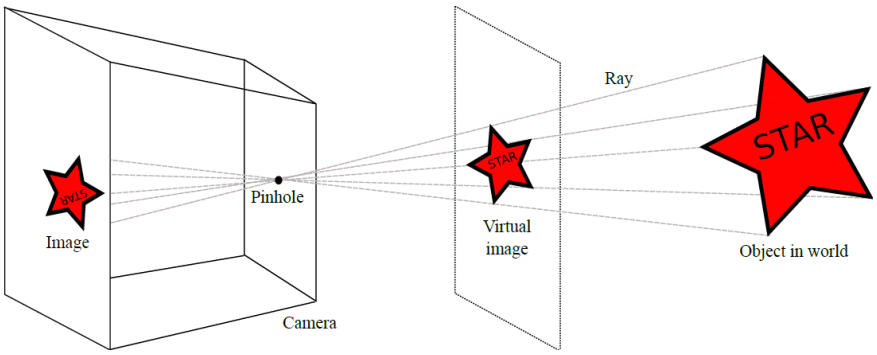


Figura 14 – Modelo de câmara *pinhole* (Prince 2012).

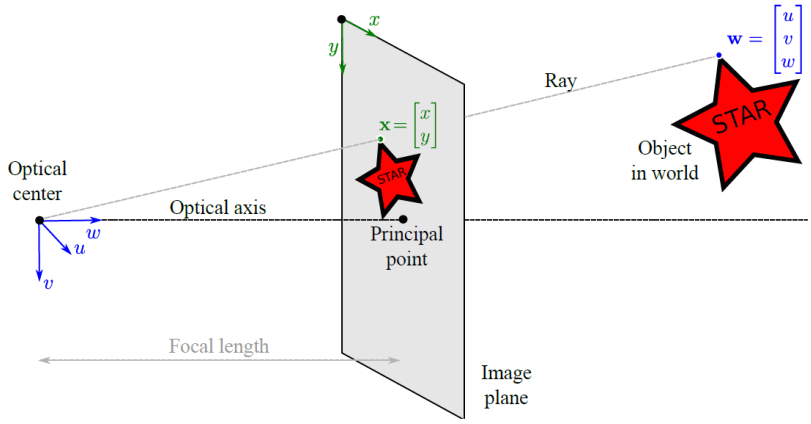


Figura 15 – Modelo de câmara *pinhole* adaptado (Prince 2012).

Através deste modelo podemos estabelecer a relação entre um determinado ponto no Mundo real ($W = [u, v, w]^T$) e a sua posição na imagem ($X = [x, y]^T$), sendo esse ponto representado por uma reta que parte do objeto, intersecta o plano da imagem e o centro ótico (Isto admitindo que o centro ótico está reduzido a um ponto, o que é fisicamente impossível, fazendo com que apenas um raio passe por cada ponto do plano da imagem). Mas as relações e particularidades existentes neste modelo vão ser abordadas nos subcapítulos que se seguem.

Neste caso concreto de estudo a câmara a utilizar é uma câmara RGB (*Red*, *Green and Blue* – espectro visível - Figura 16), sendo neste tipo de câmaras e as suas particularidades que este estudo se vai centrar. A nossa câmara é um dispositivo radiométrico que mede a energia da luz, brilho e cor (Figura 16). O espectro de cor RGB é normalmente representado por um cubo, mostrando todas as combinações de cor possíveis com as cores primárias (Vermelho, Verde e Azul). Os eixos são normalizados entre 0 e 1, podendo representar qualquer cor pela localização do seu ponto neste cubo (e.g. Ponto de cor [0,5;0,5;0,5]). Independentemente do espaço de cor escolhido deve ser possível distinguir claramente o objeto do fundo, por forma a efetuar facilmente esta distinção (objeto/fundo).

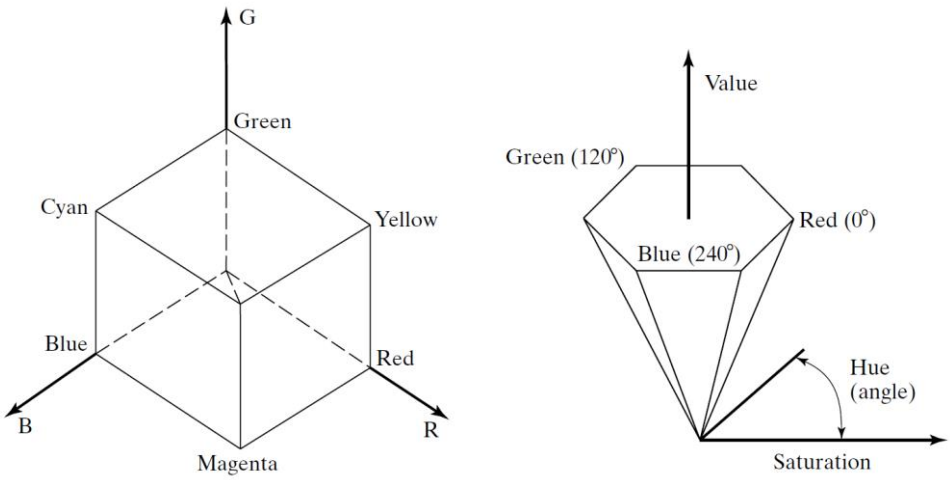


Figura 16 – Cubo RGB (esquerda) e Cone HSV (direita) (Forsyth and Ponce 2012).

Considerando um ponto no Mundo 3D com coordenadas $P_0 = [X_0, Y_0, Z_0]^T \epsilon R^3$, podemos relacionar estas coordenadas com as coordenadas da câmara da seguinte forma (Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Bradski and Kaehler 2008, Nixon 2008, Wöhler 2009, Cyganek and Siebert 2011):

$$P_{câmara} = RP_0 + T \tag{2.1}$$

com $P_{câmara} = [X, Y, Z]^T$. As matrizes R e T podem ser definidas da seguinte forma (Bigun 2006, Bradski and Kaehler 2008, Nixon 2008, Cyganek and Siebert 2011):

$$R = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{2.2}$$

$$T = O_c - O_W = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{2.3}$$

sendo R a matriz de rotação e T a matriz de translação que descreve a mudança entre a posição do centro de coordenadas da câmara (O_c) e o centro de coordenadas do Mundo (O_W). A rotação pode ser decomposta como a rotação segundo cada eixo, não sendo a ordem de rotação comutativa. As rotações podem ser representadas de variadíssimas maneiras, sendo que as representações que vão ser utilizadas nesta dissertação são rotações segundo os ângulos de Euler – matrizes de rotação - e quaterniões. Segundo cada eixo, e de acordo com os ângulos de Euler respetivos, as rotações podem ser representadas pelas seguintes matrizes homogéneas (Pervin and Webb 1982, Lepetit and Fua 2005, Bigun 2006, Bradski and Kaehler 2008, Nixon 2008, Cyganek and Siebert 2011):

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

com α a representar a rotação segundo X, β segundo Y e γ segundo (Figura 17). A ordem de rotação utilizada neste trabalho é a rotação segundo X, Y e Z (por esta ordem), sendo a matriz de rotação resultante calculada da seguinte forma (Pervin and Webb 1982, Lepetit and Fua 2005, Bigun 2006, Bradski and Kaehler 2008, Nixon 2008, Cyganek and Siebert 2011):

$$R_z R_y R_x = \begin{bmatrix} \cos\beta\cos\gamma & \cos\gamma\sin\alpha\sin\beta - \cos\alpha\sin\gamma & \cos\alpha\cos\gamma\sin\beta + \sin\alpha\sin\gamma & 0 \\ \cos\beta\sin\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma & -\cos\gamma\sin\alpha + \cos\alpha\sin\beta\sin\gamma & 0 \\ -\sin\beta & \cos\beta\sin\alpha & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

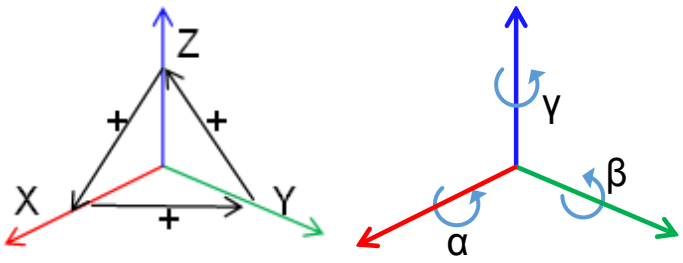


Figura 17 – Eixos de rotação (sentido positivo – regra da *mão direita*) – eixos ortogonais OXYZ.

A operação inversa de extrair os ângulos de Euler da matriz homogénea é igualmente possível. Esta representação apresenta uma grande desvantagem, quando em duas das três rotações existe o alinhamento entre eixos uma rotação deixa de ter efeito prático. Esta singularidade é conhecida como g*imbal lock* (Lepetit and Fua 2005, Van Verth and Bishop 2008, Szeliski 2010, Corke 2011, Radke 2012). Uma rotação no espaço 3D pode também ser representada recorrendo a quaterniões (Pervin and Webb 1982, Jähne, Haussecker et al. 1999, Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Trawny and Roumeliotis 2005, Szeliski 2010, Forsyth and Ponce 2012), estes são números híper-complexos[18], que podem ser escritos através da seguinte combinação linear:

$$a + bi + cj + dk \tag{2.8}$$

onde $i^2 = j^2 = k^2 = ijk = -1$. Os quaterniões podem também ser interpretados como a combinação de um escalar com um vetor de três dimensões, esta representação evita a ocorrência de g*imbal lock* desde que a norma do quaternião se mantenha unitária (mesmo quando se fazem aproximações ou estimativas). A passagem de ângulos de Euler para quaterniões é dada por:

$$Q_{Euler}(\alpha, \beta, \gamma) = q_\gamma q_\beta q_\alpha = q \tag{2.9}$$

$$q_\alpha = [\cos\tfrac{\alpha}{2}, \sin\tfrac{\alpha}{2}, 0, 0]^T \tag{2.10}$$

$$q_\beta = [\cos\tfrac{\beta}{2}, 0, \sin\tfrac{\beta}{2}, 0]^T \tag{2.11}$$

$$q_\gamma = [\cos\tfrac{\gamma}{2}, 0, 0, \sin\tfrac{\theta}{2}]^T \tag{2.12}$$

E como foi referido anteriormente a sua norma deve ser unitária, de acordo com:

$$|q| = \sqrt{q_s^2 + q_x^2 + q_y^2 + q_z^2} = 1 \tag{2.13}$$

[18] São extensões dos números complexos.

com q_s[19] a corresponder à parte real do quaternião. A conversão de quaterniões para ângulos de Euler é feita de acordo com a seguinte expressão:

$$a = A_q(q) = \frac{2\cos^{-1} q_s}{\sqrt{1-q_s^2}}[q_x, q_y, q_z]^T \tag{2.14}$$

Do modelo *pinhole* adaptado (Figura 15) podemos verificar que o ponto $P_{câmara}$ é projetado no plano da imagem tendo em conta a seguinte relação (Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Bigun 2006, Forsyth and Ponce 2012):

$$p = \begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{Z}\begin{bmatrix} X \\ Y \end{bmatrix} \tag{2.15}$$

Em coordenadas homogéneas, o modelo geométrico de uma câmara ideal é dado por (Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Bradski and Kaehler 2008, Nixon 2008):

$$\lambda\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}\begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} \tag{2.16}$$

Nos próximos subcapítulos vão ser descritos os parâmetros intrínsecos e extrínsecos da câmara, bem como a distorção existente e os processos de calibração para poder quantificar e corrigir, caso seja necessário, a distorção existente.

2.2.1. PARÂMETROS INTRÍNSECOS

Quando se captura uma imagem – *frame* - com uma câmara digital as dimensões são obtidas em termos de *pixels* (i, j), através da análise da Figura 15 é possível concluir como um ponto no espaço de coordenadas do Mundo real 3D é mapeado para um ponto no plano da imagem. A equação que representa essa relação é dada por (Luong and Faugeras 1996, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Hornberg 2007, Bradski and Kaehler 2008, Nixon 2008, Wöhler 2009, Szeliski 2010, Cyganek and Siebert 2011, Radke 2012):

$$p = (u, v)^T = (f\frac{X}{Z}, f\frac{Y}{Z})^T \tag{2.17}$$

[19] A designação q_s e q_w serão utilizadas ao longo da presente dissertação para designar a parte real do quaternião.

onde u e v representam as coordenadas 2D do ponto no plano da imagem, f a distância focal e X, Y e Z as coordenadas no Mundo real 3D. É possível reescrever a equação anterior recorrendo a coordenadas homogéneas, obtendo o seguinte:

$$p = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.18}$$

onde $\lambda = Z$ é o factor de escala homogêneo, e assumindo também que a origem é o ponto principal do plano da imagem. A origem do plano da imagem é normalmente o *pixel* com a coordenada mais à esquerda (Figura 18), sendo adicionado para este efeito um *offset.* Reescrevendo (2.18) tendo em conta este *offset* resulta no seguinte (Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Bradski and Kaehler 2008, Szeliski 2010, Cyganek and Siebert 2011, Forsyth and Ponce 2012, Prince 2012, Radke 2012):

$$p = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.19}$$

onde c_x e c_y são as coordenadas do ponto principal, considerando também que os *pixels* têm todos o formato quadrado (razão entre largura e altura unitária – Razão do aspeto ou *aspect ratio*[20]). Numa situação real, e dependendo do sensor utilizado (e.g. CCD[21]), os *pixels* normalmente não são quadrados apresentando dimensões distintas de altura e largura. Para contemplar este factor podemos reescrever (2.19) da seguinte forma (Okutomi and Kanade 1993, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Bradski and Kaehler 2008, Nixon 2008, Szeliski 2010, Cyganek and Siebert 2011, Forsyth and Ponce 2012, Prince 2012):

$$p = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f\eta_x & 0 & c_x & 0 \\ 0 & f\eta_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.20}$$

onde η_xe η_y representam o número de *pixels* por unidade de distância (admitindo que as coordenadas da imagem são medidas em *pixels*). Também existe um factor que deve ser contemplado que é o factor de *skew* (ângulo que contempla a inclinação da imagem, proporcional a $\cot(\theta)$ – sendo θ o ângulo que expressa a relação entre os eixos x e y do

[20] Define a relação matemática entre as duas dimensões constituintes do *pixel* – largura e altura.
[21] Iniciais que designam *Charge-coupled device,* este sensor é normalmente utilizado para câmaras compactas e de pequenas dimensões.

plano da imagem), sendo este factor desprezável na grande maioria das câmaras (Ma, Soatto et al. 2004). Contemplando este ângulo obtemos a seguinte expressão (Jähne, Haussecker et al. 1999, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Bradski and Kaehler 2008, Nixon 2008, Szeliski 2010, Cyganek and Siebert 2011, Forsyth and Ponce 2012, Prince 2012):

$$p = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f\eta_x & S_\theta & c_x & 0 \\ 0 & f\,\eta_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = [K|0_3]P \tag{2.21}$$

com K a representar a matriz de parâmetros intrínsecos (também designada por matriz de calibração), 0_3 é um vetor nulo de dimensão 3 e P é um ponto no Mundo real – espaço 3D e $S_\theta = fs_\theta$ sendo este um valor muito próximo de zero. Os parâmetros f, η_x, η_y, S_θ, c_x, c_y são denominados por: parâmetros intrínsecos da câmara.

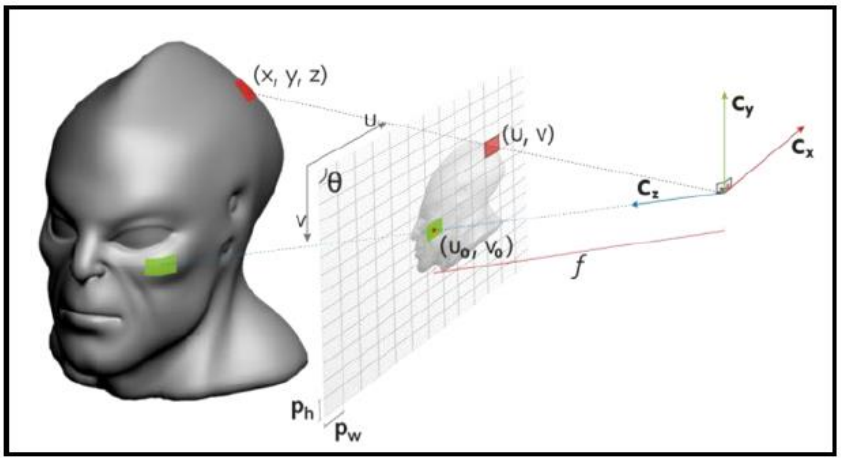


Figura 18 – Origem do centro de coordenadas no plano da imagem – Canto superior esquerdo (Lima, Simões et al. 2010).

2.2.2. DISTORÇÃO

Na realidade, as lentes existentes nas câmaras podem sofrer de distorção, sendo a modelização matemática das lentes uma tarefa tão complexa quanto mais realista pretendermos que o nosso modelo seja. As lentes permitem focar uma grande quantidade de luz, mas pode levar à existência de algumas partes da imagem que e.g. não estejam como o mesmo factor de *zoom* (ampliação). Estas diferenças levam a que o modelo *pinhole* adaptado não seja válido, logo é preciso considerar as distorções existentes. Os tipos de

distorção que mais influenciam a imagem criada (e que normalmente são considerados) são: a distorção radial e a distorção tangencial (Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Lepetit and Fua 2005, Bradski and Kaehler 2008, Szeliski 2010, Cyganek and Siebert 2011, Prince 2012, Radke 2012).

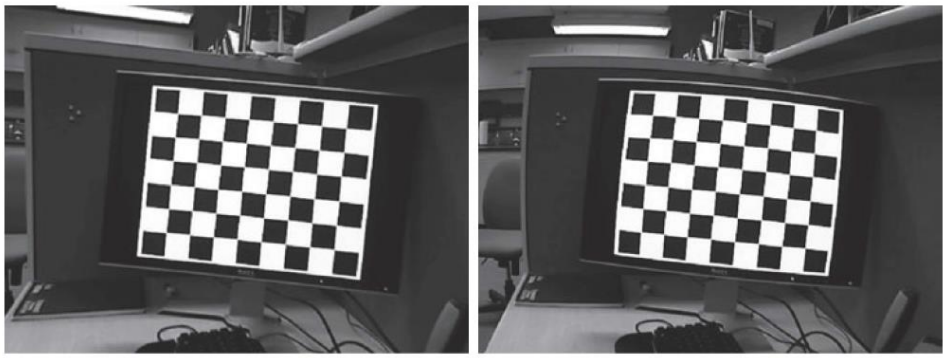


Figura 19 – Exemplo sem distorção Radial (esquerda) e com distorção radial (direita) (Radke 2012).

A distorção radial (Figura 19) representa uma distorção não linear (deformação 2D na imagem), que depende da distância ao centro da imagem. Em termos práticos, esta verifica-se quando o campo de visão da lente é amplo. Esta é facilmente detetada na imagem, conforme Figura 19 e Figura 20, pois as linhas retas no Mundo real apresentam deformações na imagem obtida. Esta distorção é normalmente modelizada como uma função polinomial, de distância r ao centro da imagem, sendo representada da seguinte forma (Lepetit and Fua 2005, Prince 2012, Radke 2012):

$$x_{radial} = x(1 + K_1 r^2 + K_2 r^4 + \cdots) \tag{2.22}$$

$$y_{radial} = y(1 + K_1 r^2 + K_2 r^4 + \cdots) \tag{2.23}$$

com K_1, K_2 a serem dois fatores que variam entre -1 e 1 que controlam o grau de distorção existente e $r^2 = x_c^2 + y_c^2$. Na maior parte dos casos dois fatores (K_1 e K_2) são suficientes para uma boa aproximação da distorção existente, caso seja necessário uma melhor aproximação basta aumentar o grau dos polinómios (2.22) e (2.23).

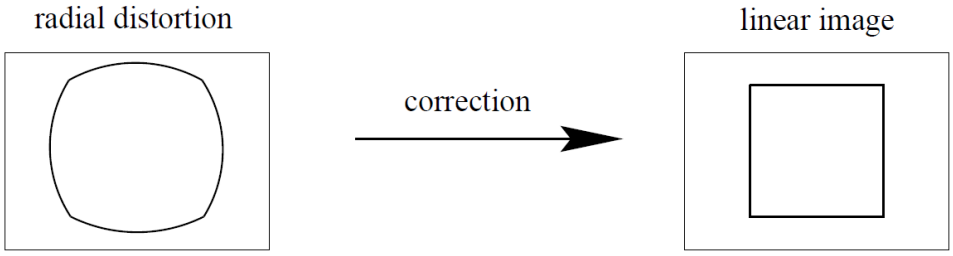


Figura 20 – Quadrado com distorção radial (esquerda) e corrigido (direita) (Hartley and Zisserman 2003).

A distorção tangencial (Figura 21) tem menos influência no resultado final, dependendo basicamente da colocação da lente (a lente não é colocada exatamente paralela ao plano da imagem). Esta pode ser expressa da seguinte forma (Lepetit and Fua 2005, Prince 2012, Radke 2012):

$$x_{tangencial} = x + [2p_1 + p_2(r^2 + 2x^2)] \quad (2.24)$$

$$y_{tangencial} = y + [p_1(r^2 + 2y^2) + 2p_2x] \quad (2.25)$$

com os valores de p_1 e p_2 a variar entre -0,1 e 0,1.Resumindo a distorção radial tem origem na forma da lente e a distorção tangencial tem origem na montagem da lente. A distorção (radial e tangencial) pode ser resumida nas seguintes expressões:

$$x_{Distorção} = x(1 + K_1r^2 + K_2r^4 + \cdots) + 2p_1xy + p_2(r^2 + 2x^2) \quad (2.26)$$

$$y_{Distorção} = y(1 + K_1r^2 + K_2r^4 + \cdots) + 2p_2xy + p_1(r^2 + 2y^2) \quad (2.27)$$

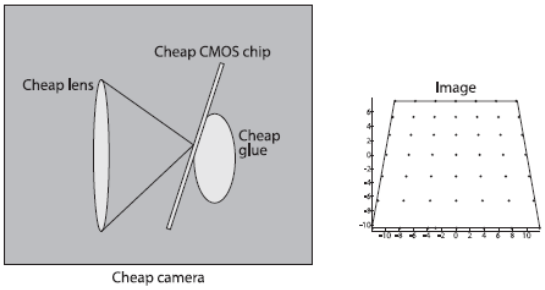


Figura 21 – Distorção tangencial (Bradski and Kaehler 2013).

Recorrendo ao *Software* MatLab e à *toolbox – Lens Distortion Explorer –* é possível verificar a influência dos parâmetros da distorção radial e tangencial (fazendo-os variar de forma independente) numa imagem de teste. A imagem de teste utilizada é a seguinte:

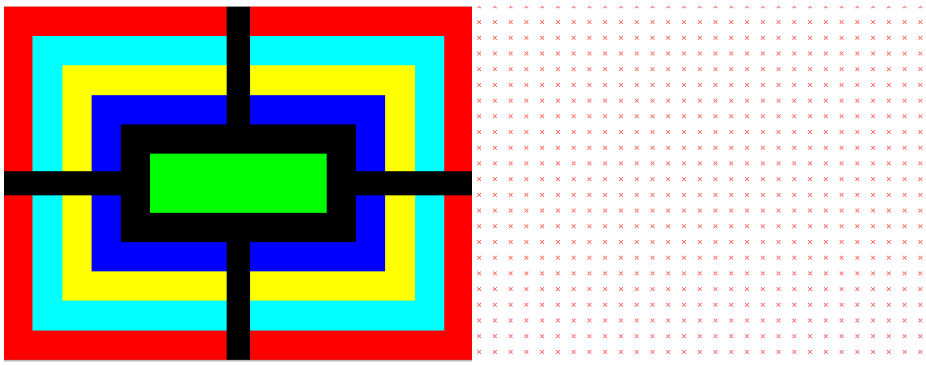


Figura 22 – Imagem de Teste (esquerda) e vetores de distorção (direita).

Ao fazer variar os valores de K_1 e K_2, mantendo p_1 e p_2 nulos, obtemos os seguintes resultados:

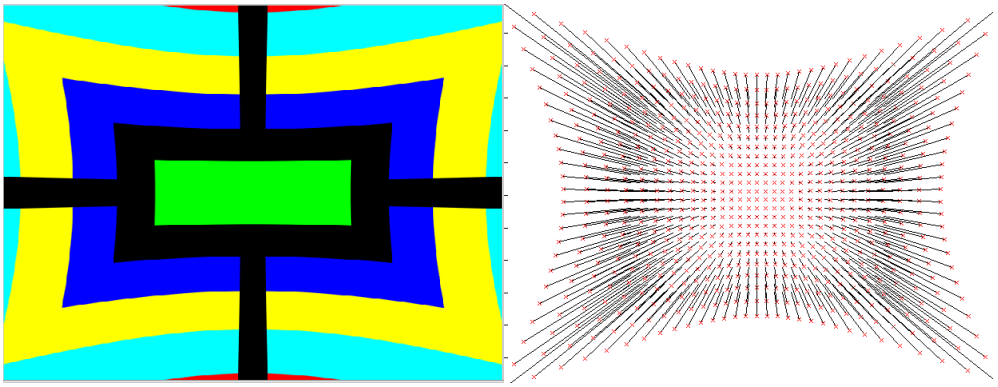


Figura 23 – Imagem de Teste (esquerda) e vetores de distorção (direita) - $K_1 = 0{,}5$ e $K_2 = 0{,}5$.

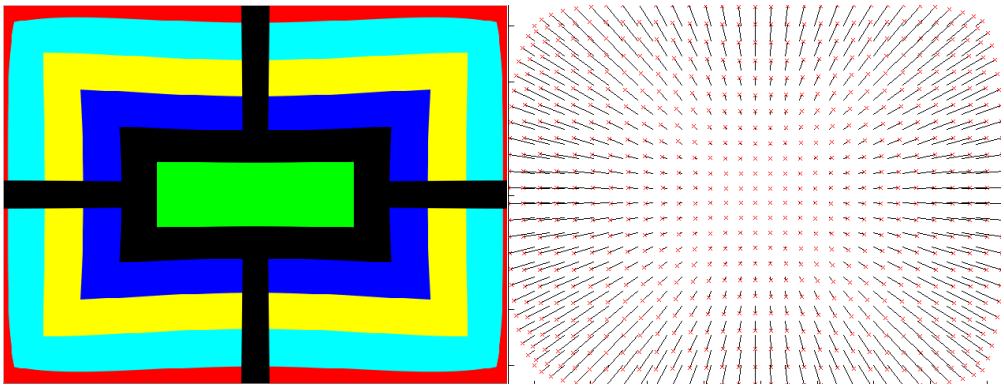


Figura 24 – Imagem de Teste (esquerda) e vetores de distorção (direita) - $K_1 = 0{,}5$ e $K_2 = -0{,}5$.

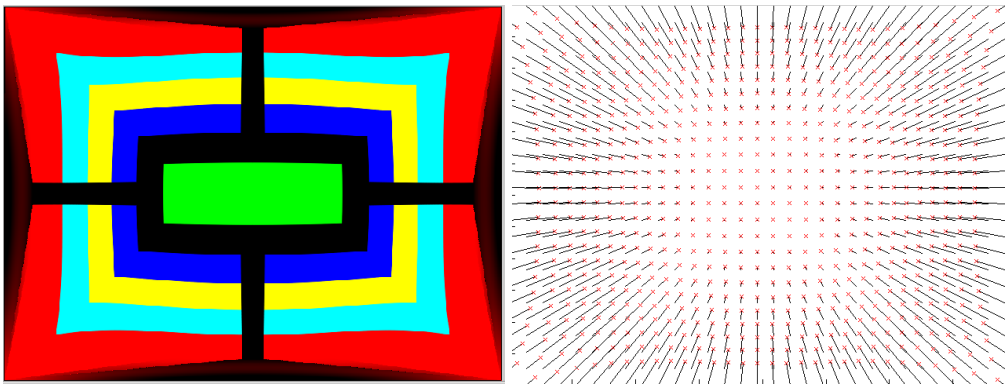


Figura 25 – Imagem de Teste (esquerda) e vetores de distorção (direita) - $K_1 = -0{,}5$ e $K_2 = 0{,}5$.

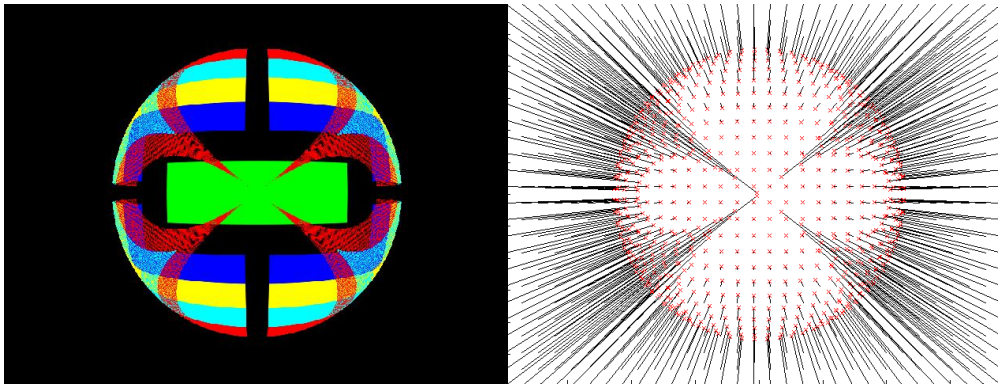


Figura 26 – Imagem de Teste (esquerda) e vetores de distorção (direita) - $K_1 = -0{,}5$ e $K_2 = -0{,}5$.

Se soubermos estes parâmetros a *priori* (possível obter através de calibração – subcapítulo 2.2.3.) podemos efetuar a correção desta distorção (remapeamento dos *pixels* constituintes da imagem). De acordo com a distância ao centro e os parâmetros escolhidos esta distorção vai ter maior influência no resultado final. Fazendo variar os valores de p_1 e p_2, e mantendo K_1 e K_2 nulos obtemos os seguintes resultados:

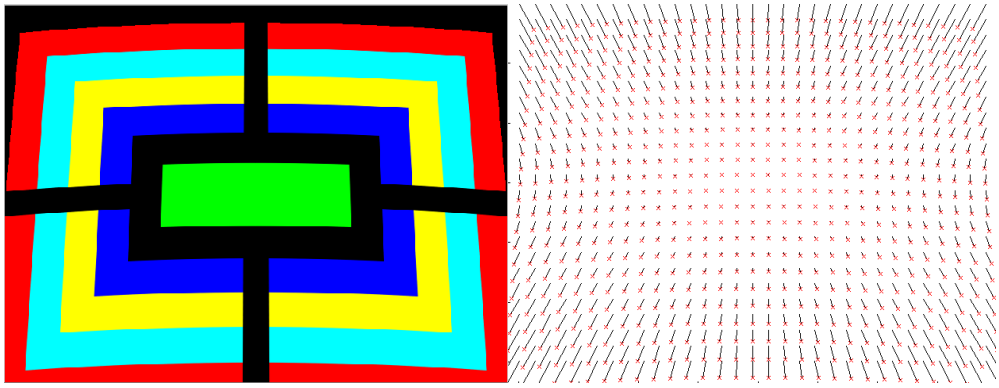


Figura 27 – Imagem de Teste (esquerda) e vetores de distorção (direita) – $p_1 = 0{,}05$ e $p_2 = 0$.

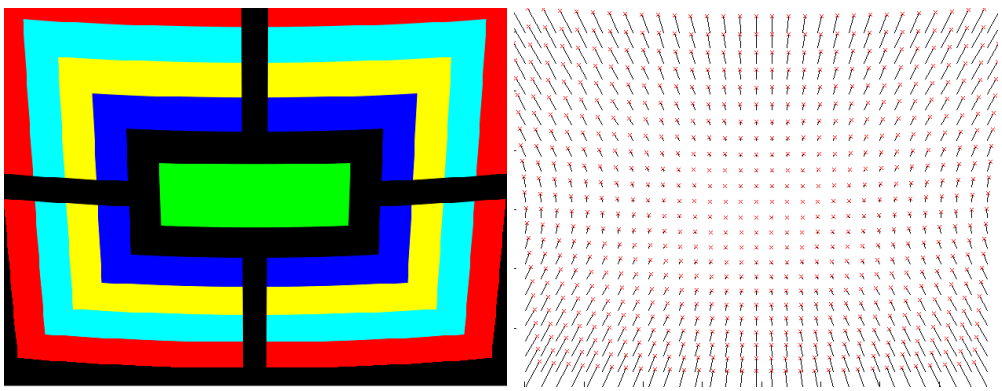


Figura 28 – Imagem de Teste (esquerda) e vetores de distorção (direita) – $p_1 = -0{,}05$ e $p_2 = 0$.

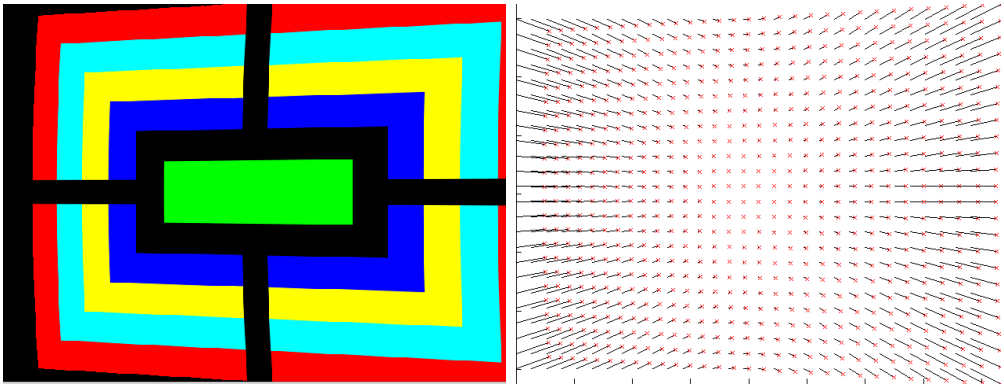


Figura 29 – Imagem de Teste (esquerda) e vetores de distorção (direita) – $p_1 = 0$ e $p_2 = 0{,}05$.

No caso da distorção tangencial a sua influência é mais facilmente percetível, através da análise das figuras anteriores em que se percebe claramente que num caso real o sensor não teria sido colocado de forma paralela ao plano da imagem.

2.2.3. CALIBRAÇÃO

Na maior parte dos métodos de *tracking* 3D é assumido que os parâmetros intrínsecos da câmara são fixos e conhecidos, não existindo variação de *zoom* durante a normal operação do sistema. Isto acontece, pois é difícil distinguir uma variação na distância focal de uma translação em Z. Em caso ser necessário fazer uma variação de *zoom* esta variação deve ser feita dentro de valores conhecidos e previamente estabelecidos, possibilitando assim a obtenção dos parâmetros intrínsecos *a priori*. Os parâmetros da câmara podem ser estimados *online* durante o normal funcionamento do sistema ou *offline* num processo conhecido como calibração. O processo de calibração não vai ser aqui exposto em grande detalhe pois não apresenta grande relevância para a compreensão do sistema desenvolvido.

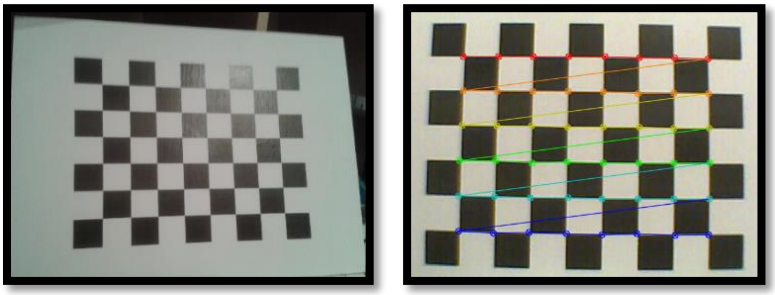


Figura 30 – Padrão de calibração utilizado – Imagem real.

O método de calibração utilizado recorre a um padrão em xadrez (Figura 30) dentro do campo de visão, sendo as suas dimensões rigorosamente conhecidas. Este padrão é impresso e colocado sobre um plano, sendo captadas várias *frames* do padrão em várias posições (Sturm and Maybank 1999, Zhang 2000, Lepetit and Fua 2005, Bradski and Kaehler 2008, Bradski and Kaehler 2013). Temos sempre a opção de utilizar os parâmetros descritos pelo fabricante (quando disponíveis), mas para tarefas em que é necessária uma grande precisão é sempre aconselhável caracterizar o equipamento de aquisição de imagem que se está a utilizar. Ao utilizar um plano podemos assumir que a coordenada Z é nula com os eixos X e Y alinhados com o padrão. Na prática 10 a 20 imagens são suficientes, não trazendo melhoria significativa nos resultados a utilização de mais imagens.

Utilizando a biblioteca *open source*[22] OpenCV é possível obter os parâmetros de distorção e da câmara (Figura 31), podendo efetuar a respetiva correção caso seja necessário. Foram obtidos os seguintes parâmetros de calibração:

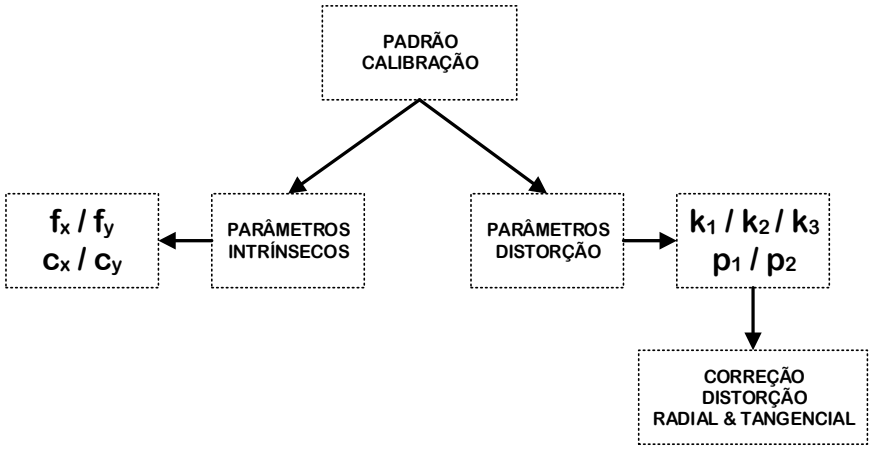


Figura 31 – Esquema Geral – Parâmetros obtidos através da calibração – utilizando biblioteca OpenCV.

Um exemplo de um caso concreto utilizado experimentalmente foi a câmara de 5 *megapixels* de um telemóvel - *Samsung Galaxy SIII mini* – os parâmetros de distorção obtidos após calibração foram:

K_1	K_2	K_3	p_1	p_2
0,1880	-0.2064	0.0000	-0.0044	0.0009

Tabela 2 – Parâmetros de distorção obtidos – Câmara telemóvel.

[22] O conceito de *open source* está associado a uma filosofia de partilha global em que todo o conteúdo pode ser visualizado e alterado sem qualquer restrição, não devendo no entanto esquecer de referenciar os direitos intelectuais de terceiros caso aplicável.

A obtenção dos parâmetros de distorção permite a correção da imagem obtida (Figura 33), obtendo assim uma imagem sem distorção (Figura 34).

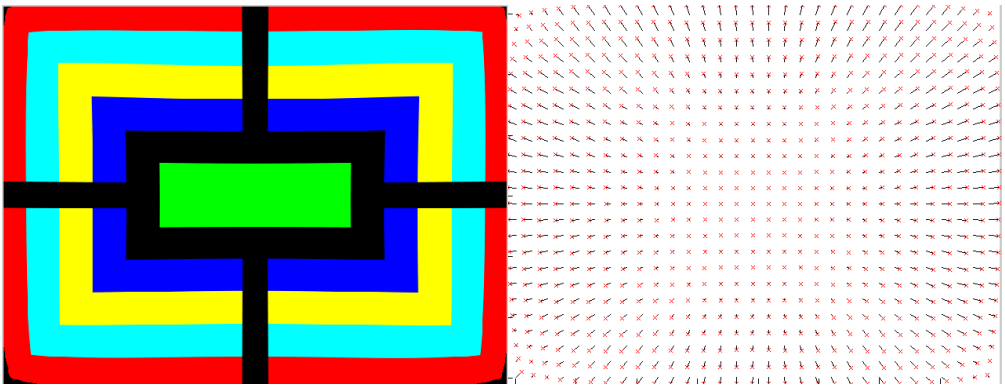


Figura 32 – Imagem de Teste (esquerda) e vetores de distorção (direita) – Câmara telemóvel.

Figura 33 – Exemplo de imagem real.

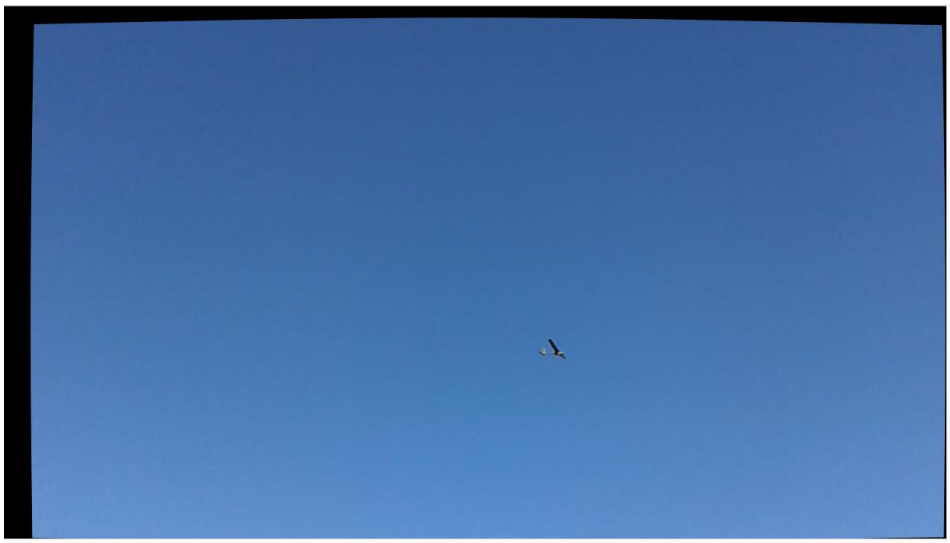


Figura 34 – Exemplo de imagem real com correção da distorção.

2.2.4. PARÂMETROS EXTRÍNSECOS

Os parâmetros extrínsecos (que dependem do sistema de coordenadas escolhido) relacionam as coordenadas da câmara (plano de imagem – 2D) com as coordenadas do Mundo (3D), descrevendo assim a posição e orientação da câmara no Mundo 3D. Considerando que a câmara está na origem do sistema de coordenadas ortonormado[23], a posição e orientação (pose) da câmara no sistema de coordenadas do Mundo pode ser escrito da seguinte forma (Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Szeliski 2010, Cyganek and Siebert 2011, Forsyth and Ponce 2012):

$$\lambda p = [K \mid 0_3] \begin{bmatrix} R & -R\tilde{C} \\ 0_e^T & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = KR\left[I \mid -\tilde{C}\right]P \tag{2.28}$$

com K a definir os parâmetros intrínsecos (como descrito anteriormente no subcapítulo 2.2.1), R representa a matriz de rotação (3x3) e $\tilde{C}$ define o centro de projeção da câmara no Mundo em coordenadas não homogéneas, I é a matriz identidade (3x3), q representa o ponto em 2D ($p = (x, y, 1)^T$) e P o mesmo ponto em 3D ($P = (X, Y, Z, 1)^T$).

A projeção de 2D para 3D pode ser efetuada seguindo o processo inverso ao descrito na equação (2.28), sendo normalmente designado por *back-projection*. Projetando assim um ponto 2D (plano da imagem) para um conjunto de pontos 3D no espaço, sendo todos os pontos possíveis (conjunto de pontos) os situados sobre uma reta que passa pelo ponto de projeção da câmara. Para um raio $Q(\lambda)$ que passa pelo centro da câmara $\tilde{C} = (\tilde{C}_x, \tilde{C}_y, \tilde{C}_z)^T$ e pelo ponto $p = (x, y, 1)^T$ no plano da imagem é definido como:

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \tilde{C} + \lambda R^{-1} K^{-1} p \tag{2.29}$$

sendo λ um factor de escala que permite definir a posição do ponto em 3D sobre a reta (ou raio como é comummente designado). Se a coordenada Z for conhecida *a priori*, as coordenadas X e Y são obtidas por:

$$\lambda = \frac{Z - \tilde{C}_z}{z_3} \tag{2.30}$$

com $(z_1, z_2, z_3)^T = R^{-1}K^{-1}p$. Os parâmetros extrínsecos descrevem assim os parâmetros geométricos necessários para mudar do referencial da câmara para o referencial do Mundo

[23] Eixos perpendiculares entre si com unidade de medida igual.

e vice-versa. Sendo estes parâmetros simplesmente duas matrizes: uma de rotação – R – e uma de translação – T.

2.3. VISÃO ESTEREOSCÓPICA

A forma mais fácil de fazer uma analogia à visão estereoscópica é analisarmos o que acontece na natureza. A maior parte dos animais, e em particular os seres humanos, possuem dois olhos tendo estes uma perspetiva diferente do Mundo que observam pois estão localizados em posições diferentes. O cérebro faz o processamento das imagens geradas pelos nossos sensores (olhos), conseguindo assim obter uma noção de profundidade. Esta abordagem é em tudo semelhante ao que se pode fazer utilizando duas câmaras, localizadas a uma determinada distância entre si, sendo o seu princípio de funcionamento abordado ao longo deste subcapítulo. O seu funcionamento vai ser explicado de forma muito resumida pois esta abordagem apesar de explorada numa fase inicial (Figura 35) acabou por não ser utilizada no sistema final.

Consideremos duas câmaras que capturam a mesma cena (Figura 35), sendo que cada imagem representa a mesma cena estática mas em duas perspetivas diferentes. A relação entre as duas imagens é dada pela denominada geometria epipolar.

Figura 35 – Visão estereoscópica – Aparato experimental utilizado.

O primeiro trabalho (de que se tem registo) que está diretamente relacionado com a geometria de visão múltipla – *multiple-view* – foi elaborado em 1913 num documento do matemático Alemão *Erwin Kruppa* (Ma, Soatto et al. 2004).

Dados dois pontos x e x' (Figura 36 - lado esquerdo) é possível verificar que a relação entre estes pontos é dada pelo plano epipolar π (C, C', X, x e x são coplanares). O ponto x no plano da imagem (Figura 36 - lado direito) pode ser projetado no espaço numa reta formada por x e o centro de projeção C. Esta reta é representada na segunda câmara por uma linha - l' - sendo que um ponto projetado em 3D para x deve estar sobre esta reta e sobre a linha l'.

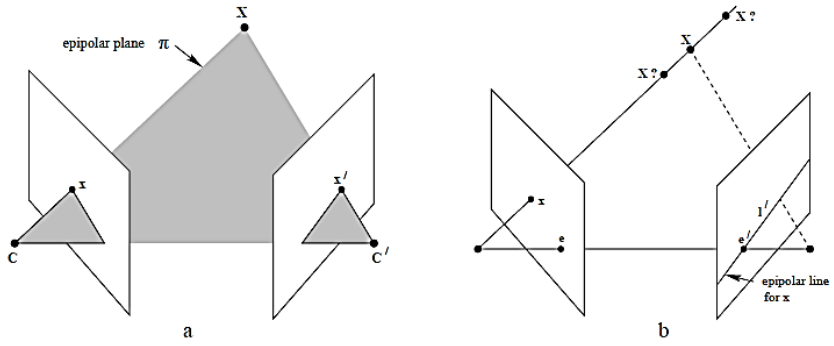


Figura 36 – Correspondência entre pontos - Geometria Epipolar (Hartley and Zisserman 2003).

É possível chegar à conclusão que um ponto x referente à primeira câmara apenas precisa de ser procurado sobre a linha l' na segunda câmara, o que restringe a área de busca. Esta restrição é conhecida neste domínio por restrição epipolar (Hartley and Zisserman 2003, Cyganek and Siebert 2011, Fahmy, Ismail et al. 2013).

A linha que passa pelos dois centros de projeção é conhecida como – linha de base – e a intersecção da linha de base com o plano da imagem é denominado por – epipolo. O plano epipolar é o plano definido pelos centros de projeção C, C' e pelo ponto 3D representado por X (Figura 36). A linha epipolar é a linha que intersecta o plano da imagem com o plano epipolar (Figura 37).

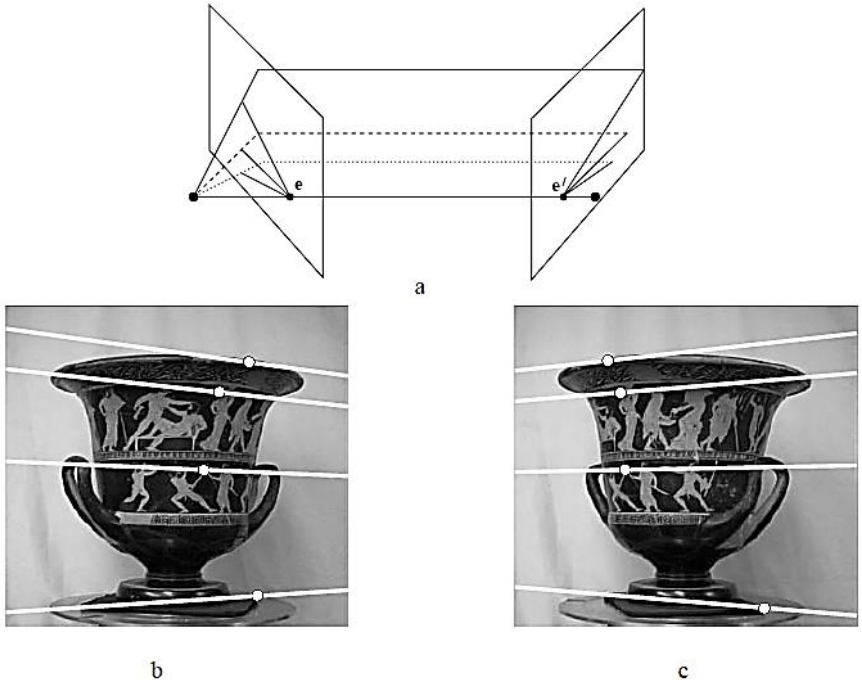


Figura 37 – Geometria epipolar (a), Par de imagens retiradas de um sistema de estereoscopia (b) e (c) (Hartley and Zisserman 2003).

2.3.1. MATRIZ FUNDAMENTAL

A matriz fundamental (F) é a representação algébrica da geometria epipolar, representando o mapeamento entre os pontos de uma imagem para as linhas epipolares de outra imagem (Hartley and Zisserman 2003). Como se pode observar na Figura 36 cada ponto da primeira imagem (x) possui uma linha epipolar (l') na segunda imagem e um ponto (x'). Existe assim um mapeamento de x para l', ou seja, de um ponto de uma imagem para a sua linha epipolar correspondente em outra imagem. A matriz fundamental tem as seguintes propriedades e características (Lamoureux , Hartley 1997, Hartley and Zisserman 2003, Ma, Soatto et al. 2004, Wu, Hu et al. 2005, Sur, Noury et al. 2008):

- **Transposição**: Se o par de câmaras (Ca_1, Ca_2) tem como F a sua matriz fundamental, então F^T representa a matriz fundamental para as câmaras (Ca_2, Ca_1);
- **Linhas Epipolares:** Um ponto x na primeira imagem pode ser correspondido à linha epipolar da segunda imagem definida por $l' = Fx$. Com $l = F^T x'$ a corresponder à relação entre um ponto na segunda imagem com a linha epipolar na primeira imagem;
- **Correspondência entre pontos:** Se x e x' são pontos correspondentes, temos que ${x'}^T Fx = 0$ para todos os pontos correspondentes ($l' = Fx$);
- **Epipolo:** Para um ponto arbitrário x da primeira imagem (à exceção de e - Figura 37), o epipolo e' na imagem 2 é um ponto na linha epipolar $l' = Fx$. Os epipolos e e e' são definidos nas coordenadas da câmara 1 e 2 respetivamente, de modo a que ${e'}^T(Fx) = \left({e'}^T F\right)x = 0$ para qualquer ponto x na linha epipolar, que implica ${e'}^T F = 0$.

Se considerarmos o par de pontos correspondente pp e pp' nas duas imagens adquiridas, podemos definir a matriz fundamental da seguinte forma (Lamoureux , Hartley 1997, Wu, Hu et al. 2005, Sur, Noury et al. 2008):

$$pp'Fpp = [x'y'1]\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \qquad (2.31)$$

Resultando em:

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0 \quad (2.32)$$

em que f pode ser considerado como um vetor de 9 termos representado por $f = (f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33})^T$, podendo reescrever (2.32) da seguinte forma:

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)f = 0 \quad (2.33)$$

Considerando n pares de pontos, o sistema linear de equações pode ser escrito da seguinte forma (Lamoureux , Hartley 1997, Wu, Hu et al. 2005, Sur, Noury et al. 2008):

$$Af = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ x'_2x_2 & x'_2y_2 & x'_2 & y'_2x_2 & y'_2y_2 & y'_2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} f = 0 \quad (2.34)$$

A correspondência de pontos entre imagens num caso real não é exata, devendo ser calculada assim uma solução aproximada. Esta solução pode ser calculada através da decomposição entre valores singulares[24] (SVD), em que f corresponde ao vetor próprio da matriz A^TA relativo ao menor valor próprio de A^TA.

As linhas epipolares contêm os epipolos, mas no cálculo da matriz fundamental esta nem sempre é singular fazendo com que a linha epipolar não seja coincidente. Isto é facilmente resolvido acrescentando uma restrição de singularidade em que F é substituída por F', F' tem um determinante nulo e minimiza $\|F - F'\|$. F pode ser decomposto em $F = UDV^T$, em que $D = diag(\sigma_1, \sigma_2, \sigma_3)$ e que $\sigma_1 \geq \sigma_2 \geq \sigma_3$. F' é dado por $F' = U\, diag(\sigma_1, \sigma_2, 0)V^T$ que minimiza $\|F - F'\|$.

Um dos métodos utilizado para o cálculo da matriz fundamental é o algoritmo dos 8 pontos (em que é necessário garantir que $n \geq 8$ por forma a obter uma solução única), sendo estas coordenadas normalizadas e com a restrição de singularidade referida anteriormente. Para um conjunto de pontos correspondentes p_i e p'_i, o algoritmo de 8 pontos é dado por (Lamoureux , Hartley 1997, Wu, Hu et al. 2005, Sur, Noury et al. 2008):

Algoritmo 8 pontos: Descrição geral simplificada
1. Normalizar coordenadas dos pontos p_i e p'_i, em que se obtém $\hat{p}_i = Tp_i$ e $\hat{p}' = T'p'_i$. T e T' são transformações de escala e translação – normalização – para o ponto p_i e p'_i respetivamente
2. Calcular a matriz fundamental $\hat{F}$ (pontos $\hat{p}_i$ e $\hat{p}'_i$), utilizando a equação (2.33)
3. Calcular a matriz $\hat{F}'$ tendo em conta $\hat{F}$, com $\hat{F}'$ a respeitar a restrição de singularidade. $\hat{F}'$ minimiza $\left\|\hat{F} - \hat{F}'\right\|$, e tem a restrição que $\det(\hat{F}') = 0$
4. Aplicar $F = T'^T\hat{F}'T$, para obter respetiva matriz fundamental

[24] É a factorização de uma matriz real ou imaginária. Este método é normalmente utilizado para resolver sistemas de equações em que o número de equações é superior ao número de incógnitas.

As matrizes T e T' são descritas da seguinte forma:

$$T = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix} \text{ e } T' = \begin{bmatrix} s' & 0 & t'_x \\ 0 & s' & t'_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.35}$$

os parâmetros s e $t = \left(t_x, t_y\right)^T$ definem o fator de escala e a translação respetivamente para a matriz T, e s' e $t' = \left(t'_x, t'_y\right)^T$ para a matriz T'. A matriz essencial pode ser escrita da seguinte forma:

$$E = K'^T F K \tag{2.36}$$

em que K e K' representam as matrizes de parâmetros intrínsecos da câmara P e P' respetivamente, sendo a constituição destas matrizes descrita anteriormente no subcapítulo 2.2.1.

O resultado final pretendido num sistema de visão estereoscópica é um mapa de profundidade, permitindo assim a extração de informação a partir de uma cena 3D. Este mapa de profundidade estima a profundidade para cada ponto de uma imagem em 2D, representando assim a cena em 3D. Esta profundidade é obtida pela correspondência entre pontos de um par de imagens, sendo a busca por um ponto p realizada na segunda imagem apenas na sua correspondente linha epipolar. Esta busca é melhorada se forem utilizadas imagens em que a distorção foi devidamente corrigida (Hartley 1997). Surge então uma medida designada por – disparidade – que quantifica a diferença entre os pontos correspondentes da imagem da esquerda e direita e pode ser usada para obter a profundidade de acordo com a seguinte relação (utilizando câmaras paralelas - Figura 38):

$$d = B \times f \times \frac{1}{z} \tag{2.37}$$

com B (linha de base – *baseline*) a corresponder à distância entre os centros de projeção das duas câmaras e f é a distância focal. As principais limitações no cálculo dos mapas de disparidade centram-se no seguinte:

- O ruído existente faz com que a correspondência entre pontos possa ser incorreta;
- A existência de oclusão faz com que não seja possível determinar a correspondência entre pontos;
- Existência de descontinuidades de profundidade que estão normalmente associadas aos limites de um objeto, sendo que se for calculada a medida de correlação de blocos próximos existem blocos com partes de objeto com duas profundidades diferentes fazendo com que a medição seja incorreta;

- A não existência de textura em certas regiões, fazendo com que a similaridade calculada possa ser a mesma.

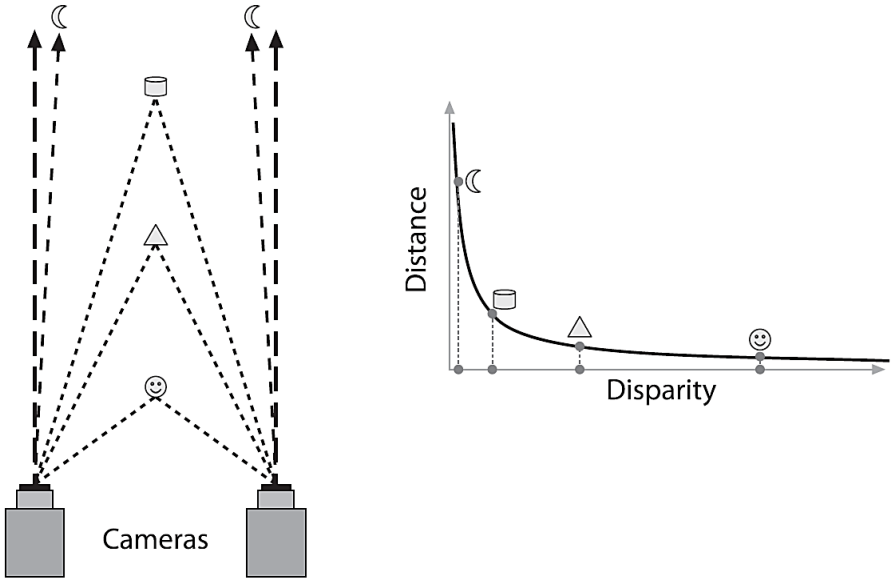


Figura 38 – Relação entre disparidade e profundidade (Bradski and Kaehler 2008).

Figura 39 – Exemplo de imagem esquerda, imagem direita e mapa de disparidade (Georgoulas and Andreadis 2011).

O mapa de profundidade (Figura 39) pode ser obtido através do seguinte algoritmo (Okutomi and Kanade 1993, Morvan 2009):

$$d(x,y) = arg\min \sum_{(i,j)\,\epsilon\, W} \left| I_1(x+i, y+j) - I_2(x+i-\check{d}, y+j) \right| \qquad (2.38)$$

em que $\check{d}$ é a disparidade candidata para um determinado par de blocos, i e j as coordenadas dos *pixels* pertencentes aos blocos,d é a disparidade calculada (profundidade obtida pela equação descrita em (2.37)) e W corresponde a uma janela (um bloco de correspondência) em volta dos *pixels* (Janela delimitada a branco na Figura 40).

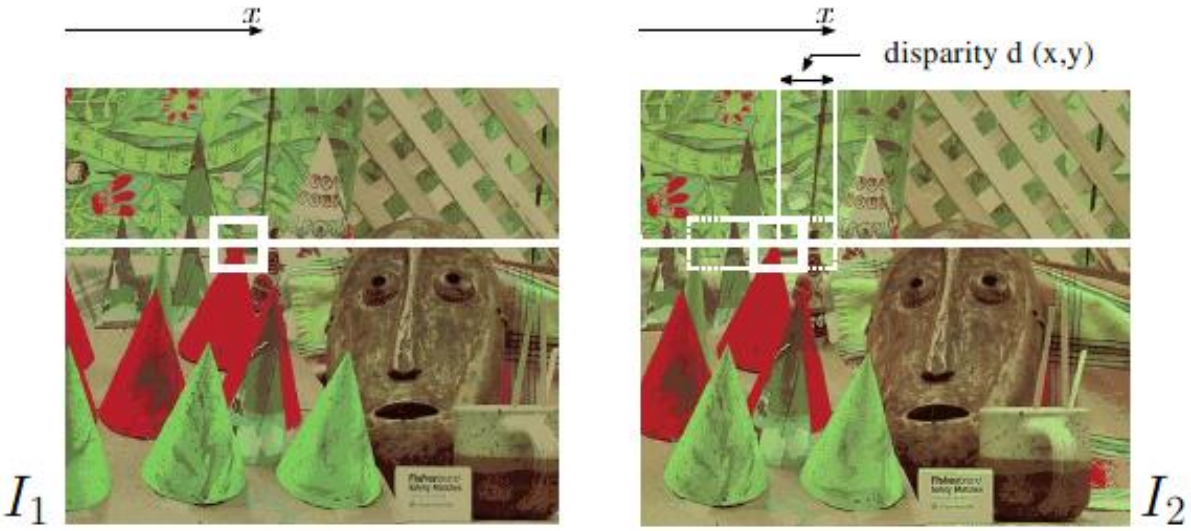


Figura 40 – A disparidade é estimada ao pesquisar o bloco mais semelhante na segunda imagem (I_2) ao longo de uma busca 1D horizontal na linha epipolar (Morvan 2009).

O *pixel* p_1 da imagem de referência I_1, corresponde a uma busca horizontal na imagem I_2 pelo *pixel* p_2. A correlação é medida pela soma das diferenças absolutas e a disparidade d de um *pixel* na imagem I_1 conforme exposto na equação (2.38). Mas existem outros algoritmos com o mesmo objetivo final, que é o cálculo do mapa de disparidade e consequente profundidade – coordenada Z.

Existem muitos mais métodos e conceitos a abordar nesta temática, mas esta arquitetura acabou por não ser utilizada no sistema final. Esta arquitetura acarreta a utilização de pelo menos uma câmara extra do que a visão monocular, e na sua arquitetura mais simples permite apenas obter a noção de profundidade e não a atitude do objeto como necessário para a malha de controlo do UAV.

2.4. CLASSIFICADOR

Um classificador (Figura 41) é uma ferramenta (um "interpretador") que permite (recorrendo a um algoritmo) através de um conjunto de dados de treino gerar conhecimento (através de aprendizagem[25]) e prever se um novo dado pertence ou não a essa classe, através da análise das suas características. O objetivo de um classificador não é aprender a agir no

[25] A aprendizagem neste contexto refere-se a um processo de otimização – minimização do erro.

conjunto específico de treino, mas sim no universo de dados. Os dados de treino podem ser divididos em (Marakas 2003, Cortes 2005, Turban, Sharda et al. 2007):

- **Conjunto de treino**: Usado para construir (criar) o classificador (quanto maior, melhor o classificador obtido);
- **Conjunto de validação**: Usado para controlar o processo de aprendizagem (este conjunto é opcional);
- **Conjunto de teste**: Usado para estimar o desempenho (quanto maior, melhor a estimativa do desempenho apresentado pelo classificador).

Figura 41 – Classificador – Esquema geral de funcionamento.

O classificador vai ser utilizado neste contexto para fazer a segmentação de imagem para posterior processamento, fazendo com que a área analisada – região de interesse – corresponda à região em que o objeto a detetar se encontra. Interessa pois abordar os classificadores, neste contexto específico de estudo, que o permitam fazer com sucesso.

Como nos encontramos num ambiente em que a intensidade luminosa não é controlada, e o fundo da própria imagem não é constante é praticamente impossível de aplicar e.g. segmentação por comparação de histogramas ou outros métodos que seriam facilmente aplicáveis se o ambiente fosse estático e de parâmetros controlados e controláveis.

O classificador utilizado é um classificador em cascata (*cascade*) - Figura 42 - baseado em estágios (*stages*), sendo cada estágio constituído por *weak learners*. Estes são basicamente classificadores lineares constituídos por uma árvore de decisão com apenas um nó[26] (Quinlan 1986, Viola and Jones 2001, Lienhart, Kuranov et al. 2003, Sammut and Webb 2011, Flach 2012, Dua and Du 2014). Um classificador deste tipo $h_j(x)$ é basicamente constituído por uma *feature*[27] (característica) - f_j, um valor de limiar (*threshold*) - θ_j e uma paridade p_j indicando a direção do sinal de desigualdade (Viola and Jones 2001):

$$h_j(x) = \begin{cases} 1 & se\ p_j f_j(x) < p_j \theta_j \\ 0 & de\ outra\ forma \end{cases} \tag{2.39}$$

com x a representar uma subjanela da imagem de dimensão $largura \times altura$ *pixels* (Viola e Jones no seu trabalho intitulado *Rapid object detection using a Boosted Cascade of Simple Features* utilizam uma subjanela de 24 x 24 pixels com bons resultados – a mesma dimensão que é utilizada neste trabalho) e $f_j(x)$ a ser o valor da característica na subjanela da imagem (quanto mais próximo da característica em análise maior o valor). Para cada característica é determinado o valor de limiar ótimo - θ_j -, de modo a o menor número de exemplos seja classificado de forma errada. Cada estágio é treinado utilizando uma técnica chamada *boost,* que permite obter um classificador preciso utilizando uma média ponderada dos resultados obtidos utilizando um conjunto de classificadores lineares (Schapire 1990, Freund and Schapire 1995, Freund and Schapire 1996, Viola and Jones 2001, Lienhart, Kuranov et al. 2003).

Cada estágio do classificador define se é positivo ou negativo a região em que a janela de pesquisa se encontra (Figura 42). Se o resultado obtido é negativo não existe necessidade de mais testes e a região não é mais pesquisada, em caso de positivo a área correspondente passa para o estágio seguinte. Se no estágio final a região for dada como positiva, o classificador reporta como tendo sido encontrado um membro da classe - objeto (Quinlan 1986, Viola and Jones 2001, Lienhart, Kuranov et al. 2003, Sammut and Webb 2011, Flach 2012, Dua and Du 2014). Os estágios também são ordenados por processamento necessário (tempo necessário para processamento das características, muito ligado também ao tamanho da janela em que a característica se encontra), fazendo com que no primeiro estágio apresente uma menor complexidade computacional do que o

[26] Também designado na literatura por *decision stumps*.
[27] O termo *feature* e característica vão ser utilizados ao longo desta tese indiscriminadamente, e servem para designar o mesmo.

segundo e por aí em diante (Viola and Jones 2001, Lienhart, Kuranov et al. 2003). Como em todos os classificadores podem ocorrer falsos positivos quando uma amostra negativa é classificada como positiva e falsos negativos quando uma amostra positiva é classificada como negativa.

Tendo em conta estes aspetos é importante apresentar uma taxa muito baixa de falsos negativos em cada estágio (quando é classificado como negativo o classificador não prossegue - Figura 42), mas poderá existir (é aceitável) uma taxa elevada de falsos positivos em cada estágio (mesmo que seja incorretamente classificado num estágio, no estágio seguinte volta a ser verificado). Mas é claro que o ideal é ter uma taxa nula de falsos (positivos ou negativos), o que na prática é utópico devendo existir assim um cuidado na escolha do número de estágios e amostras para o treino do classificador.

Deve existir um equilíbrio entre a existência de poucos estágios com uma taxa de falsos positivos baixa por estágio ou termos muitos estágios com uma alta taxa de falsos positivos por estágio. Um estágio com uma taxa reduzida de falsos positivos é constituído por mais classificadores lineares, sendo que os estágios com uma taxa elevada de falsos positivos contêm menos classificadores lineares. Idealmente é preferível ter um grande número de classificadores lineares por estágio, descendo assim a taxa de falsos positivos em cada estágio. O número de estágios deve ser escolhido por forma a obter uma taxa de falsos negativos adequada ao fim a que o classificador se destina.

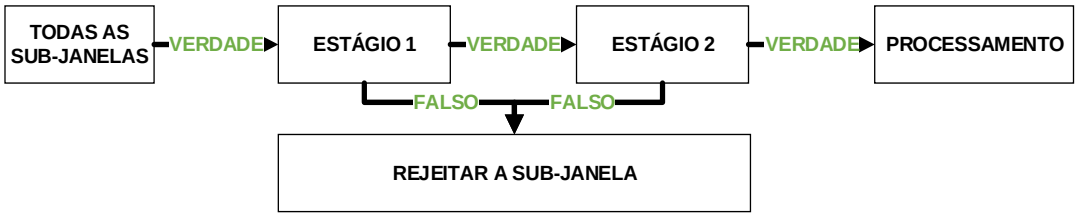


Figura 42 – Exemplo de arquitetura em cascata – Esquema simplificado.

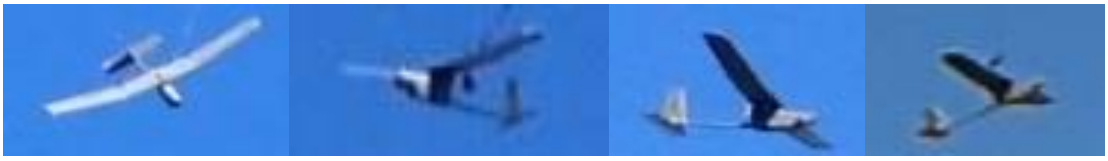


Figura 43 – Exemplo de amostras positivas- treino classificador.

Para treinar um classificador (neste domínio específico de estudo) são necessárias duas amostras distintas:

- Imagens que correspondem ao objeto a detetar – Amostras positivas (Figura 43);
- Imagens que não correspondem, nem contêm o objeto a detetar – Amostras negativas (Figura 44).

Figura 44 – Exemplo de amostras negativas – treino classificador.

A imagem é representada através de uma representação chamada de *integral image* (imagem integral), que permite uma análise muito rápida das características a analisar. Ou seja, a análise da imagem não é feita segundo as intensidades dos *pixels* mas segundo uma representação integral que é calculada utilizando poucas operações por *pixel*. A imagem integral na localização x e y contém a soma dos *pixels* acima e à esquerda de x e y inclusivé, da seguinte forma (Viola and Jones 2001):

$$ii(x,y) = \sum_{x' \leq x, y \leq y} i(x',y') \tag{2.40}$$

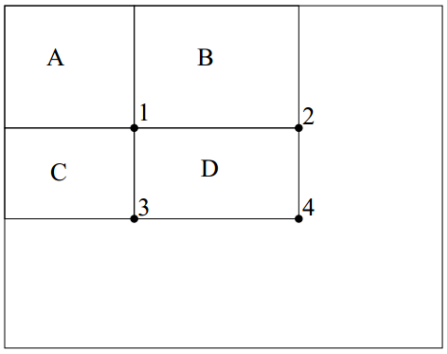


Figura 45 – Imagem integral – Soma dos *pixels* acima e à esquerda de x e y inclusive (Viola and Jones 2001).

O valor da imagem integral na localização 1 (Figura 45) é a soma dos *pixels* no retângulo A, o valor na localização 2 é A+B, na localização 3 é A+C e na localização 4 é A+B+C+D. Podemos então calcular o ponto pretendido da seguinte forma (Viola and Jones 2001):

$$s(x,y) = s(x,y-1) + i(x,y) \tag{2.41}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{2.42}$$

com $s(x,y)$ a corresponder à soma cumulativa das linhas, $s(x,-1) = 0$ e $ii(-1,y) = 0$.

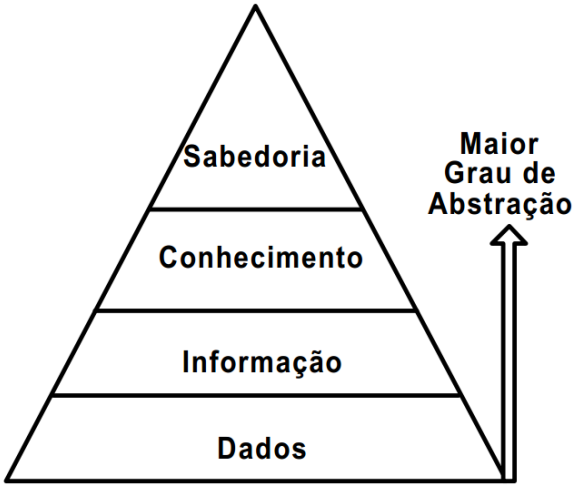


Figura 46 – Hierarquia de compreensão e utilidade (Navega 2002).

A utilização de características em vez de e.g. valores de intensidade de *pixels* como entrada do algoritmo tem como objetivo tornar a classificação mais fácil, pois as características normalmente compilam conhecimento (Figura 46). Este conhecimento é difícil de adquirir ao utilizar dados brutos e um número finito de dados de entrada para treino. A complexidade de avaliação da característica também é muito importante, sendo que é fixada uma janela de tamanho fixo para todas as escalas na imagem de entrada, calculando assim as características em qualquer posição e em qualquer escala ao mesmo tempo (Viola and Jones 2001, Lienhart, Kuranov et al. 2003, Hadid, Pietikainen et al. 2004). Foram exploradas três tipos de características (*features*):

- *Local Binary Pattern* (LBP);
- *Haar-like features*;
- *Histogram of oriented Gradients* (HOG).

É importante que as características utilizadas garantam o seguinte (Viola and Jones 2001, Lienhart, Kuranov et al. 2003, Ahonen, Hadid et al. 2004, Hadid, Pietikainen et al. 2004):

- Boa discriminação das diferentes classes, enquanto tolera variações dentro de cada classe;
- Devem ser facilmente extraídas das imagens, permitindo assim um rápido processamento;
- Tamanho de vetor curto para evitar um classificador que necessite de um processamento computacional elevado.

2.4.1. *LOCAL BINARY PATTERN* - LBP

A ideia base para o desenvolvimento do operador LBP (*local binary pattern*) foi que as texturas de uma superfície bidimensional (2D) podem ser descritas por duas medidas complementares: padrões espaciais locais e contraste em escala de cinzento. A Figura 47 demonstra um exemplo de cálculo de LBP, em que os histogramas obtidos podem ser utilizados para a descrição da textura (Ahonen, Hadid et al. 2004, Hadid, Pietikainen et al. 2004, Xiaowei, Hong et al. 2013). O histograma representa a textura e é constituído e.g. por pontos, cantos, áreas lisas, etc. A Figura 48 apresenta alguns exemplos de padrões, ao utilizar uma vizinhança circular e interpolação bilinear podemos ter valores de *pixels* não inteiros o que permite a utilização de qualquer raio (dentro da imagem) e número de *pixels* de vizinhança.

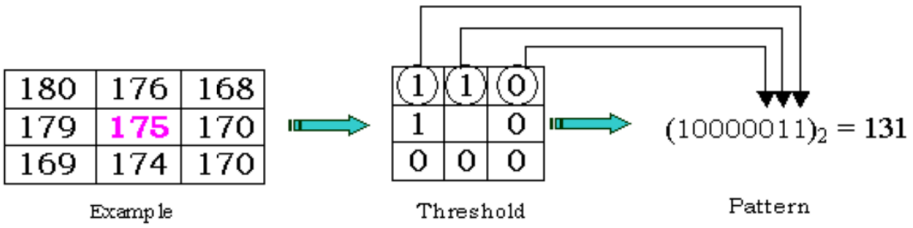


Figura 47 – Exemplo de cálculo LBP (Hadid, Pietikainen et al. 2004).

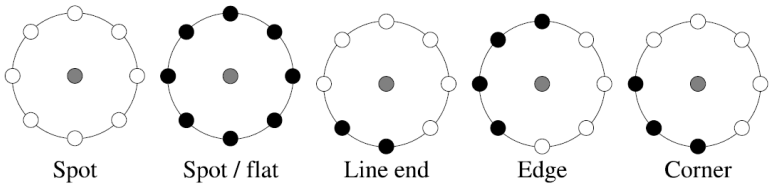


Figura 48 – Exemplo de texturas primitivas – padrões - que se podem detetar usando LBP – pontos brancos representam uns e pontos pretos representam zeros (Pietikäinen, Hadid et al. 2011).

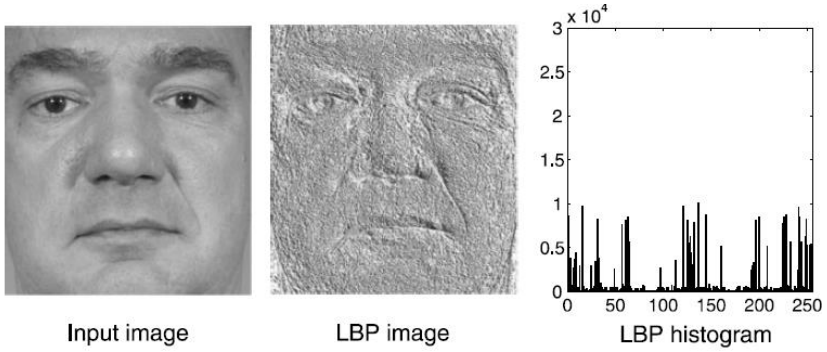


Figura 49 – Exemplo de imagem de entrada, a imagem LBP e respetivo histograma (Pietikäinen, Hadid et al. 2011).

Um padrão é definido como o conjunto de zeros e uns que compõem a textura, sendo que neste contexto são definidos alguns padrões como padrões uniformes. Os padrões uniformes são os padrões que ocorrem mais frequentemente em imagens, sendo criado um rótulo (*label*) independente para cada um destes padrões. Por exemplo, se existirem 256 padrões e 58 forem uniformes vão existir 59 diferentes rótulos (58 rótulos independentes para os padrões uniformes e um para todos os padrões não-uniformes). Numa vizinhança circular de (8,1) – 8 *pixels* num raio de 1 *pixel* – os padrões uniformes representam pouco menos de 90 % de todos os padrões e cerca de 70 % ao utilizar uma vizinhança circular de (16,2) (Ojala, Pietikainen et al. 2002). O histograma - H_i – depois da imagem ser processada (rotulada) - $f_l(x,y)$ - (Figura 49) pode ser definido como (Ahonen, Hadid et al. 2004):

$$H_i = \sum_{x,y} I\{f_l(x,y) = i\}, i = 0, \dots, n-1 \quad (2.43)$$

em que n são os diferentes rótulos produzidos pelo operador LBP e:

$$I\{A\} = \begin{cases} 1, A \text{ é } verdadeiro \\ 0, A \text{ é } falso \end{cases} \quad (2.44)$$

Como foi referido anteriormente os histogramas contêm informação sobre a distribuição dos padrões que constituem a textura, mas para uma representação eficaz temos de ter a localização espacial destas. Com este objetivo a imagem é dividida em regiões ($R_0, \dots, R_{m-1}$) - Figura 50 – sendo o histograma com informação de localização definido por (Ahonen, Hadid et al. 2004):

$$H_{i,j} = \sum_{x,y} I\{f_l(x,y) = i\} I\{(x,y) \epsilon R_j \; i = 0, \dots, n-1, j = 0, \dots, m-1 \quad (2.45)$$

(a) (b)

Figura 50 – Exemplo de divisão de uma imagem em regiões: (a) Janela 7x7 e (b) Pesos usados para a métrica de dissimilaridade (preto - peso 0 / cinzento escuro – peso 1 / cinzento claro – peso 2 / branco - peso 4) (Ahonen, Hadid et al. 2004)

Quando se divide uma imagem em regiões, é óbvio que algumas regiões vão ter mais informação que outras. No caso da Figura 50 e.g. a região dos olhos vai ter mais informação do que outras por forma a distinguir se estamos perante uma face humana ou não. Para tal é adotada normalmente uma métrica de dissimilaridade entre os histogramas da imagem – S – e os histogramas do modelo – M da seguinte forma (Ahonen, Hadid et al. 2004, Hadid, Pietikainen et al. 2004):

$$\chi_w^2(S,M) = \sum_{i,j} w_j \frac{(S_{i,j}-M_{i,j})^2}{S_{i,j}+M_{i,j}} \tag{2.46}$$

Em que w_j é o peso para a região j.

2.4.2. *HAAR-LIKE FEATURES*

Assume-se neste contexto que um objeto vai ser testado numa janela de WxH *pixels*, como representado na Figura 51, sendo um retângulo especificado por (Lienhart, Kuranov et al. 2003):

$$r = (x, y, w, h, \alpha) \tag{2.47}$$

Com $0 \leq x, x+w \leq W$, $0 \leq y, y+h \leq H$, $x, y \geq 0$, $w, h > 0$, $\alpha \in \{0°, 45°\}$ sendo a soma dos *pixels* constituintes do retângulo dada pela função $SomaRec(r)$.

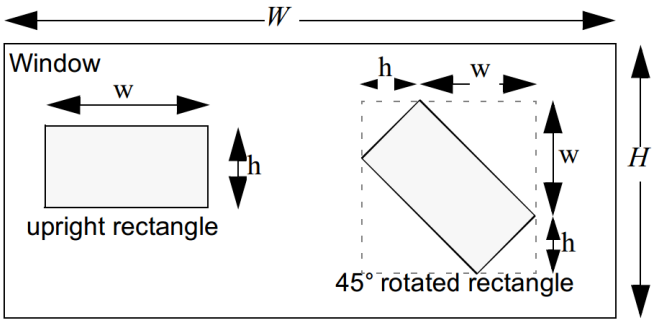


Figura 51 – Exemplo de retângulo direito e rodado 45 graus (Lienhart, Kuranov et al. 2003)

Uma característica é calculada da seguinte forma (Lienhart, Kuranov et al. 2003):

$$Caract_I = \sum_{i=1}^{N} w_i \, . SomaRec(r_i) \tag{2.48}$$

com I a corresponder à característica em análise, $w_i \in \Re$, r_i aos retângulos que constituem a característica e N ao número de retângulos utilizado (r_i e N são arbitrariamente

escolhidos). Isto permite quase combinações infinitas, sendo reduzido por razões práticas de acordo com o seguinte (Lienhart, Kuranov et al. 2003):

- Apenas combinações ponderadas da soma de *pixels* de dois retângulos é considerada (i.e., N = 2);
- Os pesos têm sinais opostos, e são usados para compensar a diferença no tamanho de área entre os dois retângulos. Para retângulos não sobrepostos temos $-w_0 \,.\acute{A}rea(r_0) = w_1 \,.\acute{A}rea(r_1)$. Podemos então definir sem restrições que $w_0 = -1$ e obter $w_1 = \acute{A}rea(r_0)/\acute{A}rea(r_1)$.

Estas restrições levam a 14 protótipos de características que se encontram representados na Figura 52. Estes protótipos são dimensionados de forma independente na vertical e na horizontal, gerando assim um conjunto completo de características. As características em linha podem ser calculadas utilizando apenas dois retângulos, assumindo que o primeiro – r_0 – engloba o retângulo preto e branco e o segundo - r_1 – representa a área a preto. Por exemplo, a característica em linha (0° - 2a da Figura 52) com uma altura total de 2 e uma largura de 6 (Figura 53) a partir do canto superior esquerdo de coordenada - $P_{\sup\_esquerdo} = (5,3)$ - pode ser escrita como (Lienhart, Kuranov et al. 2003):

$$Caract_I = -1 \cdot SomaRec(5,3,6,2,0°) + 3 \cdot SomaRec(7,3,2,2,0°) \tag{2.49}$$

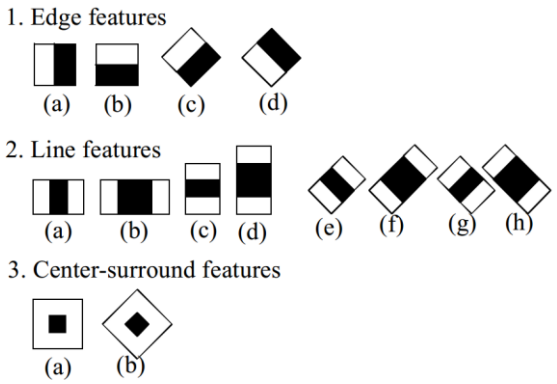


Figura 52 – *Haar-like features* – Áreas a preto têm peso negativo e branco positivo (Lienhart, Kuranov et al. 2003).

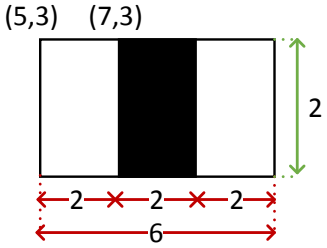


Figura 53 – Exemplo de característica em linha.

Inicialmente as características desenvolvidas foram as (Papageorgiou, Oren et al. 1998, Mohan, Papageorgiou et al. 2001, Viola and Jones 2001): 1a,1b,2a,2c e 4a (Figura 52). O número de características verticais possíveis pode ser calculado da seguinte forma (Lienhart, Kuranov et al. 2003):

$$XY.\left(W+1-w\,\frac{X+1}{2}\right).(H+1-h\frac{Y+1}{2}) \tag{2.50}$$

com $X=[W/w]$ e $Y=[H/h]$ a serem os fatores de escala máximo na direção de x e y respetivamente, w é a largura da característica, h a altura da característica, W a largura da imagem e H a altura da imagem. Uma característica rodada 45 graus gera o seguinte número de características (Lienhart, Kuranov et al. 2003):

$$XY.\left(W+1-z\,\frac{X+1}{2}\right).(H+1-z\frac{Y+1}{2}) \tag{2.51}$$

com $z=w+h$. No caso de uma imagem quadrada de 24 x 24 *pixels* o número de características possíveis é a seguinte:

Tipo de Característica:	w/h:	X/Y:	Possibilidades:
1a ; 1b	2/1 ; 1/2	12/24 ; 24/12	43200
1c ; 1d	2/1 ; 1/2	8/8	8464
2a ; 2c	3/1 ; 1/3	8/24 ; 24/8	27600
2b ; 2c	4/1 ; 1/4	6/24 ; 24/6	20736
2e ; 2g	3/1 ; 1/3	6/6	4356
2f ; 2h	4/1 ; 1/4	4/4	3600
3a	3/3	8/8	8464
3b	3/3	3/3	1521
Soma:			117941

Tabela 3 – Número de características dentro de uma janela de 24 x 24 – Exemplo (Lienhart, Kuranov et al. 2003).

2.4.3. *HISTOGRAMA DE GRADIENTES ORIENTADOS* - HOG

O histograma de gradientes orientados – *histograma of oriented gradients* – é utilizado para construir uma caracterização da estrutura espacial. A ideia base é que a aparência e forma local de um objeto pode ser caraterizado por uma distribuição local de intensidades e orientações de gradiente ou pelas direções dos seus cantos, mesmo sem um conhecimento preciso dos gradientes e direções dos cantos correspondentes (Dalal and Triggs 2005, Szeliski 2010, Prince 2012).

Na prática a imagem é dividida em pequenas regiões chamadas de células, sendo utilizadas mascaras unidimensionais (1D) da derivada discreta no eixo horizontal e vertical calculando assim o gradiente para cada *pixel* e são agrupados os *pixels* de cada região por forma a formar as células (Figura 54). Após a criação de células são criados blocos (Figura 55), agrupando células de uma determinada região. O resultado final é uma lista de histogramas de todas as células de todos os blocos. O histograma final apresenta uma atenuação das variações locais de iluminação ou de contraste através da normalização de cada histograma de acordo com os valores obtidos (Dalal and Triggs 2005, Szeliski 2010, Prince 2012).

A normalização na forma original do cálculo do HOG era feita utilizando $L_2 - hys$ (Lowe 2004), sendo esta atualmente feita aplicando $L2 - norm$ de acordo com a seguinte equação (Dalal and Triggs 2005, Szeliski 2010):

$$L2 - norm = \frac{v}{\sqrt{||v||_2^2 + e^2}} \tag{2.52}$$

com v a ser o vetor não normalizado (histograma) do descritor (*descriptor*) e e é uma constante de valor pequeno. É ao conjunto de blocos normalizados do descritor que se chama histograma dos gradientes orientados – HOG.

Figura 54 – Exemplo de imagem de entrada (a), magnitude do gradiente (b) e magnitude das células (Cruz, shiguemori et al. 2013).

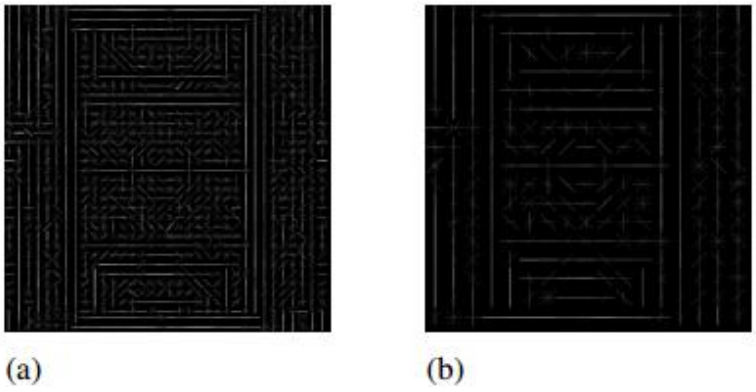


Figura 55 – Histograma de orientação das células (a) e blocos (b) (Cruz, shiguemori et al. 2013).

2.5. DETEÇÃO DE CANTOS

Os cantos – *corners* - são pontos característicos numa imagem e são considerados características de baixo nível – *low-level features*, sendo estes altamente discriminativos e usados muitas vezes para detetar objetos ou fazer correspondência entre imagens. As características de baixo nível (Figura 56) são consideradas características básicas que podem ser extraídas automaticamente de uma imagem sem qualquer informação sobre a forma – relações espaciais (Nixon 2008, Davies 2012). O conceito de pontos de interesse ou ponto-chave define um canto com uma determinada vizinhança, covariância de acordo com alguma forma de transformação (Forsyth and Ponce 2012).

Para calcular os cantos de uma imagem torna-se essencial abordar a deteção de contornos, em que as diferenças de gradiente são calculadas por forma a obter as zonas de gradiente máximo. Existem atualmente métodos bem estudados que o permitem fazer como e.g. (Nixon 2008, Davies 2012): *Sobel*, *Canny*, *Prewitt*, etc. Muita da análise atualmente existente é baseada em contornos visto a sua deteção não ser influenciada por variações de iluminação geral (Nixon 2008). Uma diferença de intensidade de gradiente pode ser facilmente calculada analisando dois pontos adjacentes em x e y de acordo com (Nixon 2008, Davies 2012):

$$Gx_{x,y} = |P_{x,y} - P_{x+1,y}| \qquad \forall x \in 1, N-1; y \in 1, N \qquad (2.53)$$

$$Gy_{x,y} = |P_{x,y} - P_{x,y+1}| \qquad \forall x \in 1, N; y \in 1, N-1 \qquad (2.54)$$

Para detetar os contornos horizontais é utilizado $Gy_{x,y}$, os verticais $Gx_{x,y}$ e combinando os dois é possível obter os contornos horizontais e verticais em simultâneo de acordo com a Figura 57.

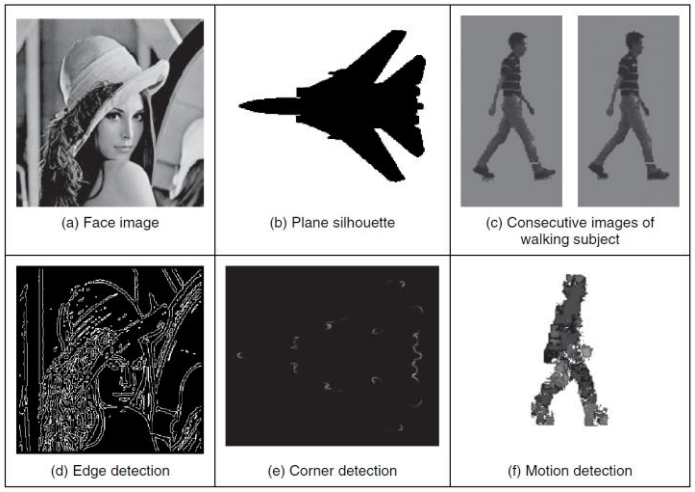


Figura 56 – Características de baixo nível – Exemplos (Nixon 2008).

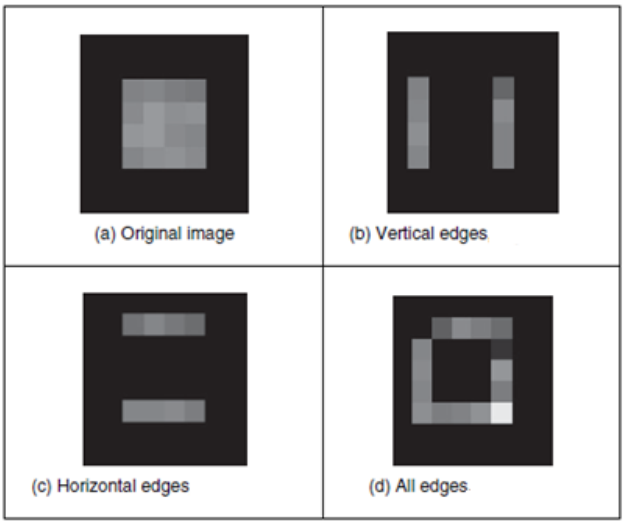


Figura 57 – Deteção de contornos (Nixon 2008).

O objetivo da deteção de contornos é obter uma imagem binária onde os valores que não são zero representam a presença de um contorno na imagem. Também existe a possibilidade de obter informação de orientação e a escala associada ao gradiente (medida de intensidade). O detetor de contornos utilizado no subcapítulo 3.2 é o detetor de *Sobel*, em que à semelhança do exposto na Figura 57 permite calcular o gradiente segundo x e y da seguinte forma (Ma, Soatto et al. 2004, Davies 2012, Prince 2012, Dawson-Howe 2014):

$$F_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.55}$$

$$F_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.56}$$

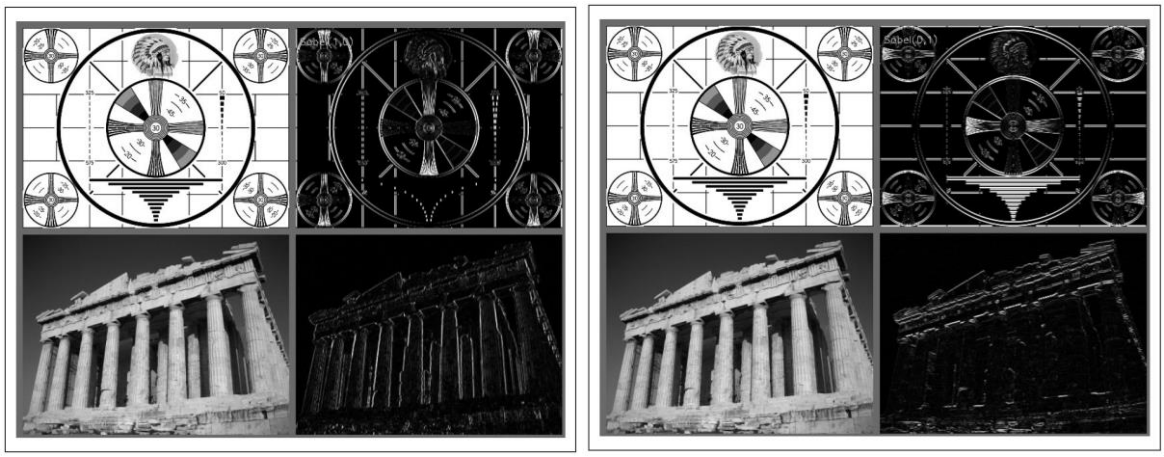


Figura 58 – Filtro de *Sobel* – dimensão x (esq.) e dimensão y (dir.) (Bradski and Kaehler 2008).

Os cantos numa imagem são posições (pontos) que contêm informação visual e podem ser obtidos de diferentes imagens do mesmo objeto. Estes pontos são localmente únicos, estando o seu desenvolvimento inicial ligado à visão estereoscópica procurando assim identificar pontos que possam ser comuns em duas imagens tiradas da mesma cena mas com uma linha de base distinta (Davies 2012).

Um exemplo de detetor de cantos é o detetor de *Harris* que considera os gradientes locais na horizontal e vertical à volta de cada ponto (equação (2.57)), tendo como objetivo encontrar pontos cuja intensidade varie nos dois sentidos (Ma, Soatto et al. 2004, Cyganek and Siebert 2011, Davies 2012, Prince 2012, Subramanyam 2013, Dawson-Howe 2014).

$$S_{i,j} = \sum_{m=1-D}^{i+D} \sum_{n=j-D}^{j+D} w_{mn} \begin{bmatrix} h_{mn}^2 & h_{mn}v_{mn} \\ h_{mn}v_{mn} & v_{mn}^2 \end{bmatrix} \tag{2.57}$$

com $S_{i,j}$ a ser o *structure tensor*[28] da imagem na posição (i, j), que é calculado numa região quadrada de tamanho $(2D + 1) \times (2D + 1)$ à volta da posição atual. O termo h_{mn} representa a resposta horizontal a um filtro derivativo (e.g. *Sobel*) na posição (m, n) e o termo v_{mn} representa a resposta vertical a um filtro derivativo. O termo w_{mn} representa o peso que ajusta a contribuição das posições que estão longe do *pixel* central (i, j).

Para identificar o canto o detetor de *Harris* considera os valores singulares λ_1, λ_2 do *structure tensor* da imagem. Se os valores singulares são pequenos a região à volta do ponto é suave e o ponto não é escolhido. Se um tem valor elevado e o outro não o gradiente da imagem está a variar num sentido e no outro não, no entanto se os dois valores são elevados o gradiente está a mudar nos dois sentidos e o ponto é classificado como canto.

O detetor de *Harris* não calcula diretamente os valores singulares, mas avalia um critério que faz o mesmo mas de uma forma mais eficiente, esse critério é definido por (Ma, Soatto et al. 2004, Cyganek and Siebert 2011, Davies 2012, Prince 2012, Subramanyam 2013, Dawson-Howe 2014):

$$c_{i,j} = \lambda_1\lambda_2 - \kappa(\lambda_1^2 + \lambda_2^2) = \det\left[S_{i,j}\right] - \kappa \cdot trace[S_{i,j}] \tag{2.58}$$

com κ a ser uma constante – valores variam entre 0,04 e 0,15 – sendo que se o valor de $c_{i,j}$ for maior que um determinado limiar de deteção o ponto é classificado como canto.

O detetor de cantos utilizado no capítulo III e IV é o FAST – *Features from Accelerated Segment Test* – proposto por *Edward Rosten* e *Tom Drummond* (Rosten and Drummond

[28] É uma matriz derivada do gradiente de uma função. Resume a direção do gradiente na vizinhança de um ponto especificado e o grau de coerência das orientações do gradiente.

2005, Rosten and Drummond 2006, Rosten, Porter et al. 2010), sendo uma solução que permite processamento em tempo real, que é o pretendido neste caso. Este detetor de cantos funciona da seguinte maneira (Rosten and Drummond 2005, Rosten and Drummond 2006, Rosten, Porter et al. 2010):

- Selecionar um *pixel* da imagem p (Figura 59) que pretende ser identificado como ponto de interesse ou não, sendo a sua intensidade I_p;
- Seleciona-se um valor de limiar apropriado $limiar_{deteção}$;
- É considerado um círculo de 16 *pixels* (Figura 59) em volta do *pixel* em teste;
- O *pixel* p é um canto se existir um conjunto de n *pixels* contíguos no círculo (Figura 59) que são mais brilhantes que $I_p + limiar_{deteção}$, ou todos mais escuros que $I_p - limiar_{deteção}$;
- Um teste de alta velocidade é proposto para excluir um grande número de não cantos. Este teste examina apenas quatro *pixels* na posição 1,9,5 e 13 (primeiro o 1 e 9 se forem muito brilhantes ou escuros é testado o 5 e 13 - Figura 59). Se p for um canto pelo menos três desses têm de ser mais brilhantes que $I_p + limiar_{deteção}$, ou mais escuros que $I_p - limiar_{deteção}$. Se nenhuma das condições for satisfeita p não é um canto. O teste completo é aplicado aos candidatos que passaram examinando todos os *pixels* do círculo (Figura 59).

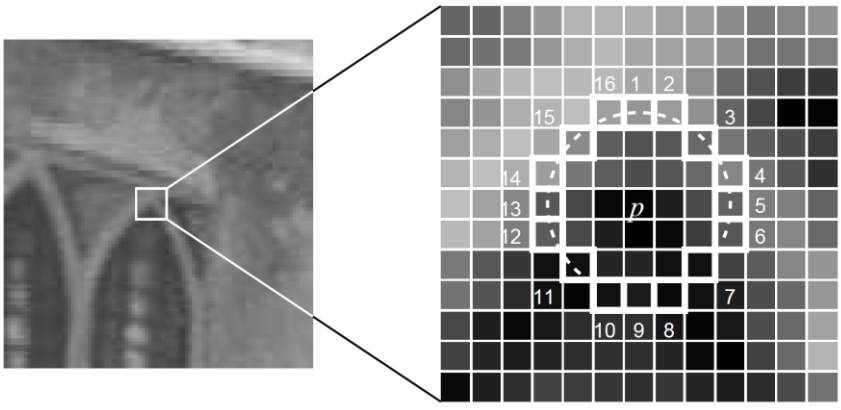


Figura 59 – *Pixels* utilizados no teste de deteção - Exemplo (Rosten and Drummond 2006).

Este detetor apresenta um alto desempenho mas tem as suas fraquezas, elas são (Rosten and Drummond 2006, Rosten, Porter et al. 2010):

- O teste de alta velocidade não generaliza bem quando $n < 12$ (conjunto n representa o número de *pixels* que são contíguos no círculo e são todos mais

brilhantes do que a intensidade do *pixel* candidato - I_p - situado no centro do círculo mais um valor de limiar – t – ou todos mais escuros que $I_p - t$) uma vez que o ponto pode ser um canto se apenas dois dos quatro *pixels* (assumindo *pixels* adjacentes) são ambos significativamente mais brilhantes ou ambos significativamente mais escuros que p (Figura 59);

- A escolha e a ordem dos *pixels* do teste de alta velocidade não é ótima pois a sua eficiência depende da ordenação e tipo de cantos presentes na imagem;
- Os resultados dos testes de alta velocidade são descartados;
- Vários cantos são detetados quando adjacentes uns aos outros.

Os primeiros três pontos são resolvidos recorrendo à aprendizagem máquina e o último usando *non-maximum suppression*[29]. A aprendizagem máquina neste contexto funciona da seguinte forma (Rosten and Drummond 2006, Rosten, Porter et al. 2010):

- Selecionar um conjunto de imagens para treino;
- Correr o algoritmo FAST em cada imagem para encontrar os cantos da imagem;
- Para cada ponto considerado canto são guardados os 16 *pixels* à sua volta em forma de vetor. Isto é feito para todas as imagens do conjunto de treino, obtendo um vetor P;
- Cada *pixel* que constitui o vetor pode ter um dos seguintes estados:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t & (mais\ escuro) \\ s, & I_p - t < I_{p\to x} < I_p + t & (semelhante) \\ b, & I_P + t \le I_{p\to x} & (mais\ brilhante) \end{cases} \quad (2.59)$$

- Dependo do estado, o vetor P é dividido em três conjuntos:
  - P_d;
  - P_s;
  - P_b.
- É definida uma nova variável booleana - K_p – que é verdadeira se P é um canto e falsa caso contrário;
- Usando o algoritmo ID3[30] (Quinlan 1986) para consultar cada subconjunto usando a variável K_p para obter o conhecimento sobre a sua verdadeira classe. Ele

[29] Ao detetar vários cantos em posições adjacentes é considerado apenas o máximo local.
[30] Algoritmo inventado por *Ross Quinlan* usado para gerar uma árvore de decisão a partir de um conjunto de dados.

seleciona o *pixel* que constitui o vetor P que tem mais informação que indique que é um canto, medindo a entropia[31] de K_p;

- Isto é recursivamente aplicado a cada subconjunto até a entropia ser zero;
- A árvore de decisão criada é usada para deteção rápida em outras imagens.

Existem outras aplicações de algoritmos de deteção de contornos e cantos, e surgem a cada dia que passa implementações adaptadas a casos de estudo novos. As implementações utilizadas nesta dissertação basearam-se no filtro de Sobel para deteção de contornos e no algoritmo FAST para deteção de cantos, sendo estes que vão ser explorados ao longo do capítulo III e IV.

2.6. DETEÇÃO BASEADA EM MODELO 3D – FILTRO DE PARTÍCULAS

Muitas aplicações nos dias de hoje têm de detetar com fiabilidade a pose de objetos, a partir de imagens provenientes de câmaras. Detetar um objeto numa sequência de vídeo implica constantemente identificar a sua localização independentemente do movimento do objeto ou câmara. A pose de um objeto rígido, como o que se encontra em estudo, é caracterizado pelos seus 6 graus de liberdade que definem a sua posição no espaço e orientação em relação à câmara (Figura 60).

Usar marcadores para simplificar a tarefa, ou seja, alterar o ambiente em observação adicionando algo que facilite a tarefa nem sempre é possível. Neste caso concreto de estudo, o único conhecimento prévio disponível é o modelo 3D CAD do objeto (Figura 4).

Os filtros de partículas basicamente são um método sofisticado de múltiplas hipóteses para estimação, que funciona normalmente bem para problemas não-lineares (Simon 2006, Haug 2012). Existem muitas variações ao filtro de partículas na sua forma mais simples, pelo que neste subcapítulo vai ser abordado o seu funcionamento simplificado sendo que no capítulo III vão ser exploradas as modificações implementadas para o tornar mais robusto e fiável para este caso concreto de estudo.

A ideia principal é bastante intuitiva e simples, sendo abordado ao longo deste subcapítulo o chamado filtro de partículas gaussiano – GPF - introduzido inicialmente por *Kotechha* e

[31] Para determinar se uma condição de teste realizada é boa, é necessário comparar o grau de entropia do nó pai com o dos nós filhos. O que tiver a maior diferença é escolhido para condição de teste.

Djurić (Kotecha and Djuric 2003, Kotecha and Djuric 2003, Thrun, Burgard et al. 2005). A ideia base é representar a distribuição de probabilidade por uma série de amostras desta distribuição.

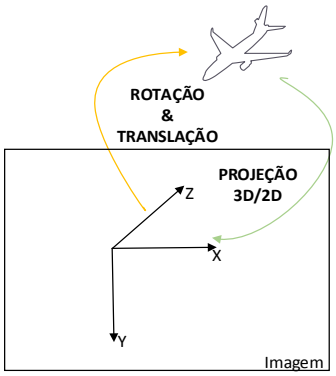


Figura 60 – Seis graus de liberdade – Rotação e translação.

Nos filtros de partículas, as amostras da distribuição posterior são chamadas partículas sendo o vetor do conjunto de amostras da distribuição posterior dado por (Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Challa 2011, Haug 2012):

$$\{x_n\} = \{x_n^{(1)}, x_n^{(2)}, \dots, x_n^{(N_s)}\} \tag{2.60}$$

em que $\{x_n\}$ representa o conjunto de amostras discretas – partículas (cada partícula é um vetor de estado do sistema) - (que tentam aproximar) da distribuição posterior (distribuição contínua), N_s é o número de partículas (amostras discretas) no conjunto $\{x_n\}$ e n corresponde a uma "amostragem" (instante de tempo - t_n). O vetor de observação (assumindo que a relação analítica entre o vetor de observação e o vetor de estado no instante de tempo t_n é conhecida) no instante n é dado por:

$$z_n = h(x_n, v_n) \tag{2.61}$$

em que h corresponde à função determinística[32] de observação, sendo este modelo de medição afetado pelo ruido v_n no instante n. Para abordar o filtro de partículas é necessário abordar as equações fundamentais de *Bayes*, sendo este um processo de dois passos. A estimativa da densidade posterior é dada por (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$p(x_n|z_{1:n}) \propto cp(z_n|x_n)p(x_n|z_{1:n-1}) \tag{2.62}$$

[32] Uma função determinística é uma função que retorna sempre os mesmos valores quando são utilizados os mesmo valores de entrada.

E o passo de predição abrangendo a estimativa da densidade preditiva (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$p(x_n|z_{1:n-1}) = \int p(x_n\,|x_{n-1})p(x_{n-1}|z_{1:n-1})dx_{n-1} \tag{2.63}$$

O conceito fundamental do GPF é assumir que tanto a densidade de probabilidade posterior como a anterior são gaussianas. Temos então (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$p(x_n|z_{1:n}) = \mathcal{N}(x_n; \mu_n, \sum_n) \tag{2.64}$$

$$p(x_n|z_{1:n-1}) = \mathcal{N}(x_n; \bar{\mu}_n, \bar{\sum}_n) \tag{2.65}$$

com a média - $\bar{\mu}_n$ -da função gaussiana da densidade de probabilidade anterior - $p(x_n|z_{1:n-1})$ – e a sua variância - $\bar{\sum}_n$ - a serem calculadas a partir de (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$\bar{\mu}_n = \frac{1}{N_s}\sum_{i=1}^{N_s} x_{n|n-1}^{(i)} \tag{2.66}$$

$$\bar{\sum}_n = \frac{1}{N_s}\sum_{i=1}^{N_s} x_{n|n-1}^{(i)} x_{n|n-1}^{(i)T} - \bar{\mu}_n\bar{\mu}_n^{\mathrm{T}} + Q \tag{2.67}$$

com Q a corresponder à matriz de covariância do ruído. No início do problema de estimação – t_0 - é gerado um conjunto N_s de vetores de estado ($x_n^{(i)} = [X, Y, Z, \alpha, \beta, \gamma]$), baseados na função de distribuição de probabilidade inicial - $p(x_0)$. Esta função é conhecida (deve ser conhecida) e deve possibilitar que o espaço de pesquisa inicial se adapte o mais fielmente ao problema em estudo, existindo vários métodos de inicialização que permitem aferir esta função em tempo real (Brandão, Bernardino et al. 2011, Choi and Christensen 2011, Periquito, Nascimento et al. 2013). Assumindo que no instante t_0, antes de qualquer observação, temos informação sobre a distribuição de probabilidade inicial e esta é dada por (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$p(x_0) = \mathcal{N}(x_0; \mu_0, \sum_0) \tag{2.68}$$

Para fazer a inicialização do filtro são retiradas amostras - $\{x_0^{(i)}\}_{i=1}^{N_s}$ – provenientes de $p(x_0)$ e são propagadas de acordo com a seguinte equação dinâmica (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$x_{n|n-1}^{(i)} = f_n\left(x_{n-1|n-1}^{(i)}\right) + \mu_n \tag{2.69}$$

com $f_n(\,)$ a corresponder à função que representa o processo e que aplica o efeito de cada parâmetro do vetor de estado do sistema em $n-1$ no vetor de estado do sistema no instante n e μ_n neste contexto de estudo é ruído aditivo gaussiano.

O calculo dos pesos - w_n – neste caso concreto é obtido da seguinte forma (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$w_n \propto \frac{f(n)}{g(n)} = \frac{p(z_n|x_n)\,p(x_n|Z_{1:n-1})}{q(x_n|Z_{1:n})} = \frac{p(z_n|x_n)\mathcal{N}(x_n;\bar{\mu}_n,\bar{\sum}_n)}{q(x_n|Z_{1:n})} \tag{2.70}$$

com $f(n)$ a ser a distribuição do sistema - *target distribution* - e $g(\,n)$ a distribuição proposta[33] - *proposal distribution*. O valor do peso é maior quanto mais próxima for a distribuição proposta à distribuição do sistema. Para um conjunto de amostras discretas - N_s – temos então que (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$\breve{w}_n^{(i)} = \frac{p(z_n|x_n=x_n^{(i)})\mathcal{N}(x_n=x_n^{(i)};\bar{\mu}_n,\bar{\sum}_n)}{q(x_n=x_n^{(i)}|z_{1:n})} \tag{2.71}$$

Os pesos obtidos então normalizados da seguinte forma (Doucet, De Freitas et al. 2001, Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Doucet and Johansen 2009, Challa 2011, Forsyth and Ponce 2012, Haug 2012):

$$w_n^{(i)} = \frac{\breve{w}_n^{(i)}}{\sum_{i=1}^{N_s}\breve{w}_n^{(i)}} \tag{2.72}$$

A nova distribuição de amostras é então obtida através de:

$$\hat{x}_n = \sum_{i=1}^{N_s} w_n^{(i)} x_n^{(i)} \tag{2.73}$$

$$P_n^{xx} = \sum_{i=1}^{N_s} w_n^{(i)}\, x_n^{(i)} x_n^{(i)\mathrm{T}} - \hat{x}_n^{(i)}\hat{x}_n^{(i)\mathrm{T}} \tag{2.74}$$

O passo mais importante do filtro de partículas é a parte da reamostragem – *resampling* – onde de acordo com o peso das partículas anteriormente utilizadas um novo conjunto N_s de partículas é criado. Em cada iteração – na fase de reamostragem – vão ser criadas novas partículas nas zonas em que a densidade de probabilidade obtida foi mais elevada,

[33] Também designada pela expressão $q(x_n|Z_{1:n})$.

existindo claramente a ideia da sobrevivência do mais forte (teoria de evolução), das partículas que apresentem maior peso.

Na Figura 63 é possível visualizar a distribuição do sistema – *target distribution* $f(\,)$ – e a distribuição proposta - *proposal distribution* $g(\,)$. O objetivo é aproximar a distribuição proposta da distribuição do sistema - Figura 63 (a), para tal são geradas amostras da distribuição proposta - Figura 63 (b) - e são quantificadas por peso (neste caso $f(x)/g(x)$) - Figura 63 (c). O passo seguinte seria a reamostragem, sendo que gradualmente $g()$ se iria aproximar de $f()$. Casos em que distribuição do sistema é multimodal (Figura 62) é preciso ter isso em conta no desenho e implementação do filtro, evitando assim ficar preso num mínimo local na fase de reamostragem. O GPF pode ser representado pelo diagrama de blocos da seguinte figura:

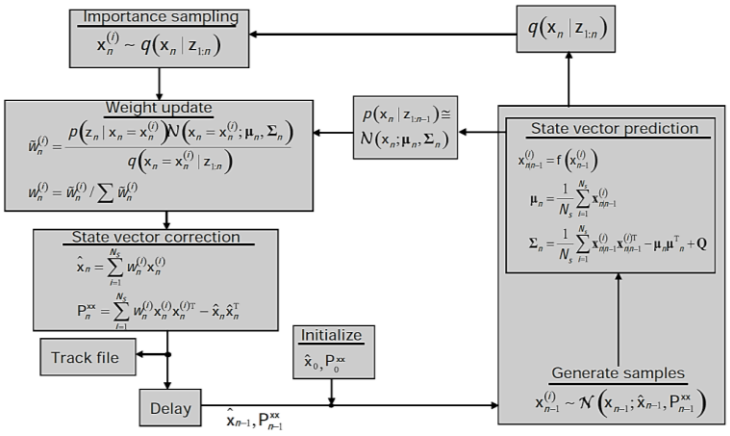


Figura 61 – Diagrama de blocos – GPF (Haug 2012).

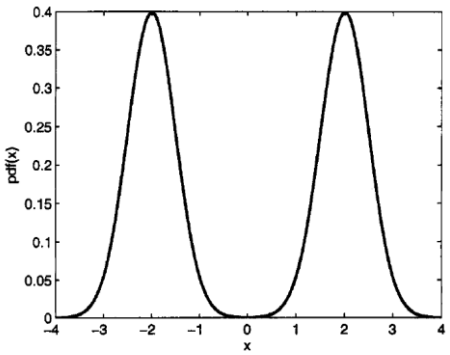


Figura 62 – Função densidade de probabilidade multimodal - Exemplo (Simon 2006).

Os filtros de partículas são baseado em aproximações encontrando-se assim sujeito a erros, sendo que para minimizar os erros existentes existem variações às abordagens básicas do filtro. Existem quatro fontes de erro de aproximação neste tipo de filtros, elas

são as seguintes (Arulampalam, Maskell et al. 2002, Thrun, Burgard et al. 2005, Simon 2006, Doucet and Johansen 2009):

- O número de partículas utilizadas é finito, o que faz com que a reamostragem (dependendo do caso de estudo) leve a erros de aproximação;
- A fase de reamostragem pode levar a uma perca de diversidade da solução, se não existirem leituras atuais por parte e.g. sensores;
- Divergência entre a distribuição do sistema e a distribuição proposta, e.g. se o modelo de movimento introduzido não for adequado à observação vai existir uma divergência;
- Privação de partículas – *particle deprivation* – quando se faz estimativa num espaço de alta dimensão podem não existir partículas na vizinhança do estado correto. Quando o número de partículas é pequeno relativamente ao espaço de todos os estados com probabilidade elevada.

Existem múltiplas implementações deste tipo de filtragem no campo da visão artificial (Nummiaro, Koller-Meier et al. 2003, Barrera, Cañas et al. 2005, Ng and Delp 2009, Sugandi, Kim et al. 2009, Taiana, Santos et al. 2010, Jalal and Singh 2012), não sendo a sua implementação nova neste campo. Vai ser desenvolvida uma arquitetura baseada em algoritmos genéticos adaptando o filtro de partículas conforme explicado no capítulo III.

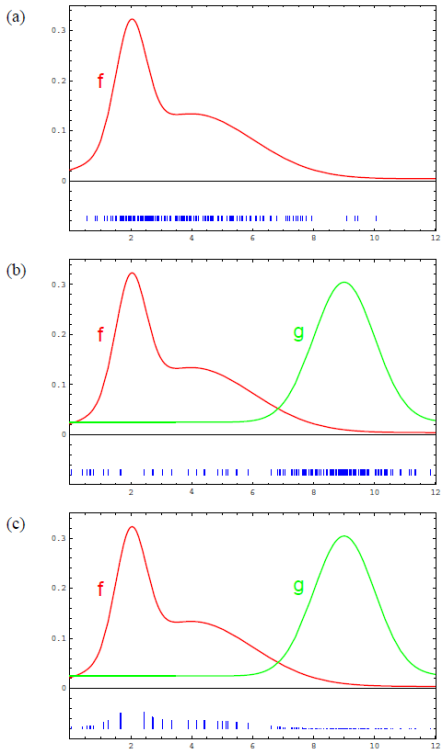


Figura 63 – Ilustração – *Resampling* (Thrun, Burgard et al. 2005).

2.6.1. FILTRO DE PARTÍCULAS VS. ALGORITMOS GENÉTICOS

O problema da pouca diversidade das amostras criadas – perda de diversidade criada na fase de reamostragem - pela utilização dos filtros de partículas, leva a que tenham de ser exploradas novas abordagens e estratégias de reamostragem. Este *empobrecimento* da solução na reamostragem deve ser evitado, pois faz com que se gastem recursos a explorar possibilidades (números de possibilidades/partículas é finita) que contribuem pouco para a melhoraria da estimativa final.

Ao efetuar reamostragem é inevitável a comparação entre a sobrevivência das melhores possibilidades e a teoria da evolução (teoria de evolução de Darwin[34]) de onde os algoritmos genéticos se basearam. É aqui que entra a utilização híbrida, em que os operadores de cruzamento – *crossover* - e mutação – *mutation* - são incorporados para reamostragem de novas partículas (Goldberg and Deb 1991, Uosaki, Kimura et al. 2003, Kwok, Fang et al. 2005, Carmi, Godsill et al. 2009, Park, Hwang et al. 2009, Xiaowei, Hong et al. 2013). Com maiores ou menores *artifícios* na sua utilização, a combinação é feita através incorporação destes operadores.

A arquitetura geral (Cagnoni, Lutton et al. 2007, Park, Hwang et al. 2007) do algoritmo genético é a seguinte:

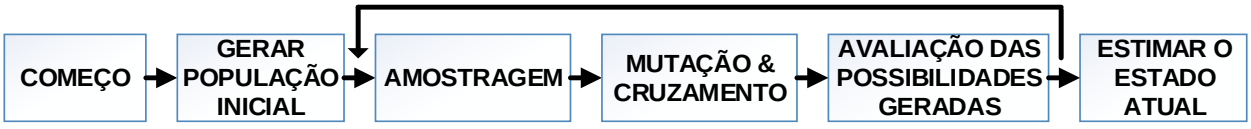


Figura 64 – Algoritmo genético – Arquitetura geral.

Basicamente o algoritmo genético (Figura 64) divide-se em três fases (Cagnoni, Lutton et al. 2007, Park, Hwang et al. 2007):

- **Amostragem ou seleção de candidatos** – Nesta fase uma nova população de cromossomas[35] é gerada a partir da população de cromossomas anterior - X_{t-1} – de acordo com uma função de distribuição de probabilidade de transição do estado;

[34] *Charles Darwin* autor da famosa teoria da evolução das espécies.
[35] Nome dado a uma possibilidade do conjunto de análise - partícula.

- **Operadores genéticos** – Nesta fase são feitas operações como o cruzamento e mutação à nova população de cromossomas criada na fase anterior;
- **Reamostragem** – Nesta fase é feita uma análise às possibilidades geradas – cromossomas – calculando o peso para cada cromossoma (semelhante ao efetuado no subcapítulo 2.6.) e são reamostrados de acordo com o seu peso. Sendo que a nova população representa a densidade de probabilidade posterior.

A utilização dos algoritmos genéticos neste contexto vai ser utilizado na fase de reamostragem, através dos operadores de cruzamento e mutação criados especificamente para o efeito. Sendo a sua utilização neste caso concreto de estudo abordada detalhadamente ao longo do subcapítulo 3.2.4.

2.7. FILTRAGEM TEMPORAL – FILTRO DE KALMAN *UNSCENTED*

A filtragem temporal tem como objetivo utilizar medições de grandezas ao longo do tempo (afetadas de incerteza e ruído) e gerar resultados que se aproximem dos valores reais das grandezas medidas e valores associados. É possível assim neste caso concreto de estudo extrair informação de uma sequência de *frames*, melhorando assim o desempenho global do sistema.

Para este efeito foi criado à cerca de 50 anos o filtro de Kalman (KF) na sua forma mais básica (sistemas lineares), e é ainda nos dias de hoje um dos mais importantes e comuns algoritmos de fusão de dados (Faragher 2012). Este foi nomeado em homenagem a Rudolf Kálmán, estando o seu grande sucesso associado ao baixo nível de processamento necessário, propriedades recursiva e o seu estatuto de estimador ótimo para sistemas lineares de uma dimensão (Anderson and Moore 2012, Faragher 2012). O uso típico deste tipo de filtragem são: a suavização (*smoothing*) de dados ruidosos e a estimação de parâmetros de interesse. As aplicações incluem sistemas de localização GPS[36], sistemas de navegação espacial (e.g. a missão Apollo que levou Neil Amstrong à lua e o trouxe), etc. (Faragher 2012).

O UAV (objeto rígido) vai ter movimento de translação e rotação ao longo do tempo (conforme descrito ao longo do subcapítulo 3.3.2.), mas o filtro de Kalman na sua forma

[36] Sigla que designa *Global Positioning System* – Sistema de Posicionamento Global.

mais básica não permite resolver a relação não-linear existente entre a orientação estimada e a medição esperada de orientação (Kraft 2003). Este problema é resolvido ao utilizar uma variação do filtro de Kalman denominado filtro de Kalman *unscented* (UKF), sendo uma extensão do KF para processos e modelos de medição não-lineares. Existe também outra extensão do KF que pode ser aplicado a este tipo de problemas o filtro de Kalman estendido (EKF), sendo a principal diferença é que o UKF aproxima a função de densidade de probabilidade gaussiana por um conjunto de pontos – denominados pontos sigma (vão ser explicados ao longo deste subcapítulo) – enquanto o EKF lineariza as equações não-lineares do modelo. Isto leva a que o UKF tenha resultados mais precisos (usa as equações originais) e com um tempo de computação inferior (não é necessário calcular a matriz Jacobiano – matriz de derivadas parciais) (Ficocelli and Janabi-Sharifi 2001, Bhat, Seitz et al. 2002, Ribeiro 2004, Yang and Welch 2005, Simon 2006, Lewis, Xie et al. 2008, Janabi-Sharifi and Marey 2010, Särkkä 2013).

Na Figura 65 é projetada uma nuvem de 5000 amostras de uma distribuição gaussiana, sendo esta propagada por uma função altamente não linear - $y = f(x)$ – sendo obtidas a covariância e a média verdadeira (Esquerda da Figura 65). De seguida estas foram calculadas através de linearização – semelhante à utilizada no EKF – sendo que os erros obtidos através desta aproximação de primeira ordem são bem visíveis (Centro da Figura 65). Foi também utilizada uma *unscented transform* – como a utilizada no filtro implementado (Subcapítulo 3.3.3.) – que permitiu uma muito melhor estimativa da realidade (Direita da Figura 65).

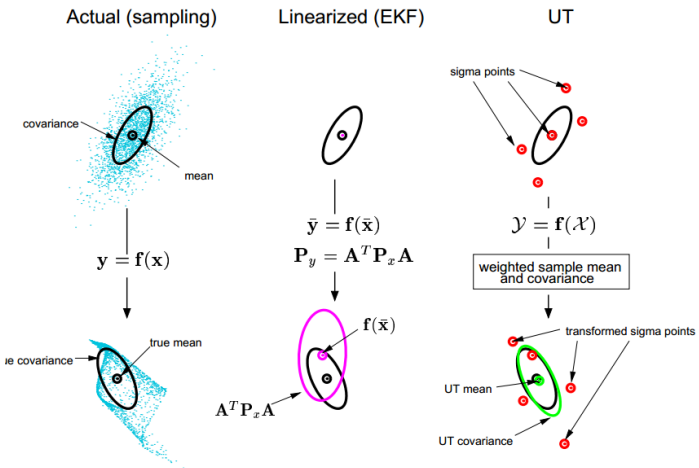


Figura 65 – Diagrama esquemático da – *Unscented transformation* (Van Der Merwe, Doucet et al. 2000).

Considerando a seguinte equação de transição de estado não-linear (Haykin, Haykin et al. 2001, Stenger, Mendonça et al. 2001):

$$X(k+1) = f(X(k), u(k+1), k+1) + v(k+1) \tag{2.75}$$

em que f descreve a dinâmica do sistema, $X(k)$ representa o vetor de estado do sistema no instante te tempo - t_k, $u(k+1)$ é o vetor de controlo de entrada e $v(k+1)$ é o ruído do processo. A matriz de covariância da distribuição do estado é dada por Σ_X. Um conjunto de observações, relacionadas com o vetor de estado, são obtidas através da seguinte equação (Haykin, Haykin et al. 2001, Stenger, Mendonça et al. 2001):

$$Z(k+1) = h(X(k+1), u(k+1), k+1) + w(k+1) \tag{2.76}$$

onde $Z(k+1)$ é o vetor de observação, h é o modelo de observação e $w(k+1)$ é o ruído de medição. O algoritmo do UKF pode ser resumido da seguinte forma (Ficocelli and Janabi-Sharifi 2001, Bhat, Seitz et al. 2002, Ribeiro 2004, Yang and Welch 2005, Simon 2006, Lewis, Xie et al. 2008, Janabi-Sharifi and Marey 2010, Särkkä 2013):

Algoritmo UKF: Descrição geral simplificada

1. Selecionar um conjunto de $2n$ pontos de amostragem $\mathcal{W}_i$, $i = 1,2,\dots 2n$ que representam as colunas da matriz $\pm\sqrt{2n \cdot \gamma \cdot (P_{k-1} + \mathcal{Q})}$. Em que γ representa um factor de peso, P_{k-1} é a covariância do erro estimado anteriormente e $\mathcal{Q}$ a matriz de covariância do ruído no processo;
2. Calcular o conjunto de pontos sigma $\mathcal{X}_i = \hat{x}_{k-1} + \mathcal{W}_i$, em que $\hat{x}_{k-1}$ representa o vetor de estado estimado anteriormente;
3. Aplicar o modelo do processo $A()$ ao conjunto de pontos sigma $\mathcal{X}_i$, de acordo com $\mathcal{Y}_i = A(\mathcal{X}_i)$;
4. Aplicar o modelo de medição $H()$ aos pontos sigma, por forma a projetar estes pontos no espaço de medição $\mathcal{Z}_i = H(\mathcal{Y}_i)$;
5. É calculada a média de $\mathcal{Z}_i$ de acordo com $z_k^- = \sum_{i=1}^{2n} W_i^{(c)} \mathcal{Z}_i$, em que $W_i^{(c)}$ é um fator de peso;
6. Calcular a inovação $\upsilon_k = z_k - z_k^-$ a partir da medição atual z_k e a média da estimativa de medição z_k^-;
7. Atualizar a matriz do ganho de Kalman $K_k(k+1)$;
8. Atualizar a estimativa do vetor de estado $\hat{x}_k = \hat{x}_k^- + K_k \upsilon_k$, em que $\hat{x}_k^-$ corresponde ao vetor de média dos pontos sigma transformados $\mathcal{Y}_i$.

Foi abordado ao longo deste subcapítulo a descrição simplificada do UKF, bem como os motivos da sua utilização. O filtro de Kalman *unscented* implementado vai ser descrito no subcapítulo 3.3.3 de forma mais detalhada, explicando o seu funcionamento e adaptação ao sistema concreto em estudo.

2.8. GPU - ARQUITETURA CUDA

A GPU[37] foi evoluindo e deixou e ser uma solução com funções exclusivas e pouco parametrizáveis para uma solução completamente programável, altamente paralelizável e com muitos núcleos disponíveis para processamento. Uma abordagem extensa a este enorme campo está fora dos objetivos desta dissertação, a abordagem vai ser focada (um pouco à semelhança da filosofia que tem sido seguida ao longo deste capítulo) no que permitiu a sua utilização neste contexto de estudo.

O desenvolvimento do poder de processamento tipicamente era feito através do aumento da frequência de trabalho dos microprocessadores utilizados, existindo assim um aumento de consumo. Este aumento de consumo faz com que o rendimento decresça, tendo o paradigma evoluído para a utilização de processadores em paralelo. Aqui entra a utilização da GPU, sendo que numa arquitetura *Intel Core* i7-4700HQ podemos ter 4 processadores e numa GPU *NVIDIA* GT 750M 384 processadores em paralelo.

As GPUs atuais são o culminar de muito desenvolvimento a nível de *software* e *hardware*, proporcionando nos dias de hoje uma acessibilidade de programação que permite alargar assim o seu leque de aplicações (Owens, Houston et al. 2008, Sanders and Kandrot 2010, Matloff 2011). Não são só utilizadas para fazer processamento gráfico, mas também para outro tipo de cálculos passíveis de serem paralelizados como e.g. imagens médicas, análise computacional de fluidos, etc., existindo claramente diferenças de filosofias de *design* entre a CPU e a GPU como demonstrado pela Figura 66. Fazendo uma comparação entre os dois obtemos (Corporation 2014, TECHPOWERUP 2014):

Característica:	***Intel Core i7-4700HQ***	***NVIDIA GT 750M***
Número de processadores:	4	384
CPU *Clock:*	2,4 GHz	941 MHz
Largura de banda memória:	25,6 Gbyte/s	64,0 Gbyte/s

Tabela 4 – CPU VS GPU – Exemplo.

O objetivo da arquitetura CUDA é proporcionar um nível de abstração entre o *hardware* e o desenvolvimento de *software*, não sendo necessário um conhecimento do seu funcionamento detalhado como pré-requisito.

[37] As GPUs retratadas nesta tese admitem a utilização da plataforma CUDA – *Compute Unified Device Arquitecture* – que vai ser abordada ao longo deste subcapítulo.

CUDA pode ser descrita como uma arquitetura que permite criar aplicações para GPU, permitindo que a partir de funções simples em C – designado de C para CUDA (Figura 67) – estas possam ser diretamente integradas em programas escritos em C *standard* (Sanders and Kandrot 2010, Wilt 2013, Corporation 2014) e compiladas com o compilador – *NVIDIA* C (NVCC) – utilizando e.g. o *Visual Studio* (descrito no Subcapítulo 1.4). Esta facilidade de configuração e utilização levou a que o número de aplicações atualmente existentes tenha aumentado grandemente, podendo utilizar interfaces de programação – API - de e.g. *Python*, *java*, etc. Fazendo com que este "Mundo" comece a ser acessível a todos os que sabem programar e não sabem apenas uma linguagem de programação específica. Mas tudo traz as suas desvantagens esta abstração ao nível do *hardware* pode levar a que este não seja explorado convenientemente, não aproveitando assim toda a sua potencialidade.

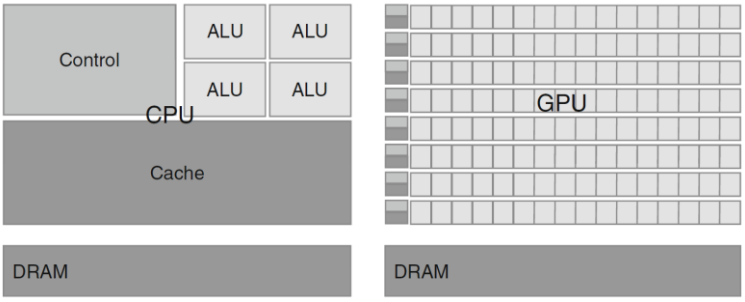


Figura 66 – CPU & GPU diferenças de filosofia no *design* (Kirk and Wen-mei 2012).

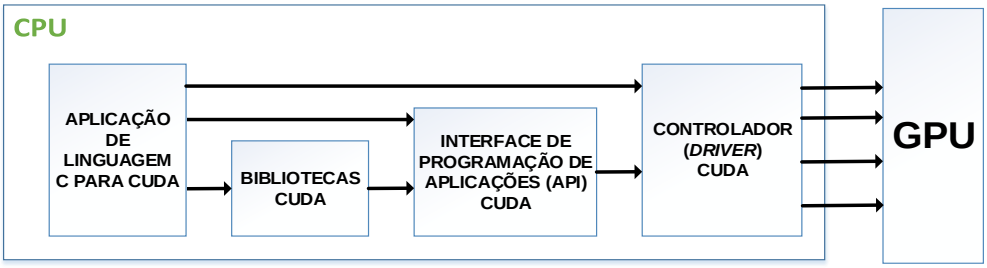


Figura 67 – *Software* CUDA – Esquemático de funcionamento.

Os programas desenvolvidos em CUDA correm em série num *host*[38] – onde está a CPU – mas podem de forma assíncrona chamar *kernels*[39] que são executados em paralelo no dispositivo – GPU. CUDA permite explorar uma série de capacidades, que podem alargar o seu leque de aplicações. Estas capacidades são (Sanders and Kandrot 2010, Farber 2011, Kirk and Wen-mei 2012, Cook 2013, Wilt 2013):

[38] Termo utilizado para designar o computador em que a GPU se encontra instalada.
[39] Função a ser executada na GPU.

- Suporte para múltiplos GPUs;
- Cálculos com *single* ou *double-precision* de vírgula flutuante;
- Acesso aos espaços de memória da GPU;
- Acesso ao temporizador da GPU para *benchmarking*[40] preciso;
- Bibliotecas matemáticas de alta velocidade de baixa precisão (e.g. cosseno);
- Execução simultânea de código no *host* e no dispositivo.

Normalmente a GPU é ligada ao host na sua *motherboard* por um barramento de alta-velocidade e.g. PCIe - P*eripheral Component Interconnect Express*. Isto permite adicionar mais GPUs que vão funcionar de forma assíncrona do processador presente no *host* podendo estes efetuar processamento em simultâneo utilizando o barramento PCIe (Figura 68) para transferência de dados entre os dispositivos (GPU-CPU).

Existem opções como o mapeamento de *pinned Memory* (Figura 69) que permite manter uma região de memória no *host* sincronizada com uma região de memória da GPU. Esta interface pode aumentar o desempenho da aplicação pois este sincronismo entre regiões de memória é feito de forma assíncrona. Existem GPUs de baixa potência e custo que partilham memória com a CPU, ao utilizar esta técnica resulta numa operação de *zero-copy* pois a GPU acede aos dados diretamente (Farber 2011, Matloff 2011, Wilt 2013). Ao mais baixo nível é o controlador – *driver* – que vai controlar estas operações.

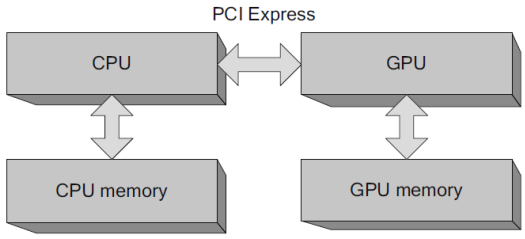


Figura 68 – CPU/GPU - Arquitetura simplificada (Wilt 2013).

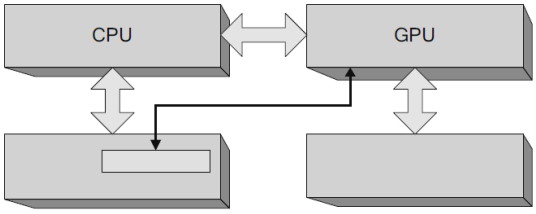


Figura 69 – *Pinned Memory* – Arquitetura simplificada (Wilt 2013).

[40] Este termo é usado para designar a comparação entre parâmetros de desempenho. Ao ser preciso, vou ter uma comparação mais fiável e correta.

O modelo de memória que são utilizados pela interface CUDA para utilização são os seguintes (Sanders and Kandrot 2010, Farber 2011, Matloff 2011, Ruetsch and Fatica 2011, Cook 2013):

- *Global memory;*
- *Shared memory;*
- *Texture memory;*
- *Constant memory;*
- *Local memory / variables.*

A memória global (memória dinâmica de acesso aleatório - *dynamic random access memory* – DRAM) - *Global memory* - é a memória do dispositivo – GPU – que é atribuída ao *host* em código. Esta pode ser lida e escrita pelo *device* e pelo *host*. As variáveis locais (DRAM) – Local variables – são definidas no código local do dispositivo e são guardadas nos registos do chip, se não existirem registos suficientes estas são guardadas na memória local. A memória constante (DRAM) - *Constant memory* - pode ser escrita e lida pelo código presente no *host* mas apenas pode ser lida pelo *device*. A memória de textura (DRAM) - *Texture memory* - é muito semelhante à memória constante mas é apenas de leitura pelo *device* e *host*. A memória partilhada (GPU) – *Shared memory* – é uma memória que pode ser acedida por todos os threads em simultâneo, sendo declarada no código do *device* (Sanders and Kandrot 2010, Farber 2011, Matloff 2011, Ruetsch and Fatica 2011, Cook 2013). Os modelos de memória são resumidos na Tabela 5 e a sua localização é ilustrada na Figura 70.

Memória:	Localização:	Tipo de acesso:	Acesso:	Tempo de vida:
Global	DRAM	Leitura/escrita	Todas *threads* e host	Aplicação
Partilhada	GPU	Leitura/escrita	Todas *threads* em bloco	*Thread*
Textura	DRAM	Leitura	Todas *threads* e host	Aplicação
Constante	DRAM	Leitura	Todas *threads* e host	Aplicação
Local	DRAM	Leitura/escrita	Uma *thread*	*Thread*
Registos	GPU	Leitura/escrita	Uma *thread*	*Thread*

Tabela 5 – Tipos de memória acesso CUDA - resumo.

Os programas em CUDA utilizam o designado por *kernels* que são basicamente sub-rotinas que são chamadas pelo *host* e que são executadas na GPU. Estas sub-rotinas não funcionam como funções e por isso não podem retornar valores e o seu funcionamento é assíncrono, ou seja, estas podem ser executadas na GPU e a CPU não espera que estas terminem para prosseguir com o seu processamento. Para existência de sincronismo entre

CPU-GPU este deve ser claramente executado pelo programador (recorrendo a funções específicas), para perceber quando é que a execução da sub-rotina terminou e utilizar o resultado desse processamento (Farber 2011, Matloff 2011).

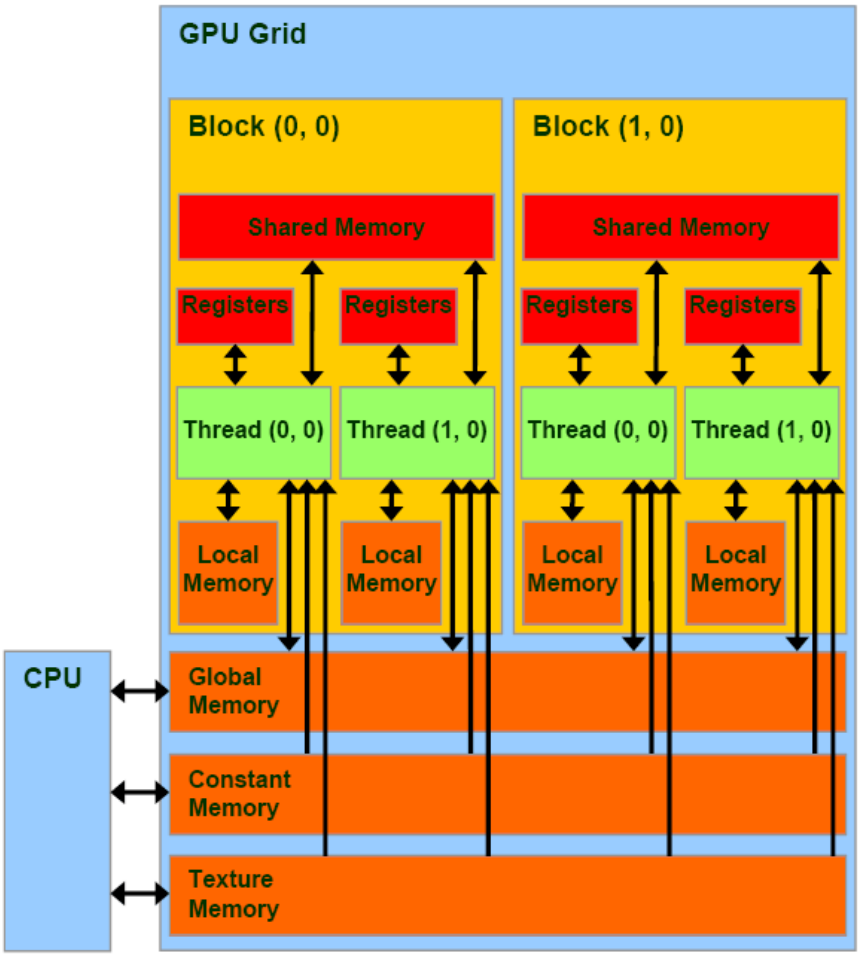


Figura 70 – Tipos de memória acesso CUDA – Localização (Gupta 2012).

A unidade básica de trabalho da GPU é uma *thread* (Figura 71), sendo estas completamente independentes entre si, funcionando como se cada uma tivesse o seu próprio processador com registos e identidade. É possível fazer com que estas comuniquem entre si, através de operações sincronizadas no entanto se não for corretamente executado perde-se velocidade de processamento. Um *kernel* deve utilizar o número máximo de *threads* possível para fazer o processamento, sendo o código executado por *threads* em paralelo, que são agrupados em blocos – 1D, 2D ou 3D – de dimensão B_x, B_y e B_z. Os blocos também são agrupados em 1D ou 2D – G_x e G_y – providenciando assim também um segundo nível mais grosseiro de paralelismo (Figura 71). Um identificador único por cada *thread* pode ser calculado da seguinte forma (Sanders and Kandrot 2010, Farber 2011, Kirk and Wen-mei 2012, Cook 2013, Wilt 2013):

$$tid = t_i + t_j B_x + t_k B_x B_y \tag{2.77}$$

sendo que $t_i, t_j \, e \, t_k$ são os índices x, y e z dos *threads* respetivos. Este identificador é utilizado normalmente para índice de uma estrutura de dados, para garantir que apenas uma *thread* em simultâneo possa aceder a uma determinada parte dos dados – evitar conflitos.

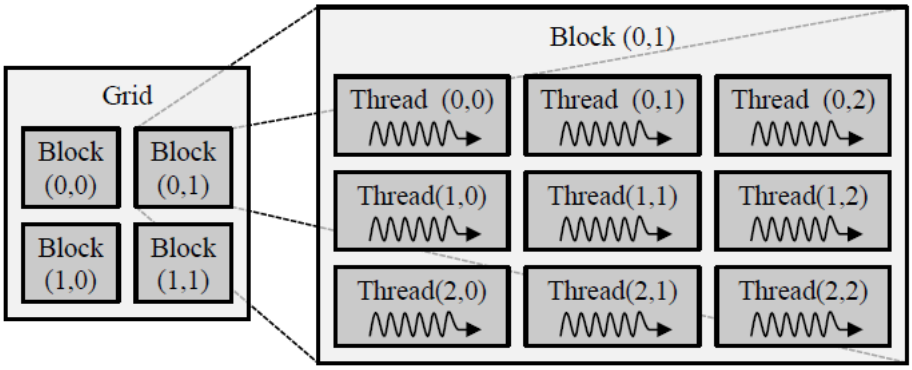


Figura 71 – Hierarquia de linha de execução – Hierarquia dos *Threads* (Brown 2010).

Existem regras que devem ser seguidas por forma a obter o máximo rendimento, estas são (Farber 2011, Cook 2013):

1. Transferir os dados para a GPU e mantê-los lá;
2. Dar à GPU processamento suficiente;
3. Focar na reutilização de dados para evitar limitações na largura de banda.

É importante transferir os dados entre o CPU-GPU e mantê-los lá, pois o barramento PCIe é muito lento quando em comparação com o barramento interno da própria GPU. Esta diferença centra-se normalmente entre as 20x e as 28x (Farber 2011, Cook 2013).

O processamento da GPU pode chegar aos teraflops, efetuando tarefas simples a uma velocidade superior ao que a CPU em si consegue chamar um *kernel*, o que faz com que exista uma perda de desempenho clara, não devendo abusar da utilização das sub-rotinas com tarefas simples e de baixo processamento.

Os dados devem ser reutilizados para evitar assim limitações na largura de banda, e a criação de novos e.g. vetores ou variáveis vai ser sempre menos eficiente do que a reutilização de variáveis já criadas. Esta prática também é valida ao utilizar a CPU.

É impossível abordar esta temática sem abordar a lei descrita por *Gene Amdahl* (Farber 2011, Cook 2013) que criou um modelo que descreve a aceleração ideal que pode

acontecer quando programas em série são convertidos para correr em paralelo. Este modelo apenas é válido se o tamanho do problema se mantiver constante depois de paralelizado, não variando significativamente depois de paralelizado o que nem sempre se verifica na realidade. Este modelo é descrito da seguinte forma (Farber 2011, Kirk and Wen-mei 2012, Cook 2013, Wilt 2013):

$$S(n) = \frac{1}{(1-P)+P/n} \tag{2.78}$$

com n a corresponder ao número de processadores, P à proporção de código que pode ser paralelizável e $(1 - P)$ à porção que não pode ser paralelizado.

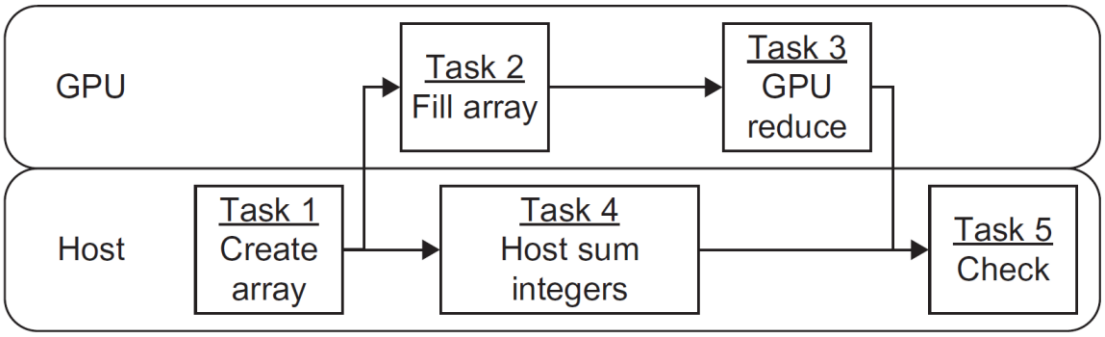


Figura 72 – Utilização híbrida - Assíncrona (Farber 2011).

Como não existe dependência entre as sub-rotinas que correm em GPU e a CPU – assíncronas – a execução ideal centra-se numa utilização que se pode chamar de híbrida. Em que as sub-rotinas na GPU são chamadas e a CPU em simultâneo executa processamento (Figura 72).

Uma aplicação CUDA, como representada na Figura 72, pode ser resumida da seguinte forma:

- Alocar memória no *device* (GPU) e *host*;
- Preparar dados no espaço de memória do *host*;
- Transferir memória entre o *host* e o *device* de forma síncrona;
- Executar os *kernels* necessários de forma assíncrona;
- Continuar a executar código série na CPU em simultâneo com a GPU;
- Transferir resultados do *device* para o *host* de forma síncrona, a CPU aguarda que esta cópia seja finalizada;
- Continuar com o código série na CPU.

A sua utilização vai ser explorada mais à frente no subcapítulo 4.2 – *Rendering* do objeto.

2.9. SISTEMAS DE ATERRAGEM

A maior parte dos sistemas atualmente existentes é baseada numa solução localizada no UAV e não numa solução localizada em terra, o que faz com que o UAV tenha de ter alguma capacidade de processamento associado aumentando por vezes o seu respetivo peso e custo global.

O sistema UAV não vai ser aqui abordado, pois o objetivo é que o sistema seja "transparente" ao tipo de objeto utilizado podendo ser assim o mais genérico possível. Vão ser sim abordados os tipos de aterragem e métodos normalmente utilizados para a aterragem automática de UAVs.

A maioria dos sistemas de aterragem automática com o processamento baseado no UAV utilizam padrões (exemplo na Figura 73) para aterragem (marcadores externos) (Sharp, Shakernia et al. 2001, Saripalli, Montgomery et al. 2002, Saripalli, Montgomery et al. 2003, Merz, Duranti et al. 2006, Lange, Sünderhauf et al. 2008, Saripalli 2009, Wenzel, Masselli et al. 2011, Xiang, Cao et al. 2012, Wu, Johnson et al. 2013, Zhao and Pei 2013), apresentando como principal vantagem a não utilização de sensores externos e.g. GPS que pode ser muito útil em ambientes em que esta informação esteja indisponível, podendo ser o método de aterragem primário ou como *backup* em caso de emergência. Mas também existem sistemas que apenas precisam de detetar o navio, recorrendo e.g. a câmara IR como o representado na Figura 74.

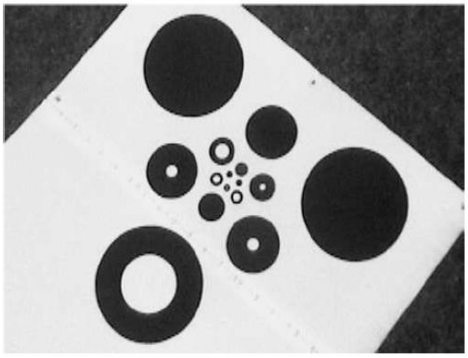


Figura 73 – Exemplo de padrão para aterragem de UAV – câmara de bordo (Merz, Duranti et al. 2006).

Existem também sistemas baseados em terra (Figura 75) baseados em visão estereoscópica na gama dos infravermelhos (Kong, Zhang et al. 2013), em visão estereoscópica (Figura 76) na gama do visível (Hazeldene, Sloan et al. 2004, Moore, Thurrowgood et al. 2009) ou em visão trinocular (Figura 77) utilizando três câmaras (Martínez, Campoy et al. 2009).

Figura 74 – Imagem do navio após processamento (esq.) e localização da câmara (dir.) (Xu, Chen et al. 2011).

Existem também outros sistemas que detetam automaticamente as pistas de aterragem, para efetuar a respetiva orientação (Miller, Shah et al. 2008, Williams and Crump 2012, Gui, Guo et al. 2013). Que em caso de emergência têm a capacidade de detetar um local seguro para a aterragem (Fitzgerald, Walker et al. 2005, Hubbard, Morse et al. 2007, Mejias, Fitzgerald et al. 2009), que navegam e aterram de forma autónoma utilizando marcas naturais da paisagem (Yang and Tsai 1998, Cesetti, Frontoni et al. 2010), entre muitas outras variantes atualmente já estudadas e implementadas.

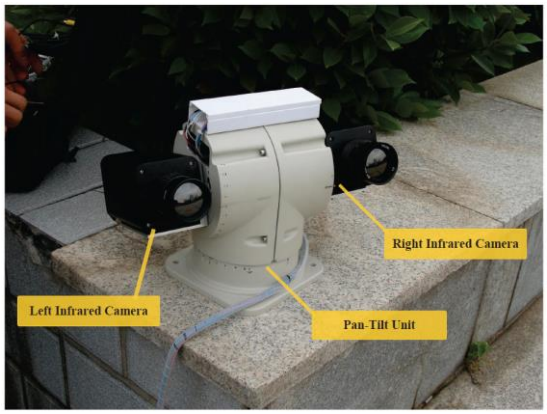


Figura 75 – Exemplo de sistema baseado em terra – visão estereoscópica Infravermelhos (Kong, Zhang et al. 2013).

Figura 76 – Exemplo de sistema baseado em terra – visão estereoscópica espectro visível (Hazeldene, Sloan et al. 2004).

No que concerne aos sistemas de aterragem direcionados para plataformas navais, matéria de estudo desta dissertação, para os sistemas que permitem vertical *Take-off and landing* – VTOL – é muitas vezes simulado fisicamente um navio recorrendo a uma plataforma de 6 graus de liberdade (Garratt, Pota et al. 2009, Sanchez-Lopez, Saripalli et al. 2013) (um navio pode ser simulado recorrendo a 6 graus de liberdade, 3 de translação e 3 de rotação - Figura 78). Um exemplo deste tipo de plataforma está representado na Figura 79 e Figura 80. Por vezes é também utilizado um pequeno robô com movimento associado (Figura 81), por forma a simular o movimento da plataforma (Hérissé, Hamel et al. 2012, Chaves, Wolcott et al. 2013).

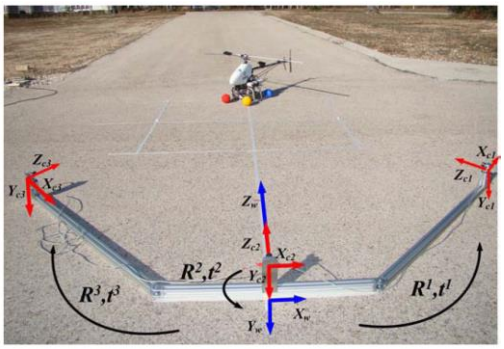


Figura 77 – Exemplo de sistema baseado em terra – trinocular (Martínez, Campoy et al. 2009)

Figura 78 – Movimentos padrão de um navio – eixos de rotação e translação (Riola, Diaz et al. 2011).

Figura 79 – Exemplo de outro tipo de plataforma de 6 graus de liberdade – Simulação de um navio (Garratt, Pota et al. 2009).

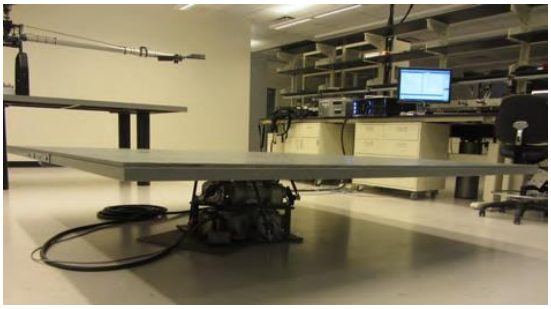


Figura 80 – Plataforma de 6 graus de liberdade – Simulação de um navio (Sanchez-Lopez, Saripalli et al. 2013).

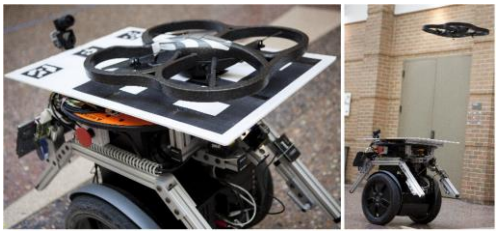


Figura 81 – Exemplo de plataforma móvel, utilizado para simular o movimento de um navio (Chaves, Wolcott et al. 2013).

Na gama do infravermelho (e noutros espectros), é muitas vezes também acrescentado um marcador (perfeitamente compatível com as características da plataforma) como e.g. o representado na Figura 82 em que foram efetuados testes reais com uma marcado do tipo "T".

O objetivo muitas vezes é utilizar sistemas já existentes, pois têm as dimensões adequadas e encontram-se devidamente adaptados para o ambiente de operação. Em caso de incapacidade de utilização de processamento interno, e alguns casos apenas por opção, o processamento pode ser efetuado por uma unidade auxiliar e o seu resultado comunicado ao sistema de controlo do UAV (como se pretende fazer no sistema abordado nesta dissertação)

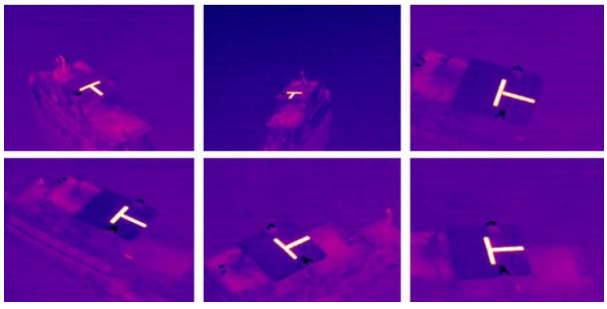


Figura 82 – Imagens infravermelhos de um exemplo de marcador usado no convés de voo de um navio (Xu, Zhang et al. 2009)

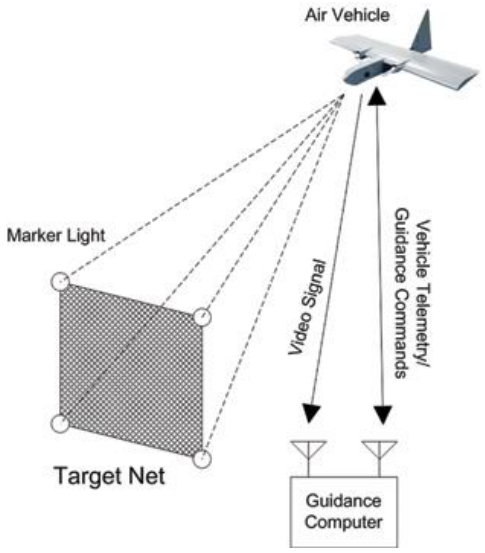


Figura 83 – Representação de um sistema de aterragem automática (Kahn 2010).

Existem mesmo alguns sistemas que efetuam o chamado – determinação do período quiescente – que determinam qual o período em que a ondulação (Figura 84) vai ter menos influência na movimentação do navio (período mais calmo) (Riola, Diaz et al. 2011).

Apesar de ser largamente estudada a aterragem automática de UAV, quer de asa fixa ou de asa rotativa, existem poucas referências a aterragens em navios propriamente ditos sendo uma temática muito específica e explorada por poucos. Apesar de ser uma mais-valia a comunidade naval só muito recentemente se andam a desenvolver e a criar parcerias com empresas e universidades.

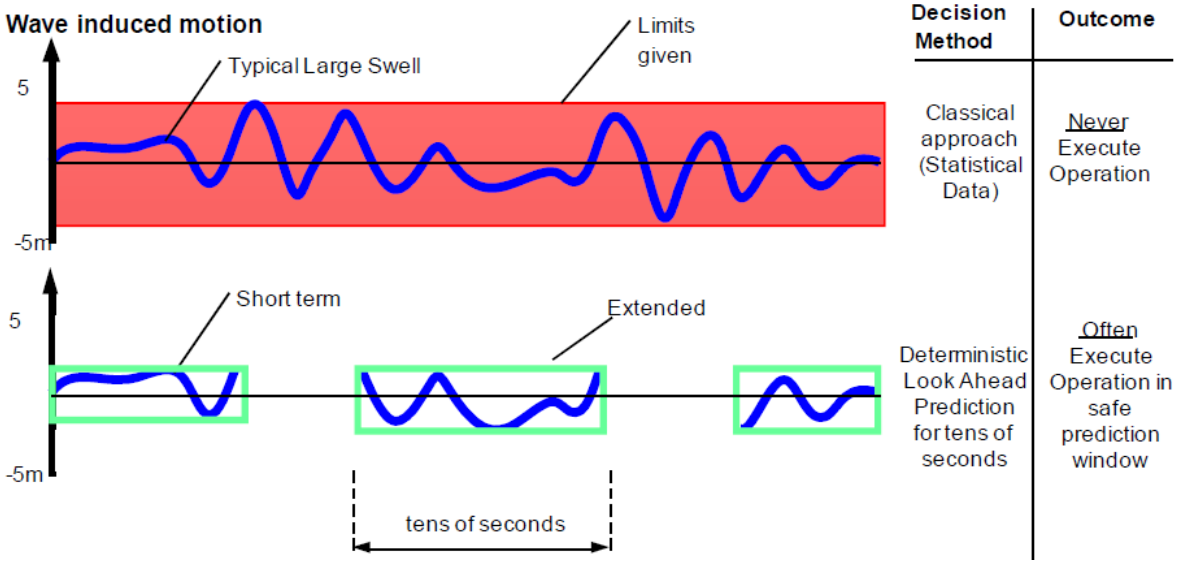


Figura 84 – Princípios básicos da predição do período quiescente (Riola, Diaz et al. 2011).

2.10. CONCLUSÕES OU SÍNTESE

Este capítulo efetua uma abordagem ao fundamento teórico mais relevante utilizado na elaboração desta dissertação, devido à extensão dos temas abordados torna-se inexequível abordar aqui ao pormenor cada temática. Foram então resumidos os temas mais relevantes para a sua correta compreensão.

Foi descrito o modelo geométrico da câmara detalhando o que são os parâmetros intrínsecos e extrínsecos, bem como a distorção e forma de calibração que permite obter os parâmetros para correção da distorção existente.

A visão estereoscópica apesar de não estar presente no sistema final desenvolvido leva a uma compreensão do tema, e de outros métodos de estimar a posição de um objeto.

Os classificadores são largamente usados em estatística e em quase todos os domínios para representar aproximações a um conjunto de dados, criando algo que nos diga como novos elementos seriam classificados consoante as variáveis em estudo. Nesta temática o classificador tem um objetivo semelhante, sendo abordado o seu funcionamento.

Os algoritmos de deteção de cantos que são importantes na arquitetura desenvolvida também são aqui abordados. A deteção baseada num modelo 3D e o filtro de partículas, bem como a sua ligação com os algoritmos genéticos, que são a base da arquitetura desenvolvida são explicados procurando dar a entender o seu funcionamento e como foram adaptadas à realidade de estudo.

É também abordada a filtragem temporal, justificando o porquê da escolha do filtro de Kalman *unscented* e resumindo o seu princípio de funcionamento. É também abordada a arquitetura CUDA necessária para efetuar processamento em tempo real (requisito do sistema) e por fim são abordados os sistemas de aterragem automática recorrendo a visão atualmente existentes.

O capítulo seguinte faz uma descrição geral à arquitetura desenvolvida, cujo desempenho será estudado posteriormente.

(Página intencionalmente deixada em branco)

CAPÍTULO III
DESCRIÇÃO GERAL DO SISTEMA

3.1. INTRODUÇÃO

Este capítulo descreve a arquitetura do sistema desenvolvido para deteção de pose e *tracking*, bem como todos os pormenores considerados relevantes para a sua compreensão utilizando uma arquitetura baseada no modelo 3D do UAV. O problema em estudo é complexo pois envolve a aterragem de um UAV numa plataforma móvel com grandes movimentos laterais, devido às condições de mar adversas, resumindo-se no fundo (após testes de fiabilidade de vários métodos, conforme descrito durante a introdução) a uma aterragem sobre uma rede de polipropileno colocado sobre uma superfície irregular, conforme a seguinte figura (Gonçalves-Coelho, Veloso et al. 2007):

Figura 85 – Rede de aterragem na plataforma móvel a utilizar inicialmente.

A abordagem inicial idealizada foi a seguinte:

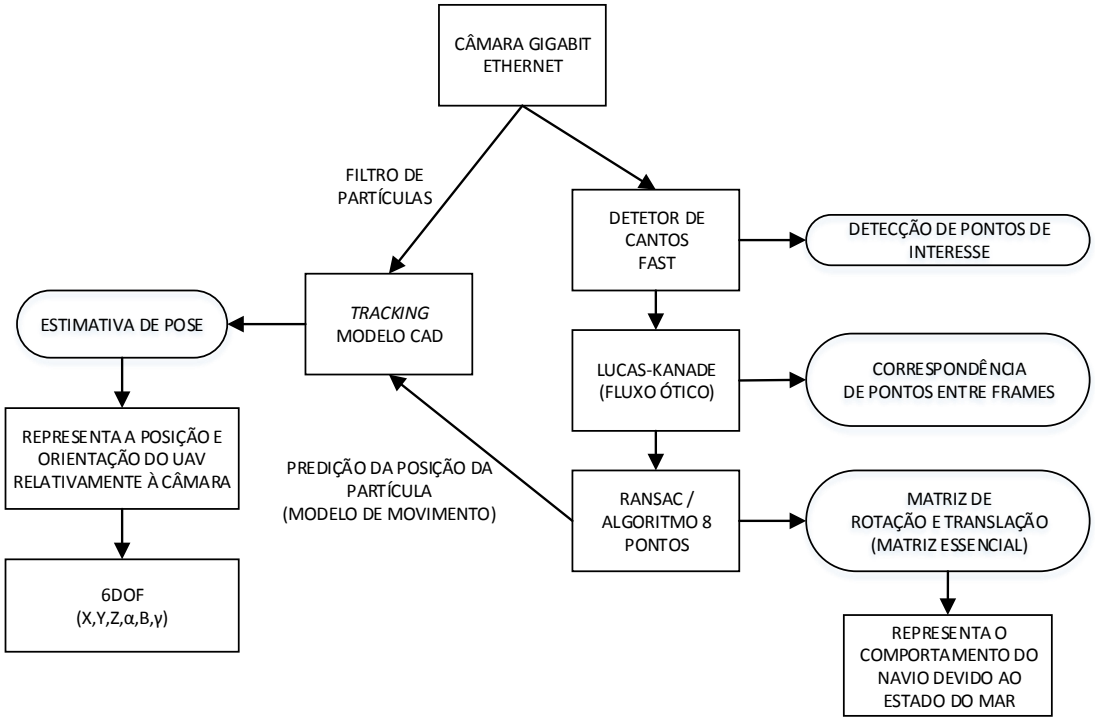


Figura 86 – Esquema Geral Inicial.

A compensação do balanço proveniente das condições de mar adversas, seria feita de acordo com uma arquitetura em que duas *frames* sucessivas (Figura 87) seriam utilizadas para calcular a matriz de rotação e translação (como acontece na visão estereoscópica – Matriz Essencial). Para tal seria utilizada uma arquitetura baseada na deteção dos pontos de interesse (*keypoints*) nas duas figuras recorrendo a um método baseado no algoritmo FAST.

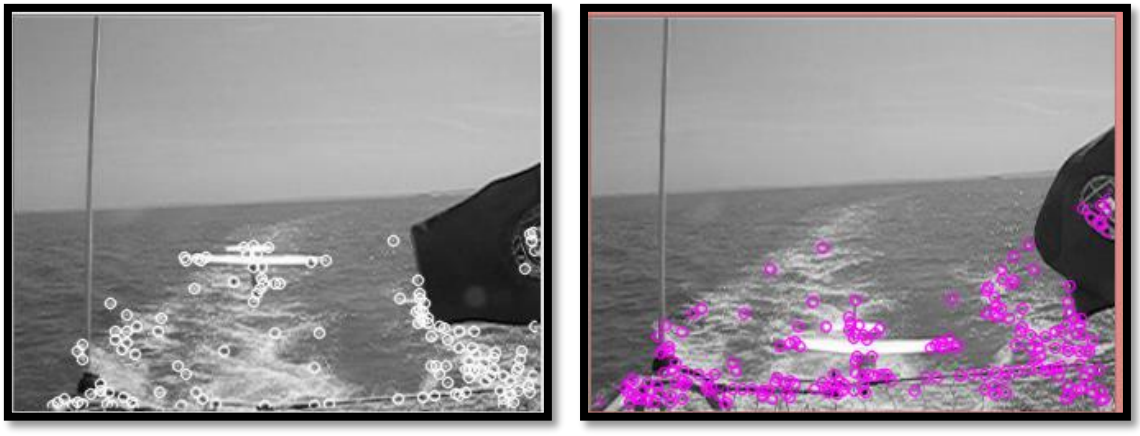


Figura 87 – Detetor de cantos – FAST - $frame = 0$ (esquerda) e $frame = 5$ (direita).

Figura 88 – Fluxo ótico calculado – Representado pelas linhas.

Isto faz com que em cada iteração para uma média de 220 pontos de correspondência se demore aproximadamente 11 ms[41], 620 pontos aproximadamente 19 ms e 2600 pontos aproximadamente 45 ms. Como se vai poder verificar no capítulo seguinte todo o tempo de processamento gasto é precioso nesta implementação, principalmente quando o tempo a acrescentar neste cálculo não é constante e é grandemente influenciado pelo número de pontos utilizados. O tempo de cálculo de cantos/pontos de interesse é desprezável pois é inferior a 1 ms utilizando o algoritmo FAST.

Para resolver este problema vão ser utilizados sensores externos na implementação real nomeadamente GPS[42] e IMU[43], procurando assim tirar o processamento da CPU. Testaram-se variadíssimos sistemas já existentes em laboratório mas as baixas taxas de amostragem do GPS disponível – 1 Hz – e a componente fechada (baseadas em *software* proprietário que permite pouca configuração específica) das IMUs fez com que fosse necessária a aquisição de novos produtos (Figura 89). O GPS – *Venus* GPS – com uma taxa de amostragem até 20 Hz e a *Razor* IMU de 9 graus de liberdade (Figura 89). As taxas de amostragem máximas utilizadas atualmente nos UAVs é de cerca de 50 Hz para a rotação (IMU) utilizada e 5 Hz para a translação (GPS).

[41] Os tempos de processamento apresentados ao longo desta dissertação são referentes a um computador portátil com 2,40 GHz Intel i7 CPU e com uma NVIDIA *GeForce* GT 750M.
[42] *Global Positioning System* permite obter a localização em tempo real através da triangulação da informação proveniente de satélites específicos para o efeito.
[43] *Inertial Measurement Unit* permite obter a velocidade, orientação e força da gravidade. Sendo constituída por uma combinação de giroscópios e acelerómetros.

No entanto nesta dissertação ainda não vai ser contemplada a compensação devido ao balanço da plataforma, foi estudada numa fase inicial para aferir a sua exequibilidade mas rapidamente se percebeu que a utilização de sensores externos seria o mais adequado.

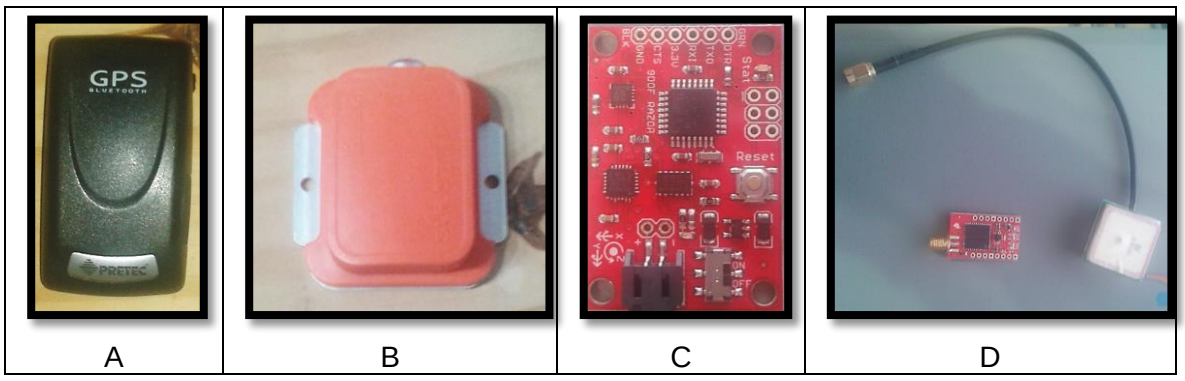


Figura 89 – GPS *Bluetooth* (A), IMU (B), IMU adquirida (C) e GPS adquirido (D).

Quanto ao *tracking* este problema complexo foi decomposto em dois subproblemas distintos (Figura 90): Estimação da pose da aeronave e o seguimento utilizando algoritmos que permitam que a informação obtida na *frame* anterior não seja perdida (*tracking* propriamente dito), obtendo assim resultados mais fiáveis. O principal objetivo é sempre obter a estimativa da pose da aeronave num ambiente exterior, utilizando uma plataforma móvel (navio). As não linearidades do sistema são óbvias, pois os parâmetros do sistema são altamente influenciados e.g. pelo local geográfico, entre outros fatores. O vetor de estado $\boldsymbol{P_t}$ é definido pela posição 3D (X, Y, Z) e a orientação é representada pelos ângulos de Euler segundo X, Y e Z (γ, β, α), sendo o objetivo principal estimar o vetor $\{\boldsymbol{P_t}; \boldsymbol{t} \boldsymbol{\epsilon} \mathbf{N}\}$ em cada instante temporal e enviar esta informação para uma estação de controlo[44].

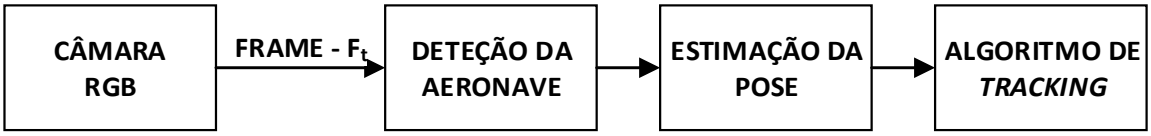


Figura 90 – Esquema Geral da abordagem de estudo utilizada.

O resultado da estimativa desta arquitetura pode ser aumentado não só pelo aumento da fiabilidade nos próprios métodos utilizados, mas também aumentando o número de

[44] Esta estação de controlo vai inserir esta posição numa malha de controlo, proporcionando assim uma aterragem automática do UAV.

sistemas utilizados em paralelo fazendo com que seja possível efetuar uma média dos valores obtidos ou pesar os resultados obtidos por alguns fatores multiplicativos por forma a obter uma estimativa o mais próxima da realidade possível. Sendo importante, e devido ao elevado custo computacional envolvido, que cada sistema tenha capacidade de processamento independente.

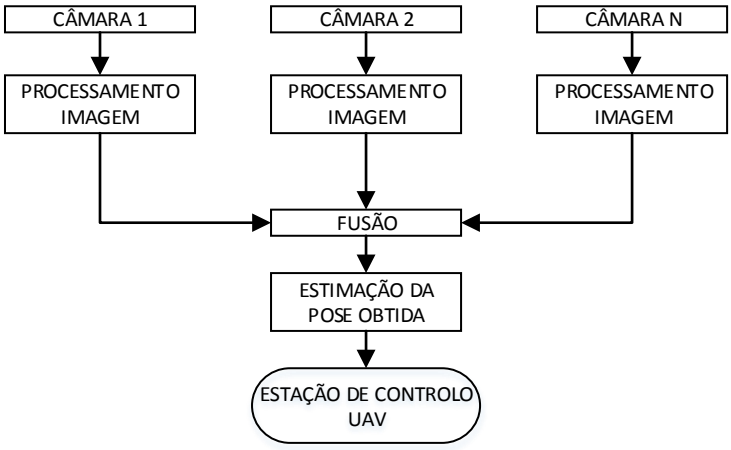


Figura 91 – Exemplo de esquemático para aumento da fiabilidade na estimação de pose.

Os próximos subcapítulos vão descrever a arquitetura de deteção de pose desenvolvida, bem como a respetiva adaptação para *tracking* efetuada. Descrevendo assim toda a implementação e as soluções adotadas.

3.2. DETEÇÃO DE POSE

A deteção de pose é efetuada através de uma arquitetura dividida em quatro fases:

- **Deteção da Aeronave** – Nesta fase existe uma pesquisa na imagem, procurando por zonas onde possam existir veículos aéreos com a aparência do UAV[45] a aterrar;
- **Inicialização das Partículas** – Nesta fase tenta-se efetuar a inicialização do filtro, efetuando a comparação das regiões obtidas no ponto anterior com uma base de dados de *bounding boxes* (BB) em múltiplas poses. Todas as poses com um valor de semelhança elevada vão ser utilizadas para inicialização do filtro;

[45] A terminologia - objeto – vai muitas vezes ser utilizada ao longo deste capítulo e tem como objetivo designar o UAV a detetar.

- **Avaliação das Partículas** – A cada partícula gerada no ponto anterior vai ser atribuído um valor de *likelihood*[46], de acordo com a distância da partícula dois métodos distintos foram utilizados (conforme descrito no subcapítulo 3.2.3);
- **Otimização de Pose** – Baseado no valor obtido através das métricas de *likelihood* utilizadas, as partículas são reamostradas e otimizadas para melhor se ajustar à aparência real do objeto.

Nos próximos subcapítulos, cada ponto da arquitetura utilizada vai ser descrito procurando assim detalhar o seu funcionamento.

3.2.1. DETEÇÃO DA AERONAVE

Inicialmente nesta fase de deteção da aeronave houve a tentativa de utilizar uma seleção manual, em que o utilizador através de um periférico (e.g. rato) selecionava a localização do objeto a detetar e utilizava também a informação da posição para a inicialização das partículas. Como o tamanho do objeto a detetar é conhecido (Figura 92), poderíamos utilizar a *bounding box* para tentar ter uma aproximação inicial da sua localização. Analisando o exemplo presente na Figura 93 (em que as coordenadas em *pixels* dos extremos da *bounding box* encontram-se representadas), é possível obter a seguinte diferença de *pixels* em X:

$$diferença_X \cong 768 - 723 \cong 45\ pixels \tag{3.1}$$

Utilizando o tamanho do objeto 1,27 (Figura 92), temos:

$$l\ (distância\ objeto) \cong \frac{tamanho_{objeto} \times \left(\frac{fx+fy}{2}\right)}{diferença_X} = \frac{1{,}27 \times 1306}{45} \cong 36{,}86 \tag{3.2}$$

Utilizando o tamanho do objeto 1,78 (Figura 92), temos:

$$l\ (distância\ objeto) \cong \frac{1{,}78 \times 1306}{45} \cong 51{,}7 \tag{3.3}$$

Tendo em conta que a matriz de parâmetros intrínsecos é dada por:

$$K = \begin{bmatrix} 1307.372768 & 0.000000 & 618.970838 \\ 000000 & 1305.056687 & 349.740848 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix} \tag{3.4}$$

[46] Neste caso é a designação anglo-saxónica para verossimilhança dada à análise da partícula, nesta dissertação este termo vai ser utilizado em detrimento da expressão verossimilhança ou probabilidade de ser igual ou superior.

Nas equações anteriores foi utilizado um valor de 1306 para a distância focal, correspondendo a uma média entre f_x e f_y ($\frac{f_x+f_y}{2}$) conforme parâmetros obtidos por calibração e descritos em (3.4). Como não sabemos para onde o objeto está virado inicialmente, temos de tentar utilizar um valor intermédio. Utilizando o valor intermédio de 1,53 temos:

$$l\,(distância\ objeto) \cong \frac{1{,}53\times 1306}{45} \cong 44 \tag{3.5}$$

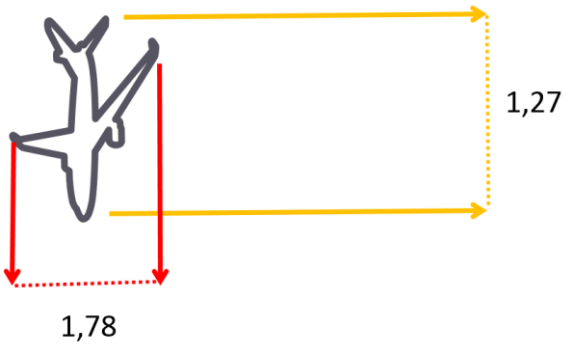


Figura 92 – Dimensões do objeto a detetar (metros).

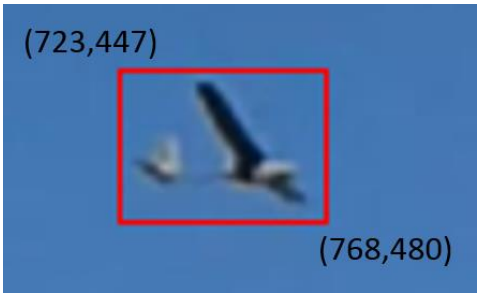


Figura 93 – Exemplo de inicialização manual – imagem real.

Para verificar se o erro era proveniente do operador ao mesmo exemplo anterior foi aplicado um algoritmo FAST e obtida a *bounding box* que inclui os pontos obtidos *(*Figura 94).

Figura 94 – Algoritmo FAST e respetiva *bounding box* dos pontos obtidos.

Obtemos então o seguinte:

$$dif_X \cong 744 - 709 \cong 35\ pixels \tag{3.6}$$

Utilizando o tamanho do objeto 1,27 (Figura 92), temos:

$$l\ (distância\ objeto) \cong \frac{1{,}27 \times 1306}{35} \cong 47 \tag{3.7}$$

Utilizando o tamanho do objeto 1,78 (Figura 92), temos:

$$l\ (distância\ objeto) \cong \frac{1{,}78 \times 1306}{35} \cong 66 \tag{3.8}$$

Como não sabemos para onde o objeto está virado inicialmente, temos de tentar utilizar um valor intermédio. Utilizando o valor intermédio de 1,53 temos:

$$l\ (distância\ objeto) \cong \frac{1{,}53 \times 1306}{35} \cong 57 \tag{3.9}$$

O que faz com que a diferença obtida entre considerar 1,27 e 1,78 muito grande nos dois casos, sendo a área de pesquisa também muito grande o que não facilita a inicialização. A distância em Z correta deveria ser de aproximadamente 25,3, o que está algo longe do que se obtém nos dois casos anteriores. Adicionalmente, o erro introduzido pelo operador na seleção da *bounding box* pode influenciar grandemente o erro obtido na primeira abordagem. Outros problemas, tais como a inclusão de outros objetos na imagem, dificultam a determinação dos pontos que realmente pertencem ao objeto.

Para resolver este problema estudaram-se algumas implementações específicas que permitissem detetar o objeto e respetiva região de interesse com o mínimo de erro possível. Foram testadas variadíssimas soluções das quais se destacam duas que não tiveram o sucesso desejado:

- *Template Matching* (Lepetit and Fua 2005, Dhindsa and Babbar 2011, Guo and Osher 2011, Mahalakshmi, Muthaiah et al. 2012) – Obtenho um valor de 0 a 1 consoante a aproximação do objeto na imagem à imagem de referência. Na implementação prática efetuada verificou-se que poses distintas obtinham valores de semelhança iguais, além de mostrar uma grande sensibilidade a mudança de cor. O que em ambientes exteriores em que a iluminação está sempre a mudar, se demonstra inútil;
- *Perspective-n-point* (PnP) (Haralick, Lee et al. 1994, Lepetit and Fua 2005) – Foram selecionados os 4 pontos extremos do modelo 3D CAD, detetados os pontos de interesse na imagem (FAST) e desenhada a respetiva *bounding box* (com base nos *keypoints*). Foram utilizados esses pontos para o algoritmo PnP, mas como o objeto

não é um objeto simples (como e.g. um cubo) os resultados obtidos foram bastante dispares da realidade.

A deteção da região de interesse inicial é muito importante (*Region of interest* – ROI), uma vez que vão ser utilizados na próxima fase do filtro os pontos fundamentais da imagem (*keypoints*) para efetuar a inicialização das partículas a utilizar. Uma vez que estamos a operar em ambiente exterior é muito importante obter (o mais possível) invariância da arquitetura a variações de luminosidade, garantindo uma taxa de deteção do objeto elevada.

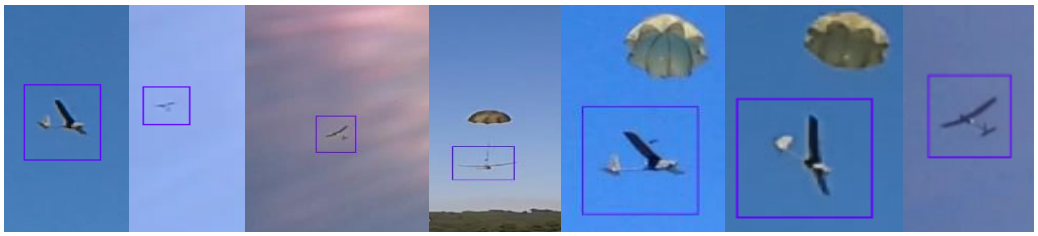


Figura 95 – Exemplos de utilização do classificador para diversas imagens reais – LBP. Com diferentes condições de luminosidade a região de interesse (Retângulo violeta) correspondente ao UAV é corretamente detetada.

A posição inicial do UAV é detetada utilizando um classificador - *Local binary pattern cascade classifier* (Viola and Jones 2001, Pietikäinen, Hadid et al. 2011), sendo esta deteção inicial muito importante uma vez que estamos a operar no exterior e a presença de outros objetos na imagem (e.g. um pássaro, nuvens, etc.) afeta o desempenho da arquitetura proposta. Foram testados também as características (*features*) *Haar Feature-based* (Lienhart and Maydt 2002) e *Histogram of oriented gradients* (Dalal and Triggs 2005) – descritas no subcapítulo 2.4.2 e 2.4.3 respetivamente.

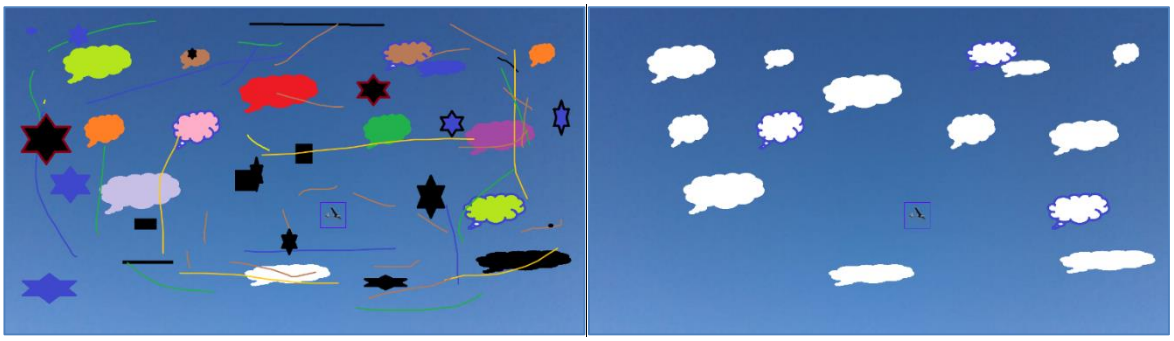
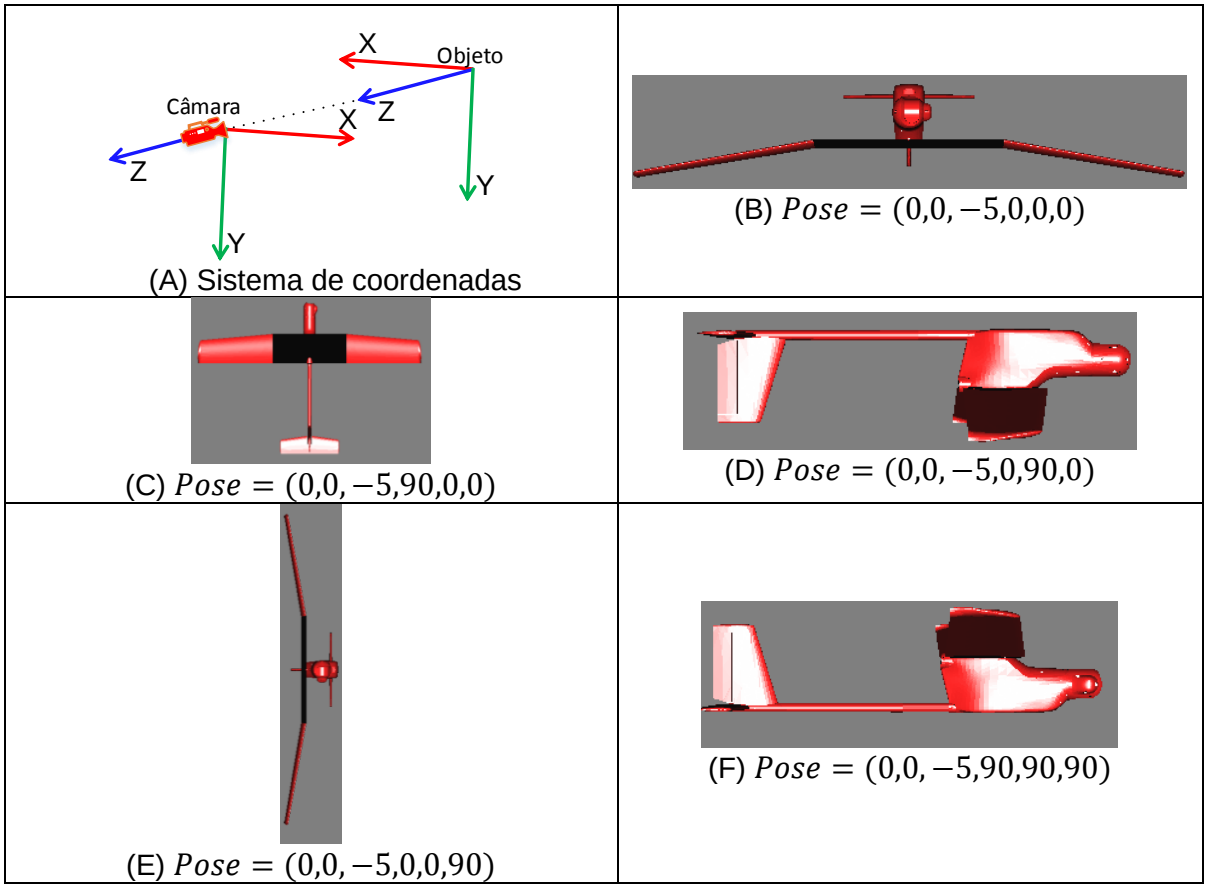


Figura 96 – Exemplo de utilização do classificador – LBP – com adição de Ruído (outros objetos). Apesar da adição de ruído (figuras e linhas) a região de interesse (Retângulo violeta) correspondente ao UAV é corretamente detetada.

3.2.2. INICIALIZAÇÃO DAS PARTÍCULAS

Da fase anterior resulta uma região de interesse que permite obter uma *bounding box* (BB) orientada (Figura 95 e Figura 96) que indica a posição e a atitude de forma grosseira do UAV. Para inicializar as partículas com uma atitude próxima da atitude real do UAV, são comparadas as características da BB detetada com uma base de dados de BB geradas sinteticamente em múltiplas poses. Esta base de dados (Tabela 6) foi criada *offline* e inicializada para memória no início do programa por forma a obter um tempo de acesso quase nulo. Esta base de dados é constituída pelos ângulos de Euler α,β e γ (rotação segundo o eixo dos X,Y e Z respetivamente), o ângulo da BB em relação à horizontal, a largura e altura da BB e as coordenadas do seu centro. Como o modelo da câmara utilizado é o modelo de perspetiva, a base de dados (nível de treino) pode ser criada independentemente da posição do objeto na imagem. É no entanto importante garantir que o centro de rotação se encontra no centro do objeto (Figura 98), senão o erro da projeção com o obtido pela análise da imagem obtida é bastante elevado impedindo uma estimativa correta conforme testes práticos efetuados.



(A) Sistema de coordenadas

(B) $Pose = (0,0,-5,0,0,0)$

(C) $Pose = (0,0,-5,90,0,0)$

(D) $Pose = (0,0,-5,0,90,0)$

(E) $Pose = (0,0,-5,0,0,90)$

(F) $Pose = (0,0,-5,90,90,90)$

Figura 97 – Exemplos de $Pose = (X,Y,Z,\alpha,\beta,\gamma)$ – (B), (C), (D), (E) E (F) – respetivos sistemas de eixos – objeto e câmara - (A).

A Figura 97 apresenta a relação entre o sistema de eixos da câmara (OpenGL) e do objeto (modelo CAD), em que é possível verificar que o eixo dos Z e Y se encontra alinhado e o dos X apresenta sinal oposto. As rotações têm o seu sentido positivo de acordo com a regra da "mão direita" (Figura 17), sendo feitas pela ordem α, β e γ respetivamente – Exemplos na Figura 97 (B) a (F).

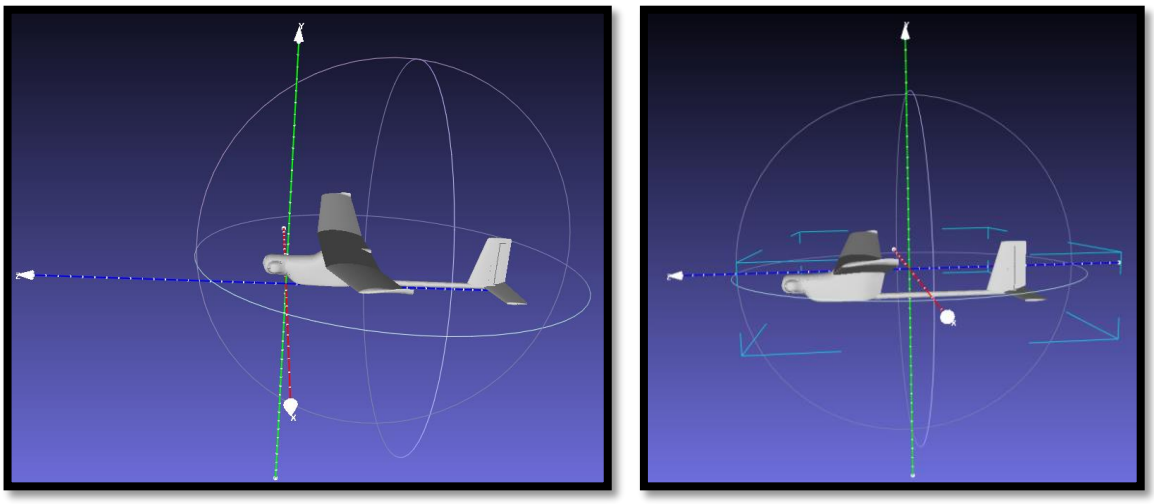


Figura 98 – Modelo do UAV utilizado com eixo de rotação frente (esquerda) e centro (direita).

As BB detetadas são obtidas aplicando o algoritmo FAST (Rosten and Drummond 2006, Rosten, Porter et al. 2010) para detetar os pontos-chave (*keypoints*) da imagem observada, selecionando assim os pontos que estão dentro da região de interesse obtida. Esta região de interesse é obtida simplesmente na fase de deteção da aeronave, tornando assim possível selecionar apenas os pontos que pertencem ao UAV.

Alfa α	Beta β	Gama γ	Ângulo BB Θ	Altura BB	Largura BB	Aspect Ratio BB	BB Centro C_x	BB Centro C_y

Tabela 6 – Parâmetros guardados na base de dados criados.

A base de dados é gerada projetando o modelo 3D CAD do UAV numa posição fixa (Figura 99), fazendo variar a sua rotação de acordo com uma distribuição Gaussiana - $\mathcal{N}(0,90)$ para os parâmetros γ, β e α. A base de dados criada representa 20000 orientações possíveis com: 13620 possibilidades (68,1 %) para a semiesfera frontal (ângulo de *pitch* entre $-90°$ e $90°$) e o resto (6380) estão contidos na semiesfera de trás. O UAV tem um tamanho de 1,27 m, o que faz com que o raio da esfera criada seja de 0,63 m. A sua rotação cria uma esfera de superfície:

$$A = 4\pi r^2 = 4 \times \pi \times 0{,}63^2 \cong 4{,}99\, m^2 \cong 5\, m^2 \quad (3.10)$$

Se dividirmos a área obtida em dois obtemos a área de cada semiesfera$\cong 2{,}5\, m^2$. Dividindo a área pelo número de possibilidades obtemos:

$$\delta Area_{Frente} = \frac{2{,}5}{13620} \cong 1{,}84 \times 10^{-4} m^2 \quad (3.11)$$

$$\delta Area_{trás} = \frac{2{,}5}{6380} \cong 3{,}92 \times 10^{-4}\, m^2 \quad (3.12)$$

O ângulo sólido é dado pelo seguinte:

$$\Omega_{\text{frente}} = \frac{\delta A}{r^2} = \frac{1{,}84 \times 10^{-4}}{0{,}63^2} \cong 4{,}64 \times 10^{-4} sr \quad (3.13)$$

$$\Omega_{\text{trás}} = \frac{\delta A}{r^2} = \frac{3{,}92 \times 10^{-4}}{0{,}63^2} \cong 9{,}88 \times 10^{-4} sr \quad (3.14)$$

Convertendo esferorradiano (sr) em graus:

$$\theta_{frente} = \sqrt{\Omega_{\text{frente}}} \times \frac{180}{\pi} \cong \sqrt{4{,}64 \times 10^{-4}} \times \frac{180}{\pi} \cong 1{,}23\ graus \quad (3.15)$$

$$\theta_{trás} = \sqrt{\Omega_{\text{trás}}} \times \frac{180}{\pi} \cong \sqrt{9{,}88 \times 10^{-4}} \times \frac{180}{\pi} \cong 1{,}8\ graus \quad (3.16)$$

Obtém-se assim uma diferença média entre posições na semiesfera frontal de 1,23º e de 1.8º para as possibilidades na semiesfera de trás. A semiesfera frontal foi favorecida pois numa situação de aterragem a pose do UAV estará localizada na semiesfera positiva.

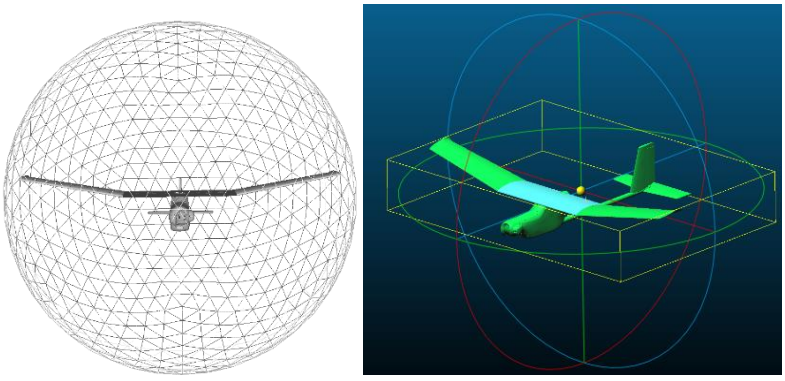


Figura 99 – Representação da esfera 3D de orientações possíveis (esquerda) eixo de rotação (direita).

Para cada posição gerada é obtida a BB que melhor se encaixa no objeto projetado e esta é guardada numa base de dados (Gráfico 1) juntamente com o seu ângulo em relação à horizontal - θ – e o seu *Aspect ratio*[47] – AR.

[47] Relação entre largura e altura da *bounding box* obtida.

$$AR = Aspect\ Ratio\ Bounding\ Box = \frac{largura}{altura} \tag{3.17}$$

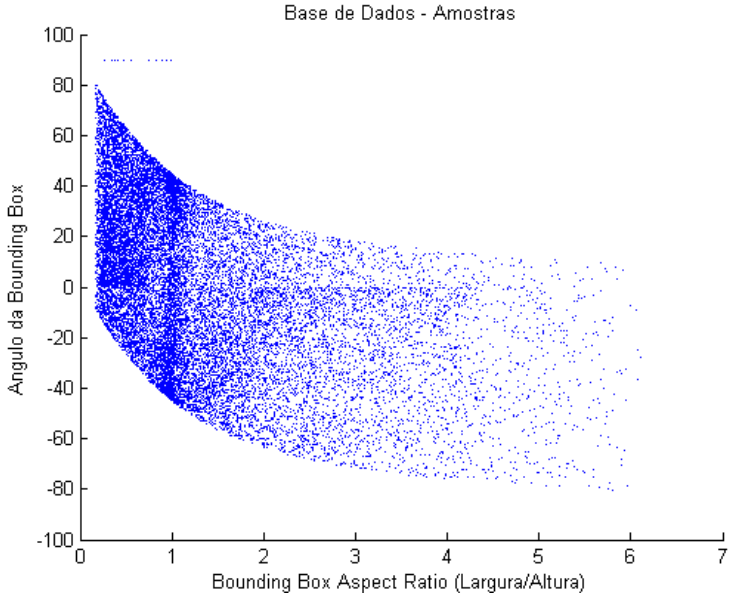


Gráfico 1 - Distribuição das 20000 amostras guardadas na base de dados – $BB_{Angulo} = f(BB_{Aspect\ Ratio})$.

A diferença entre a BB observada (obs) e a da base de dados (dados) é calculada *online* usando a distância Euclidiana (também foram testadas outras distâncias e.g. *Manhattan* mas esta foi a que obteve melhores resultados) (Jähne, Haussecker et al. 1999, Hornberg 2007, Bradski and Kaehler 2008, Nixon 2008, Davies 2012):

$$d(\theta, AR) = \sqrt{(\theta_{obs} - \theta_{dados})^2) + (AR_{obs} - AR_{dados})^2} \tag{3.18}$$

Para as melhores possibilidades é assumido que todos os pontos do objeto se encontram à mesma profundidade (coordenada Z), projetados num plano paralelo à imagem. A coordenada Z pode ser calculada pela relação entre a área das *bounding boxes* e a sua profundidade da seguinte forma:

$$Z = Z_{Base\ de\ Dados} \sqrt{\frac{Area_{Base\ de\ dados}}{Area_{Observação}}} \tag{3.19}$$

A coordenada X e Y é calculada pela relação entre as coordenadas do centro da BB observada, a coordenada Z obtida, e os parâmetros intrínsecos obtidos através da calibração da câmara – distância focal (f_x e f_y) e coordenadas do centro da câmara (C_X e C_y).

$$X = \frac{Z(BB_X - C_X)}{f_x} \tag{3.20}$$

$$Y = \frac{Z(BB_Y - C_Y)}{f_y} \tag{3.21}$$

É importante garantir que a calibração da câmara é feita de forma correta para assegurar precisão no desempenho do sistema. A Figura 100 mostra o diagrama de blocos que resume o procedimento de inicialização utilizado.

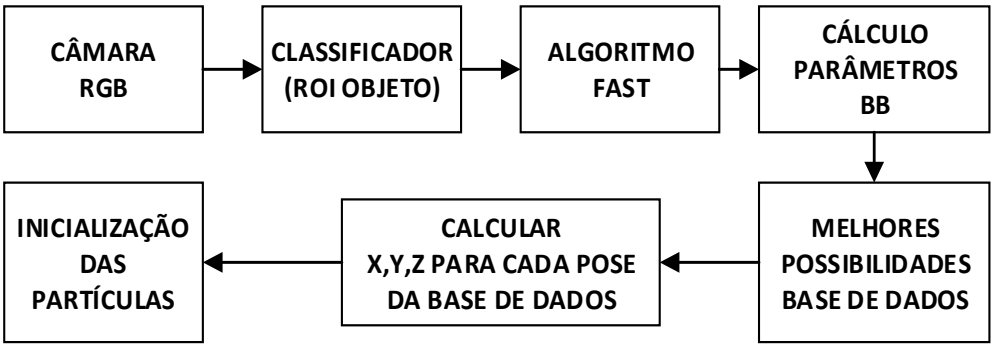


Figura 100 – Arquitetura de inicialização das partículas utilizada.

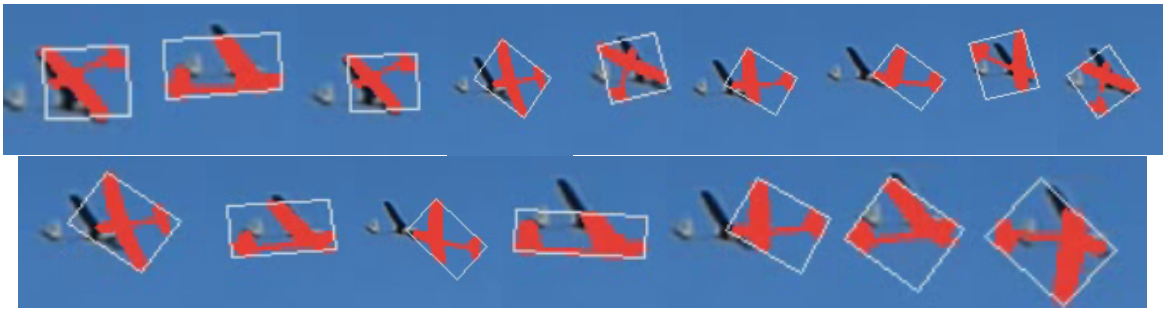


Figura 101 – Exemplo de 16 possibilidades de inicialização obtidas – Base de dados.

3.2.3. AVALIAÇÃO DAS PARTÍCULAS

Cada partícula vai corresponder a uma pose 3D do objeto (posição e atitude). Ao projetar o modelo 3D CAD de uma partícula na imagem, obtemos uma localização espectável da sua localização (hipótese). Para avaliar a qualidade da hipótese, vamos amostrar a *frame* atual na localização expectável e avaliar com uma função de *likelihood*. Para diminuir o erro de estimativa existente, duas funções de *likelihood* distintas foram utilizadas (de acordo com os resultados práticos expostos no próximo capítulo, esta foi a abordagem que apresentou melhores resultados):

- Diferença entre o histograma interior (objeto) e o histograma exterior (Figura 102) limitado por uma bounding box (distância do UAV superior ou igual a 25 metros);

- Abordagem híbrida combinando a função de *likelihood* descrita no ponto anterior, com uma função de *likelihood* baseada em contornos (distância do UAV inferior a 25 metros).

Acima de 25 metros a partícula com o peso mais elevado é a partícula que maximiza a diferença entre os dois histogramas (Chang and Krumm 1999, Cha and Srihari 2002, Gevers and Stokman 2004, Bradski and Kaehler 2008) (Figura 102, Figura 103 e Figura 104). Utilizando esta abordagem é possível ter uma função de *likelihood* invariante a mudanças de iluminação. Esta robustez é muito importante tendo em conta que esta é uma aplicação exterior, onde as variações de brilho são imprevisíveis.

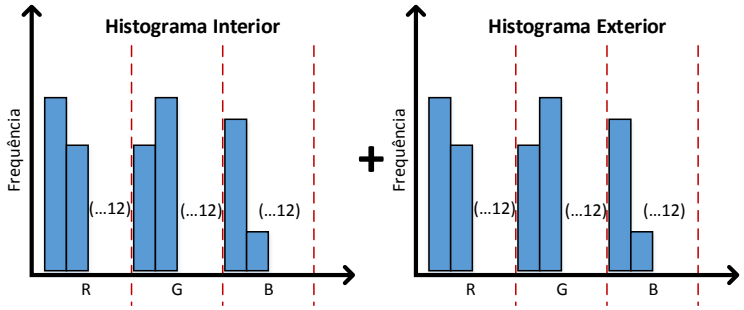


Figura 102 – Ilustração do Histograma interior e exterior.

Os histogramas são obtidos no espaço de cor RGB[48] (12 *bins* para cada cor – Γ= 36), e a distância entre eles é calculada utilizando a métrica de semelhança de Bhattacharyya da seguinte forma (Bradski and Kaehler 2008, Taiana, Nascimento et al. 2008, Dubuisson 2010, Taiana, Santos et al. 2010):

$$L_{textura} = 1 - \sum_{b=1}^{\Gamma} \sqrt{h^{interior}(b).h^{exterior}(b)} \qquad (3.22)$$

onde $h^{interior}$ é o histograma interior, $h^{exterior}$ é o histograma exterior e b o respetivo *bin* do histograma.

Figura 103 – Exemplo da região interior (objeto) e exterior (entre o objeto e a bounding box) onde os histogramas são calculados para a métrica de *likelihood*.

[48] Foram testados outros espaços de cor (e.g. HSV) tendo este espaço de cor apresentado os melhores resultados.

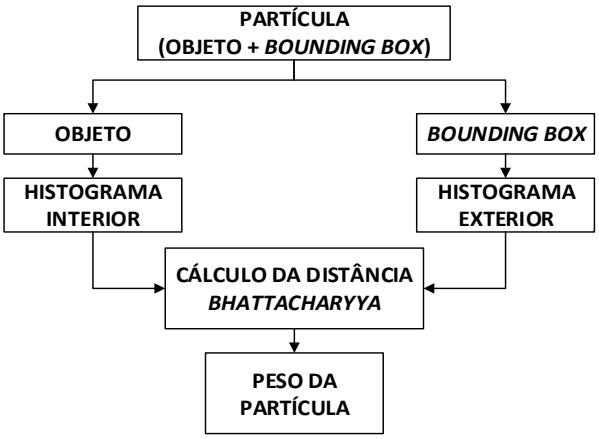


Figura 104 – Esquema simplificado da arquitetura utilizada para o cálculo do peso das partículas.

A função de *likelihood* híbrida (utilizada quando a partícula se situa a distâncias inferiores a 25 metros) combina a função de *likelihood* descrita anteriormente, com a informação de contornos. Os contornos visíveis (modelo CAD 3D) são projetados no plano 2D de acordo com a pose a testar (Figura 105), sendo que os segmentos de reta que compõem o contorno são identificados.

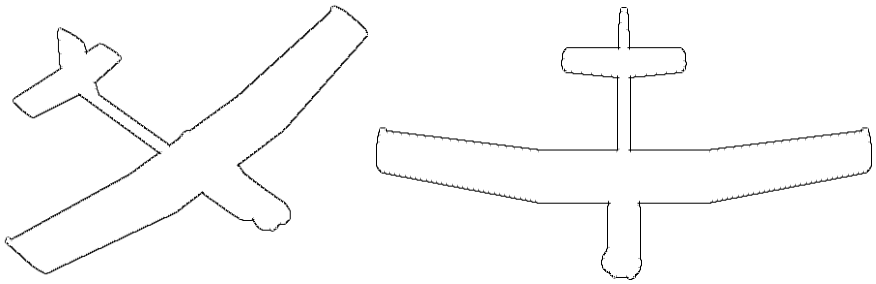


Figura 105 – Exemplos da representação dos contornos do objeto.

Para cada segmento de reta é amostrado o ponto central (v), sendo efetuada uma pesquisa 1D perpendicular ao segmento de reta identificado (Figura 107). Esta pesquisa pretende fazer corresponder o ponto amostrado com o contorno mais próximo (m), depois de calcular as correspondências existentes a função de *likelihood* baseada em contornos (Figura 106) é calculada da seguinte forma (Choi and Christensen 2011):

$$L_{contornos} = exp^{\left(-\lambda_v \frac{(p_v - p_m)}{p_v}\right)} \times exp^{(-\lambda_e \overline{e})} \tag{3.23}$$

onde $\overline{e}$ é a distância média entre os pontos amostrados e os pontos do contorno, λ_v e λ_e são fatores de sensibilidade utilizados para ajustar a função de *likelihood* de contornos; p_v é o número de pontos amostrados visíveis e p_m é o número de pontos de amostragem em que foi efetuada a correspondência.

Para aumentar o desempenho do sistema quando uma imagem é obtida a magnitude e orientação do gradiente (Filtro de *Sobel*) é calculado e guardado.

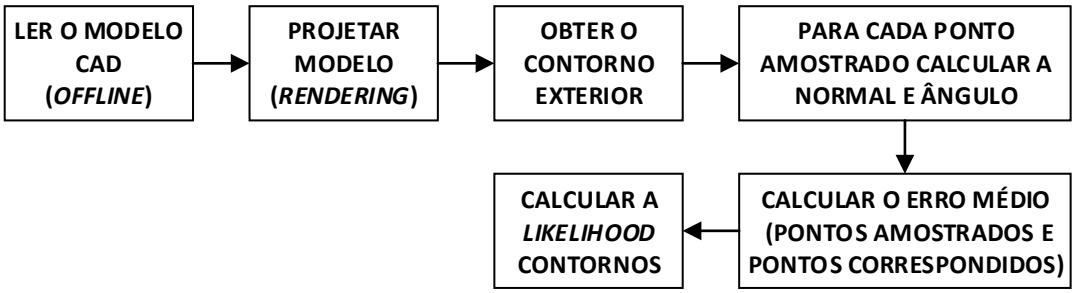


Figura 106 – Esquema simplificado – *Likelihood* Contornos.

Para obter menos erro no processo de correspondência, para cada ponto amostrado o seu ângulo é calculado e comparado com o gradiente obtido no ponto em que foi efetuada correspondência. Se o ângulo divergir mais de 45 graus este ponto é excluído, por forma a minimizar o erro existente.

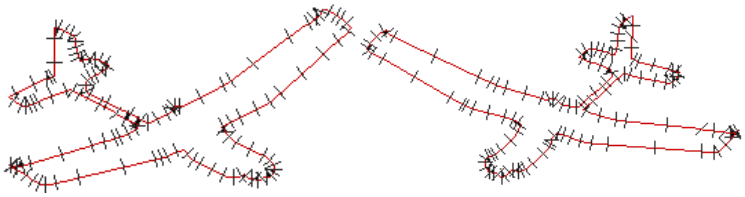


Figura 107 – Pontos amostrados e pesquisa 1D (linhas pretas).

A função de *likelihood* hibrida pode ser criada então pela junção das duas funções de *likelihood* descritas anteriormente pelas equações (3.22) e (3.23), e é calculada da seguinte forma:

$$L_{total} = L_{contorno} + A \times L_{textura} \tag{3.24}$$

em que A é um termo de sensibilidade relativa para ajuste fino da função de *likelihood* híbrida descrita. Um esquema simplificado deste cálculo pode ser visto na Figura 108.

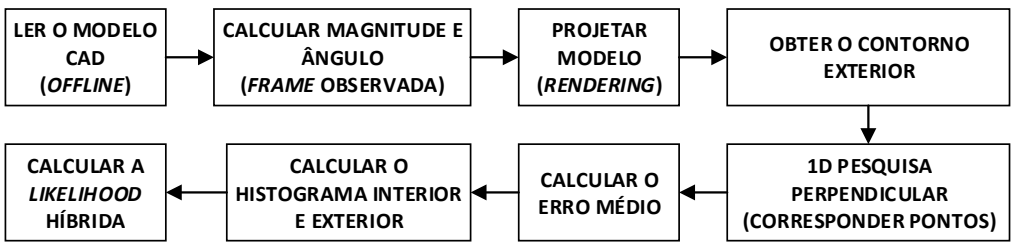


Figura 108 – Esquema simplificado do cálculo da *likelihood* Híbrida.

3.2.4. OTIMIZAÇÃO DA POSE

Diversas arquiteturas foram abordadas nesta fase, nomeadamente o filtro de partículas Gaussiano que é implementado em algumas aplicações de visão artificial. Dado um conjunto de partículas e a função de *likelihood*, o processo de otimização da pose (Figura 109) que demonstrou um melhor desempenho foi o baseado nas seguintes três fases:

- Inicialização (*Bootstrap*);
- Otimização grosseira (*Coarse Optimization*);
- Otimização Fina (*Fine Optimization*).

Durante a fase de inicialização as 100 melhores possibilidades obtidas por comparação com a base de dados são escolhidas e colocadas numa lista (a que se vai chamar – Top 100). A *likelihood* de cada partícula é avaliada e guardada na lista, sendo que as M melhores partículas são guardadas num *buffer* (região de armazenamento temporário) auxiliar (a que vamos chamar – Top M). A Figura 110 ilustra a estrutura dos dados usados nesta fase. As partículas com um peso muito próximo de zero (inferior a $\delta = 0.01$) são eliminadas e substituídas por partículas aleatórias selecionadas do buffer Top M, adicionando algum ruído gaussiano de covariância Σ_B. Neste momento, todas as partículas têm uma *likelihood* superior a δ.

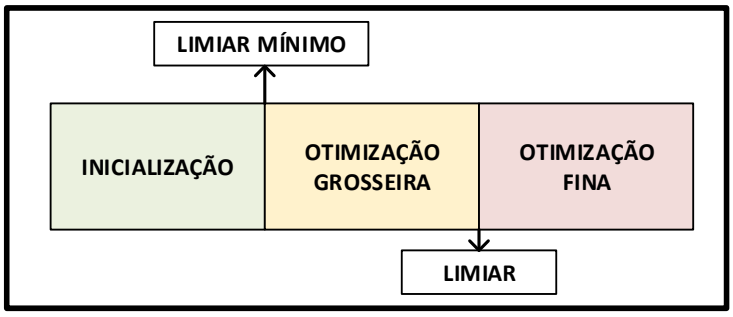


Figura 109 – Fases de otimização do filtro de partículas.

Depois, são efetuadas até 10 iterações para melhoria dos resultados. Em cada iteração todas as partículas são avaliadas e comparadas com as presentes no Top M, se o peso obtido for superior o Top M é atualizado. Se existirem pelo menos duas partículas no Top M com uma *likelihood* superior ao – limiar mínimo – a fase de inicialização termina. Caso contrário, cada partícula é perturbada com ruído gaussiano de covariância $\Sigma_{B.}$ Se após 10 iterações para melhoria dos resultados não existirem duas partículas acima do limiar mínimo, a fase de inicialização é reinicializada até um limite de 3 vezes. Nas experiências efetuadas a ocorrência de reinicializações demonstrou ser bastante rara.

A fase de otimização grosseira começa quando pelo menos duas partículas têm um peso superior ao limiar mínimo, sendo que em qualquer fase (otimização grosseira e fina) as duas partículas de valor mais elevado presentes no Top M desempenham um papel importante pois vão ser os cromossomas para a abordagem baseada nos algoritmos genéticos (Kwok, Fang et al. 2005, Park, Hwang et al. 2009) . Após variados testes práticos esta abordagem demonstrou ser mais efetiva do que usar perturbações aleatórias e reamostrar como nos filtros de partículas tradicionais. A abordagem funciona como se segue.

Cada partícula presente na lista do Top 100 proveniente da fase de inicialização é analisada Se a partícula é a melhor é perturbada com ruído gaussiano, se por outro lado o peso da partícula for inferior as 2 melhores partículas são combinadas por cruzamento[49] para criar uma nova partícula. A operação de cruzamento consiste na seleção aleatória de atributos (X,Y,Z,γ,β,α) das partículas originais. Para metade das partículas geradas por cruzamento é ainda aplicada uma mutação suave adicionando ruído gaussiano ao resultado obtido. Estas regras juntas permitem obter uma diversidade nas partículas geradas focando-as nas melhores possibilidades testadas, simultaneamente convergindo para a melhor solução e evitando um mínimo local. O processo termina quando existem pelo menos duas partículas acima do valor de limiar. Se tal não acontecer em 10 iterações, o filtro retorna para a fase de inicialização automaticamente.

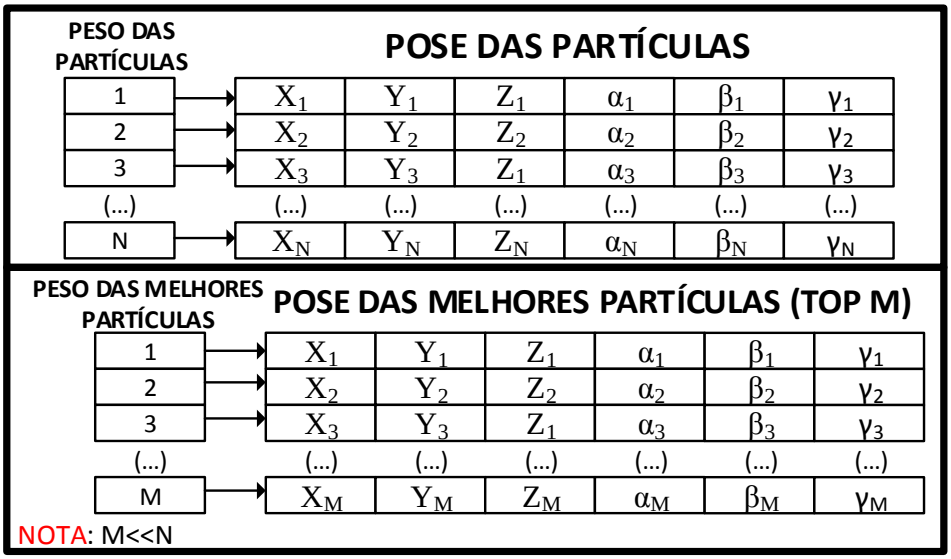


Figura 110 – Pesos das partículas e das melhores partículas (M = 3).

[49] Designado em linguagem anglo-saxónica por – *crossover* - é o que efetivamente acontece ao escolher aleatoriamente os atributos de duas partículas. Isto permite manter a diversidade enquanto nos aproximamos da solução.

A fase de otimização fina é análoga à fase de otimização grosseira mas a variância do ruído gaussiano aplicado no processo de mutação é inferior, permitindo assim fazer um ajuste suave à pose obtida. A otimização fina termina após 5 iterações.

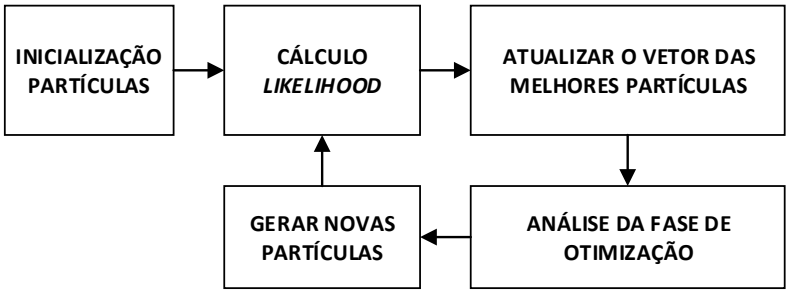


Figura 111 – Arquitetura da fase de otimização de pose.

3.2.5. RESUMO GERAL – DETEÇÃO DE POSE

A arquitetura descrita pormenorizadamente nos subcapítulos anteriores vai ser resumida de forma sucinta neste subcapítulo. Em *offline* são carregados para a RAM[50] os dados descritos na seguinte figura:

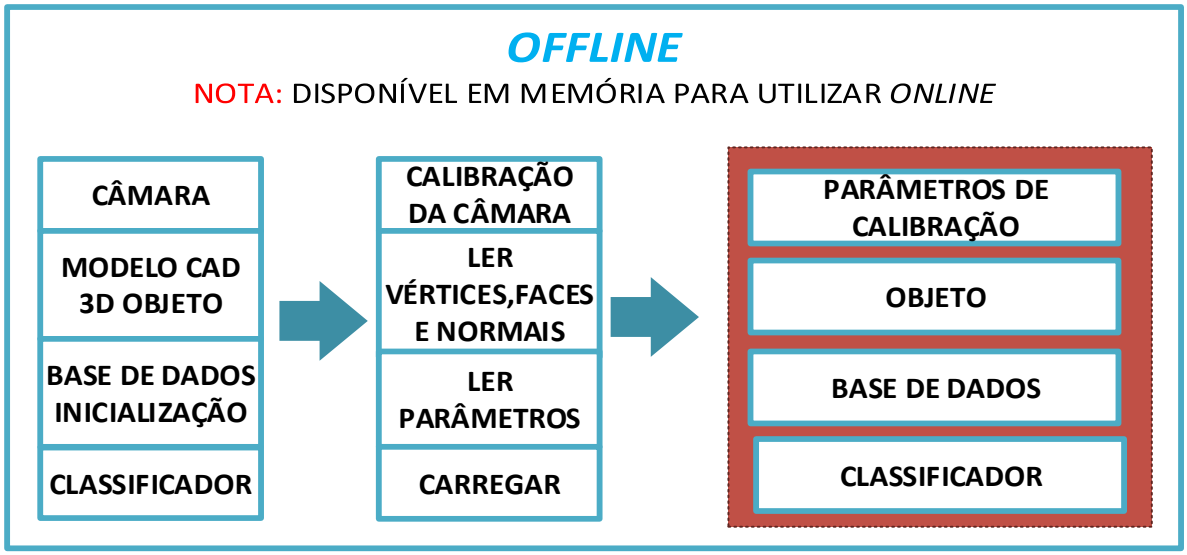


Figura 112 – Arquitetura Geral de deteção de pose – Funcionamento o*ffline.*

[50] *Random Access Memory* ou memória de acesso aleatório. Esta memória permite a leitura e a escrita, sendo a memória primária utilizada pela CPU.

Podemos resumir o conteúdo descrito nos subcapítulos anteriores da seguinte forma:

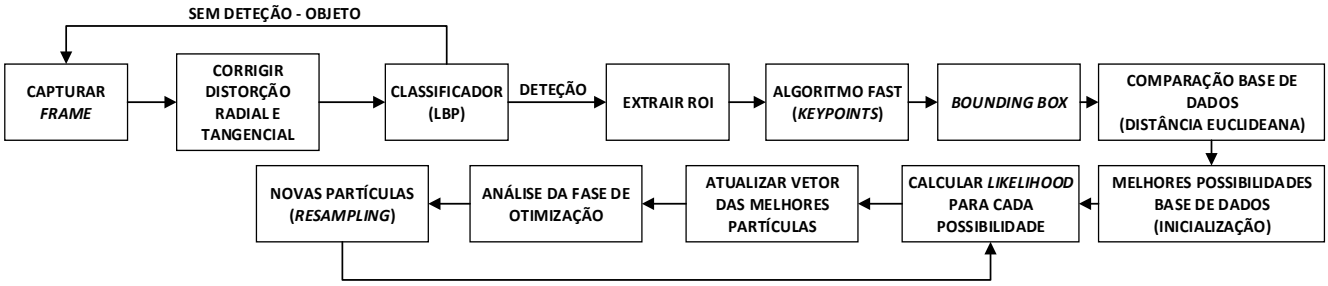


Figura 113 – Arquitetura Geral de deteção de pose – Funcionamento *online*.

O algoritmo é descrito de forma simplificada ao longo do apêndice D – subcapítulo D.1. – Deteção de pose.

3.3. ARQUITETURA DE *TRACKING*

O objeto a detetar tem uma velocidade de cruzeiro de 58 km/h (≅ 16 m/s), se tivermos uma câmara que capture 30 fps o objeto entre *frames* desloca-se:

$$d_{frame\_Cruzeiro} = \frac{1}{30} \times 16 \cong 0{,}53\ m \tag{3.25}$$

Tendo em conta os testes práticos efetuados com um aeromodelo telecomandado COTS (Gráfico 2) em 2005 (Gonçalves-Coelho, Veloso et al. 2007), as velocidades de aterragem foram as seguintes:

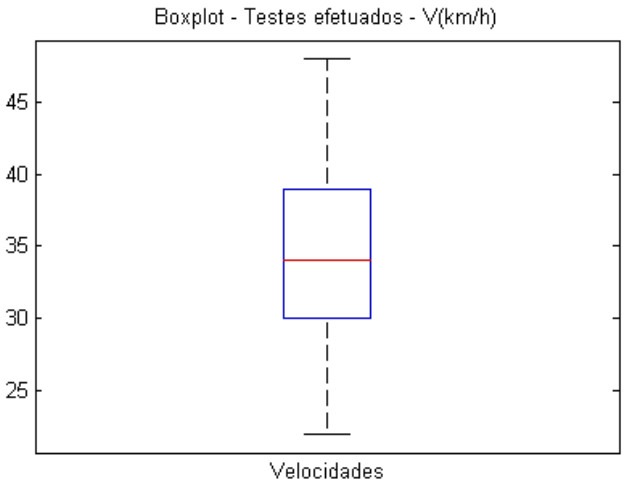


Gráfico 2 – *Box Plot* dos testes efetuados (dados de 26 aterragens) com sucesso do aeromodelo COTS.

	Média	Mediana	Quartil Superior	Quartil inferior	*Whisker* Superior	*Whisker* Inferior
km/h	34	34,42	39	30	48	22
m/s	9,45	9,56	10,84	8,34	13,34	6,12

Tabela 7 – Análise do *Box Plot* criado – Velocidades testes aeromodelo COTS.

A grande discrepância de valores deve-se à natureza telecomandada da aeronave, não existindo um controlo preciso da sua velocidade por parte do operador. Admitindo porém que a velocidade necessária para aterragem (esta velocidade é relativa, tendo já em consideração o movimento da plataforma móvel) é de 34 km/h (9,45 m/s) para uma câmara de 30 fps temos o seguinte deslocamento entre *frames*:

$$d_{frame} = \frac{1}{30} \times 9{,}45 \cong 0{,}32\ m \cong 31{,}5\ cm \tag{3.26}$$

É importante ter em conta que em cada *frame* captada vai existir um atraso devido aos algoritmos de processamento de imagem aplicados, o que faz com que o deslocamento do objeto entre *frames* seja ainda maior do que o obtido em (3.26). Tornando-se assim essencial garantir uma arquitetura a mais robusta e fiável possível por forma a conseguir compensar os deslocamentos consideráveis do objeto, devido à especificidade da aplicação.

A atual arquitetura, e como descrito no subcapítulo 4.3.3, utilizando 100 partículas permite processar aproximadamente 1,3 fps o que é claramente insuficiente ao analisarmos o resultado obtido em (3.26). Iriamos ter um deslocamento entre imagens de:

$$d_{frame} = \frac{1}{1{,}4} \times 9{,}45 \cong 6{,}75\ m \tag{3.27}$$

Ao usarmos 50 partículas tínhamos:

$$d_{frame} = \frac{1}{2{,}6} \times 9{,}45 \cong 3{,}64\ m \tag{3.28}$$

Esta baixa velocidade de processamento por imagem prende-se maioritariamente pela arquitetura definida, em que a possibilidade a testar é gerada na GPU – OpenGL – e transmitida para a CPU e posteriormente processada. Como descrito no subcapítulo 2.7 a transferência entre a GPU e a CPU é lenta devido à largura de banda existente, sendo aconselhado para obter maiores performances que a possibilidade seja gerada e processada na GPU utilizando a arquitetura CUDA (Subcapítulo 4.2).

A metodologia de *tracking* adotada foi basicamente a adaptação da arquitetura de deteção de pose explorada ao longo do subcapítulo 3.2 para contemplar informação temporal sobre o movimento do objeto por forma a fazer *tracking* entre *frames*.

Nos próximos subcapítulos, cada ponto da arquitetura de *tracking* desenvolvida vai ser descrito procurando assim detalhar a sua implementação e funcionamento.

3.3.1. ADAPTAÇÃO DA DETEÇÃO DE POSE

A ideia de utilizar uma arquitetura que não utilizasse apenas a informação proveniente da *frame* actual mas que também usasse alguma informação da *frame* anterior, levou a que a arquitetura de deteção de pose tivesse de ser adaptada por forma a utilizar também essa informação (Okuma, Taleghani et al. 2004). Sendo este um dos objetivos, seguir a pose do UAV (posição 3D e atitude) entre *frames* e em tempo real.

Ao implementar o filtro de Kalman *unscented* – UKF - procurou-se assim fazer uma filtragem da melhor partícula obtida em cada iteração, considerando velocidade constante (linear e angular) entre *frames* consecutivas. Vai-se então combinar um filtro de partículas com um filtro de Kalman *unscented*, conforme vai ser descrito no subcapítulo 3.3.3. e resumido no subcapítulo 3.3.4.

A técnica de reamostragem entre *frames* é essencial, sendo que foram exploradas as seguintes combinações:

- Reamostragem – distribuição normal – das M melhores partículas obtidas na *frame* anterior;
- Reamostragem através de otimização local (método de otimização descrito no subcapítulo 3.2.4.);
- Obter a *bounding box* do UAV na *frame* atual e utilizar a base de dados de inicialização (método descrito no subcapítulo 3.2.2. – Inicialização das partículas - e analisado no subcapítulo 4.4.1 - Reamostragem).

Porque a utilização do classificador sucessivamente em cada *frame* torna o processo lento, é guardada em cada frame a BB anterior. Esta BB é afetada pelo modelo de movimento do objeto e é acrescentado um factor de 25% sobre a sua dimensão, criando assim uma região de interesse para aplicar o algoritmo FAST e calcular a nova BB. Toda esta arquitetura vai ser explorada ao longo dos próximos subcapítulos, sendo o seu desempenho e limitações descritas ao longo do subcapítulo 4.4. – Análise da arquitetura de *tracking*.

O esquema implementado será explicado nos próximos subcapítulos, por forma a descrever corretamente a técnica de filtragem utilizada.

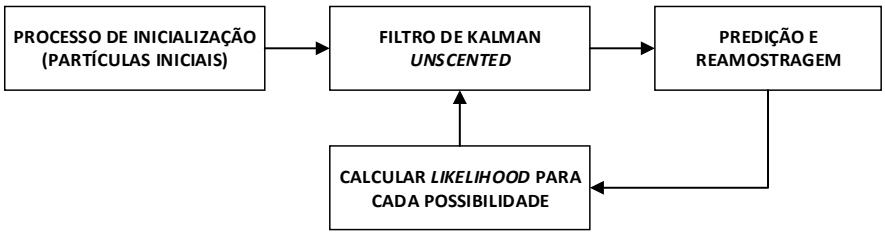


Figura 114 – Arquitetura Geral de *tracking* – Esquema simplificado.

3.3.2. DINÂMICA DO OBJETO

Na análise da dinâmica do objeto considerou-se que a velocidade do objeto é constante entre *frames* consecutivas (aceleração nula), para o movimento de rotação e translação. O movimento de translação para um objeto rígido, como é o caso do objeto de estudo, é independente do seu movimento de rotação (Haug 2012). Os movimentos de translação e rotação vão ser abordados separadamente e depois fundidos por forma a obter a dinâmica geral do sistema. Neste subcapítulo vai ser abordada de forma genérica a dinâmica de um objeto tendo em conta a sua atitude (posição 3D e orientação), sendo que as adaptações para o filtro utilizado serão descritas no subcapítulo seguinte. O modelo do sistema (neste caso de estudo rotação e translação – pose) pode ser representado por:

$$X_n = f(X_{n-1}, \nu_{n-1}) \Rightarrow X_n = f(X_{n-1}) + \nu_{n-1} \tag{3.29}$$

considerando que ν_{n-1} é ruído aditivo gaussiano. O vetor que representa a posição cartesiana do centro de gravidade do objeto de estudo é dado por:

$$p_n = [x_n, y_n, z_n]^T \tag{3.30}$$

Expandindo p_n (no instante de tempo t_n - $p(t_n)$) numa série de Taylor em relação a um tempo anterior t_{n-1} podemos escrever que:

$$p_n = p_{n-1} + T_n \dot{p}_{n-1} + \frac{T_n^2}{2}\ddot{p}_{n-1} + \frac{T_n^3}{2}\dddot{p}_{n-1} + \cdots \tag{3.31}$$

com $T_n = \Delta t = t_n - t_{n-1}$, ou seja, é o tempo entre *frames* consecutivas e $\dot{p}, \ddot{p}, \dddot{p}$ correspondem às primeiras derivadas temporais da posição. Como foi referido anteriormente vamos considerar velocidade constante entre *frames*, utilizando assim um vetor de estado de seis dimensões representado da seguinte forma:

$$p_n^{(1)} = [p_n^T; \dot{p}_n^T] \tag{3.32}$$

O modelo da dinâmica apresentada pela translação do centro de gravidade do objeto é o seguinte:

$$p_n^{(1)} = F^{(1)} p_{n-1}^{(1)} + v_{n-1}^{(1)} \tag{3.33}$$

com $v_{n-1}^{(1)}$ a ser o ruído que pode ser causado por alguma possível aceleração do objeto (incerteza existente) e $F^{(1)}$ é definido pela seguinte matriz:

$$F^{(1)} = \begin{bmatrix} 1 & 0 & 0 & T_n & 0 & 0 \\ 0 & 1 & 0 & 0 & T_n & 0 \\ 0 & 0 & 1 & 0 & 0 & T_n \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.34}$$

No que concerne à rotação esta pode ser representada de uma série de formas (como descrito no subcapítulo 2.2.) tendo como principal desvantagem a existência de algumas singularidades – *gimbal lock* - ao utilizar ângulos de Euler. Para tal vai-se optar por fazer a conversão para quaterniões e de quaterniões para ângulos de Euler conforme necessário. A conversão de ângulos de Euler para quaterniões, e considerando o angulo $\phi = [\phi_x, \phi_y, \phi_z]$, é dada por (subcapítulo 2.2.):

$$q = Q_a(\phi) = \left[\frac{\cos\|\phi\|}{2}, \frac{\sin\|\phi\|}{\|\phi\|}[\phi_x, \phi_y, \phi_z]^T\right] \tag{3.35}$$

A conversão de quaternião para ângulo de Euler é dada por:

$$\phi = A_q(q) = \frac{2\cos^{-1} q_s}{\sqrt{1-q_s^2}}[q_x, q_y, q_z]^T \tag{3.36}$$

Como foi referido anteriormente para a translação vamos considerar velocidade constante entre *frames*, utilizando assim um vetor de estado de seis dimensões também para a rotação representado da seguinte forma:

$$\phi_n^{(1)} = [\phi_n^T; \dot{\phi}_n^T] \tag{3.37}$$

com ϕ_n^T a ser a posição angular e $\dot{\phi}_n^T$ a velocidade angular. O modelo da dinâmica apresentada pela rotação do objeto é dada por:

$$\phi_n^{(1)} = g_{n-1}^{(1)}\left(\phi_{n-1}^{(1)}\right) + \rho_{n-1}^{(1)} \tag{3.38}$$

com $\rho_{n-1}^{(1)}$ a representar o ruído existente na velocidade angular, e $g_{n-1}^{(1)}()$ é a função de que vai aplicar o efeito da rotação do sistema em $n-1$ e originar a rotação prevista no instante n . Ao utilizar quaterniões é possível tratar este ruído - $\rho_{n-1}^{(1)}$ - como se fosse um quaternião ruído, obtido da seguinte forma:

$$q_{ruído} = Q_a(\Delta\phi) = Q_a(T\dot{\phi}) \tag{3.39}$$

com $\Delta\phi$ a ser a mudança incremental da rotação originada por uma rotação de velocidade angular constante - $\dot{\phi}$ - ao longo do intervalo e tempo – T. O quaternião da rotação no instante de tempo anterior $(n-1)$ é dado por:

$$q_{\phi_{n-1}} = Q_a(\phi_{n-1}) \tag{3.40}$$

É possível assim multiplicar os dois quaterniões (quaternião ruído e quaternião do instante de tempo anterior), podendo reescrever (3.38) da seguinte forma:

$$\phi_n^{(1)} = \begin{bmatrix} \phi_n^{(1)} \\ \dot{\phi}_n^{(1)} \end{bmatrix} = \begin{bmatrix} A_q(Q_a\left(T\dot{\phi}_{n-1}^{(1)}\right) Q_a\left(\phi_{n-1}^{(1)}\right)) \\ \dot{\phi}_{n-1}^{(1)} \end{bmatrix} + \rho_{n-1}^{(1)} \tag{3.41}$$

O vetor de estado combinado é dado por:

$$x_n^{(1)} = \begin{bmatrix} p_n^{(1)} \\ \phi_n^{(1)} \end{bmatrix} \tag{3.42}$$

Com um ruído de cada componente – translação e rotação - dada por:

$$r_n^{(1)} = \begin{bmatrix} v_n^{(1)} \\ \rho_n^{(1)} \end{bmatrix} \tag{3.43}$$

A dinâmica total do sistema é então dada por:

$$x_n^{(1)} = f_{n-1}^{(1)}\left(x_{n-1}^{(1)}\right) + r_{n-1}^{(1)} \tag{3.44}$$

com $f_{n-1}^{(1)}(\,)$ a ser a função que aplica o efeito de cada parâmetro do estado do sistema em $n-1$ obtendo o estado do sistema no instante n. O ruído no processo é considerado gaussiano e de média nula, podemos então considerar que este se encontra na seguinte forma:

$$r_n^{(1)} \sim N(0_{12}, Q^{(1)}) \tag{3.45}$$

com $Q^{(1)}$ a corresponde à matriz de covariância[51] do ruído no processo. O modelo de observação é dado por:

$$Z_n = h_n(X_n, \omega_n) \tag{3.46}$$

com Z_n a corresponder à função de medições, h_n a ser a função que mapeia os parâmetros do vetor de estado - X_n - no domínio de medição e ω_n a corresponder a ruído gaussiano de média nula, entrando aqui a covariância obtida na observação dos *pixels* da imagem.

[51] Medida do grau de interdependência numérica entre duas variáveis aleatórias.

O modelo de observação permite calcular a *likelihood* que uma hipótese particular gerada na imagem – observação – apresenta. A *likelihood* para uma partícula é calculada dado o modelo do objeto, o modelo de projeção para uma câmara omnidirecional e uma imagem omnidirecional conforme descrito no subcapítulo 3.2.3. Toda a dinâmica do objeto descrita neste subcapítulo vai funcionar em conjunto com uma filtragem temporal utilizando o filtro de Kalman *Unscented* – UKF – conforme descrito no subcapítulo seguinte.

3.3.3. FILTRAGEM – FILTRO DE KALMAN *UNSCENTED*

O esquema de filtragem utilizado neste trabalho foi adaptado ao problema em estudo, tendo por base filtros de Kalman *unscented* muito utilizados para a determinação de orientação em veículos espaciais utilizando a representação de atitude recorrendo a quaterniões (Julier 2002, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Turner 2012). É também feita uma transformação designada de *unscented* – *Unscented transformation* – pois é necessário estimar as médias e a covariância de uma transformação não linear, sendo este um método "elegante" de calcular de forma precisa a média e a covariância até à segunda ordem (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Julier 2002, Li, Zhang et al. 2003, Guo and Qin 2007, Zhou, Yang et al. 2011).

Apesar de neste caso concreto não existir uma limitação de processamento clara, poderia ter-se incluído nos vetores de estado – modelo - a aceleração linear e angular. Tal não foi feito porque se quer desenvolver um programa o mais eficiente possível, tendo em conta o menor processamento possível. Também o tempo entre imagens capturadas consecutivas deve centrar-se na ordem do milissegundo o que fisicamente não permite ao objeto grande mudança de atitude.

De seguida vai-se explicar pormenorizadamente cada passo da filtragem utilizada, sendo o seu funcionamento resumido no fim do capítulo.

O somatório da covariância do erro estimado anteriormente (matriz $n\ x\ n = 12\ x\ 12$) - P_{k-1} - e a covariância do ruído no processo (vetor de 12 dimensões) – Q – são transformados num conjunto $2n$ - $\{\mathcal{X}_i\}$ – de vetores de 13 dimensões (pontos sigma - representação da rotação utilizando quaternião), sendo que na primeira iteração são inicializados com valores pré-definidos. A matriz S (chamada neste contexto por matriz "raiz quadrada" – "*square root"*) é calculada por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$S = \gamma \sqrt{P_{k-1} + Q} \tag{3.47}$$

com o parâmetro escalar $\gamma = \alpha^2(n + k)$, sendo que α é um parâmetro de escala positivo que controla os efeitos de ordem mais elevada resultantes da não linearidade existente e k é outro parâmetro de escala que controla a distância entre os pontos sigma e a sua média. A matriz S apresenta a matriz da raiz quadrada da soma da covariância do erro da estimativa anterior - P_{k-1} - e da covariância do ruído no processo - Q -, afetada de um fator de peso – γ. Como P_{k-1} é uma matriz de covariância simétrica e positiva definida[52], é possível a utilização da decomposição de *Cholesky* (Higham 1990, Press, Teukolsky et al. 1996, Guo, Han et al. 2007, Zhou, Yang et al. 2011, Turner 2012) para o cálculo de S. Podendo multiplicar assim as n colunas de S por $\pm\sqrt{2n}$ para formar o conjunto $\{\mathcal{W}_i\}$ definido da seguinte forma (com $n = 12$ neste caso concreto de estudo e $i = 0$, sendo $\mathcal{W}$ uma matriz 12×12) (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$\mathcal{W}_{i,i+n} = colunas\left(\pm\sqrt{2n \cdot \gamma \cdot (P_{k-1} + Q)}\right) \tag{3.48}$$

É assim possível criar os pontos sigma - *sigma points* – através da seguinte expressão(Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$\mathcal{X}_i = \hat{x}_{k-1} + \mathcal{W}_i = \begin{bmatrix} X^{(0)} & & X^{(2n)} \\ Y^{(0)} & & Y^{(2n)} \\ Z^{(0)} & & Z^{(2n)} \\ v_x^{(0)} & \cdots\cdots & v_x^{(2n)} \\ v_y^{(0)} & \cdots\cdots & v_y^{(2n)} \\ v_z^{(0)} & \cdots\cdots & v_z^{(2n)} \\ q_x^{(0)} & \cdots\cdots & q_x^{(2n)} \\ q_y^{(0)} & \cdots\cdots & q_y^{(2n)} \\ q_z^{(0)} & \cdots\cdots & q_z^{(2n)} \\ q_w^{(0)} & \cdots\cdots & q_w^{(2n)} \\ w_x^{(0)} & & w_x^{(2n)} \\ w_y^{(0)} & & w_y^{(2n)} \\ w_z^{(0)} & & w_z^{(2n)} \end{bmatrix} \tag{3.49}$$

em que $\hat{x}_{k-1}$ representa o vetor de estado estimado anteriormente, X, Y e Z representam a posição 3D do objeto no espaço, v_x, v_y e v_z representam as velocidades lineares em cada

[52] Uma matriz real A de ordem $n \times n$ é definida positiva se $w^T A w > 0$ com $w \in \mathbb{R}^n$ (w^T é o transposto de w).

eixo, q_x, q_y, q_z e q_w a representação do quaternião de orientação e w_x, w_y e w_z as velocidades angulares segundo cada eixo.

Neste caso específico de estudo no que concerne à rotação (quaternião utilizado no vetor de estado) não é possível uma adição direta conforme descrito na equação (3.52). Para calcular os pontos sigma vai então ser calculado um quaternião erro que vai afetar cada ponto sigma (i), sendo este calculado obtendo o erro - δ_{q_w} - para a parte real - q_w- e erro - $\delta_{q_{x,y,z}}(i)$ – para a parte imaginária - q_x, q_y, q_z – da seguinte forma (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Cheon and Kim 2007, Guo and Qin 2007):

$$\delta_{q_w}(i) = \frac{-a\|\mathbb{X}^{qw}(i)\|^2 + f\sqrt{f^2 + (1-a^2)\|\mathbb{X}^{qw}(i)\|^2}}{f^2 + \|\mathbb{X}^{qw}(i)\|^2} \tag{3.50}$$

com $a = 1$, $f = 2(a+1)$ e com $\mathbb{X}^{q_w}(i)$ a corresponder a:

$$\mathbb{X}^{q_w}(i) = \begin{bmatrix} \mathcal{W}(7,0) & \dots & \mathcal{W}(7,12) \\ \mathcal{W}(8,0) & \dots & \mathcal{W}(8,12) \\ \mathcal{W}(9,0) & \dots & \mathcal{W}(9,12) \end{bmatrix} \tag{3.51}$$

$$\delta_{q_{x,y,z}}(i) = f^{-1}[a + \delta_{q_w}(i)]\mathbb{X}^{q_w}(i) \tag{3.52}$$

No caso de termos $i = 0$, $\|\mathbb{X}^{q_w}(0)\|^2$corresponde ao quadrado da norma do vetor $\mathbb{X}^{q_w}(0) = [\mathcal{W}(7,0), \mathcal{W}(8,0), \mathcal{W}(9,0)]$, possibilitando assim o cálculo de $\delta_{q_w}(0)$ e $\delta_{q_{x,y,z}}(0)$. Da equação (3.49) e (3.51) são obtidos os diversos quaterniões erro que vão ser multiplicados pela parte correspondente à atitude criando os pontos sigma $\mathcal{X}_i$. De referir que os pontos sigma são criados através multiplicação dos quaterniões erro e dos conjugados dos quaterniões erro originando no total 24 pontos sigma ($2n$).

De seguida o modelo do processo - A ( ) - é aplicado ao conjunto de pontos sigma - $\mathcal{X}_i$ -, de acordo com o seguinte (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$\mathcal{Y}_i = A(\mathcal{X}_i, 0) \tag{3.53}$$

Não é contemplado ruído adicional neste ponto pois este já foi contemplado na representação da distribuição dos pontos sigma criados pela equação (3.49). Em termos práticos a dinâmica do sistema (considerando velocidade constante entre frames) descrita no subcapítulo 3.3.2 vai ser representada da seguinte forma:

$$A^{translação} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.54}$$

Em que a translação é considerado um movimento linear entre *frames*, resultando em:

$$\mathcal{Y}_i^{translação} = A^{translação} \mathcal{X}_i^{translação} \tag{3.55}$$

sendo que $\mathcal{X}_i^{translação}$ corresponde à parte de translação presente nos pontos sigma criados X, Y, Z, v_x, v_y e v_z. No que concerne à rotação, a não linearidade vai ser contornada adotando toda a sua representação em quaterniões, gerando um quaternião - q_Δ – que vai representar o modelo de movimento em rotação do objeto. Para este cálculo necessitamos apenas da informação das velocidades angulares segundo cada eixo, sendo este calculado da seguinte forma (Julier 2002, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Turner 2012):

$$A^{rotação} = q_\Delta \tag{3.56}$$

$$q_\Delta = [\cos\left(\frac{\alpha_\Delta}{2}\right), \vec{e}_\Delta \sin\left(\frac{\alpha_\Delta}{2}\right)] \tag{3.57}$$

com α_Δ (ângulo) a ser dado por (Julier 2002, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Turner 2012):

$$\alpha_\Delta = |\vec{w}_k| \cdot \Delta t \tag{3.58}$$

com $|\vec{w}_k|$ a representar o módulo das velocidades angulares segundo cada eixo, obtido por:

$$|\vec{w}_k| = \sqrt{(w_x)^2 + \left(w_y\right)^2 + (w_z)^2} \tag{3.59}$$

$\vec{e}_\Delta$ (eixos) são obtidos através da seguinte expressão (Julier 2002, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Turner 2012):

$$\vec{e}_\Delta = \frac{\vec{w}_k}{|\vec{w}_k|} \tag{3.60}$$

De forma equivalente é também gerado um quaternião perturbação - q_w – mas a partir de um vetor de velocidades aleatório gerado a partir de uma distribuição normal de valor médio nulo. A estimativa *a priori* - $\hat{x}_k^-$ - é calculada como a média dos pontos sigma transformados - $\mathcal{Y}_i$ – da seguinte forma (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Cheon and Kim 2007, Guo and Qin 2007):

$$\hat{x}_k^- = \frac{\sum_{i=1}^{24} W_i^{(m)} y_i}{\left|\sum_{i=1}^{24} W_i^{(m)} y_i\right|} \tag{3.61}$$

Usando os pesos W_i provenientes da transformação *unscented* (Wan and Van Der Merwe 2000, Julier 2002).Para o ponto sigma central - $\mathcal{Y}_0$ - o peso é dado por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Cheon and Kim 2007, Guo and Qin 2007):

$$W_0^{(m)} = \frac{\lambda}{(n+\lambda)} \tag{3.62}$$

Para os restantes pontos o peso é dado por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Cheon and Kim 2007, Guo and Qin 2007):

$$W_i^{(m)} = \frac{\lambda}{\{2(n+\lambda)\}} \qquad i = 1, \ldots, 2n \tag{3.63}$$

O peso referente ao ponto sigma central (equação (3.62)) afeta diretamente a magnitude dos erros de quarta e quinta ordem para distribuições anteriores simétricas (Julier 2002). É retirado o vetor da média dos pontos sigma transformados - $\hat{x}_k^-$ - a cada elemento de acordo com (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$[\mathcal{Y}_i - \hat{x}_k^-] \leftrightarrow \mathcal{W}_i' \tag{3.64}$$

No que concerne à posição, velocidades angulares e lineares $\mathcal{W}_i'$ é obtido pela diferença entre os seus componentes sendo que o mesmo não acontece na parte de orientação – quaternião. Para tal é necessário calcular essa diferença da seguinte forma (Julier 2002, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Turner 2012):

$$\delta_{x,y,z,w}^{atitude} = q_i \bar{q}^{-1} \tag{3.65}$$

$$\mathcal{W}_i^{atitude'} = f\frac{\delta_{x,y,z}^{atitude}}{a+\delta_w^{atitude}} \tag{3.66}$$

A covariância do processo *a priori* - P_k^- - é calculada através do conjunto – $\{\mathcal{W}_i'\}$ – da seguinte forma (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Cheon and Kim 2007, Guo and Qin 2007):

$$P_k^- = \sum_{i=1}^{2n} W_i^{(c)} \mathcal{W}_i' \mathcal{W}_i'^{\mathrm{T}} \tag{3.67}$$

Com os pesos a serem dados por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Cheon and Kim 2007, Guo and Qin 2007):

$$W_i^{(c)} = \frac{1}{\{2(n+\lambda)\}} \qquad i = 1, \dots, 2n \tag{3.68}$$

$$W_0^{(c)} = \frac{\lambda}{(n+\lambda)} + (1 - \alpha^2 + \beta) \tag{3.69}$$

em que β é um parâmetro que controla o peso do ponto sigma central para o cálculo da covariância, sendo que os restantes parâmetros já foram descritos anteriormente ao longo deste subcapítulo. Sendo que acaba aqui o passo designado por predição. De seguida o modelo de medição é aplicado aos pontos sigma – $\{\mathcal{Y}_i\}$ – por forma a projetar estes pontos no espaço de medição (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011).

$$\mathcal{Z}_i = H(\mathcal{Y}_i, 0) \tag{3.70}$$

Os pontos sigma são assim projetados num espaço em que cada ponto corresponde a um vetor de 7 dimensões, correspondendo à posição 3D do objeto e ao quaternião atitude (as velocidades lineares e angulares são descartadas).

É calculada a média de – $\{\mathcal{Z}_i\}$ – de acordo com (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$z_k^- = \sum_{i=1}^{2n} W_i^{(m)} \mathcal{Z}_i \tag{3.71}$$

A estimativa de covariância da medição - P_{zz} – é dada por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$P_{zz} = \sum_{i=1}^{2n} W_i^{(m)} [\mathcal{Z}_i - z_k^-][\mathcal{Z}_i - z_k^-]^{\mathrm{T}} \tag{3.72}$$

O cálculo da inovação é dado por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$v_k = z_k - z_k^- \tag{3.73}$$

Em que z_k é a medição atual e z_k^- é a media da estimativa de medição calculada de acordo com a equação (3.71). A covariância da inovação - P_{vv} - é determinada ao adicionar o ruído de medição – R – à covariância - P_{zz} - obtida na equação (3.72) da seguinte forma (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$P_{vv} = P_{zz} + R \tag{3.74}$$

A matriz de correlação cruzada - P_{xz} – é calculada a partir dos conjuntos $\{\mathcal{Z}_i\}$ e $\{\mathcal{Y}_i\}$ da seguinte forma (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$P_{xz} = \sum_{i=1}^{2n} W_i^{(m)} [\mathcal{Y}_i - \hat{x}_k^-][\mathcal{Z}_i - z_k^-]^{\mathrm{T}} \tag{3.75}$$

O ganho de Kalman é calculado através da seguinte expressão (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$K_k = P_{xz} P_{vv}^{-1} \tag{3.76}$$

A estimativa *a posteriori* é obtida através de (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$\hat{x}_k = \hat{x}_k^- + K_k v_k \tag{3.77}$$

O quaternião é calculado mais uma vez utilizado o vetor de parâmetros modificados de Rodriguez (como efetuado nas equações (3.49) e (3.50)). O cálculo da covariância da estimativa de erro - P_k – conclui a parte de correção do filtro e é dada por (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

$$P_k = P_k^{-1} - K_k P_{vv} K_k^{\mathrm{T}} \tag{3.78}$$

O cálculo dos erros e do valor médio dos quaterniões para o cálculo das covariâncias utiliza os parâmetros modificados de *Rodriguez – Modified Rodriguez Parameteres* – que resulta numa estimativa mais estável que outros métodos iterativos ou baseados em heurística. Sendo esta filtragem efetuada em cada iteração – entre captura de *frames* – de acordo com os passos descritos anteriormente. Uma representação esquemática pode ser vista através da seguinte figura:

O esquema de filtragem pode ser resumido da seguinte forma (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Crassidis and Markley 2003, Kraft 2003, Cheon and Kim 2007, Guo and Qin 2007, Zhou, Yang et al. 2011):

1. A soma da covariância do erro anterior - P_{k-1} – e a covariância do ruido do processo – Q – são transformados num conjunto $2n$ de vetores de 12 dimensões - $\{\mathcal{W}_i\}$;
2. A estimativa do estado anterior - $\hat{x}_{k-1}$ é aplicada a $\{\mathcal{W}_i\}$, e como resultado temos o conjunto $2n$ de 13 dimensões - $\{\mathcal{X}_i\}$ – que são os pontos sigma;
3. O modelo do processo $A()$ transforma $\{\mathcal{X}_i\}$ em $\{\mathcal{Y}_i\}$;
4. A estimativa *a priori* - $\hat{x}_k^-$ - é calculada como a média dos pontos sigma transformados - $\{\mathcal{Y}_i\}$;
5. O conjunto $\{\mathcal{Y}_i\}$ é transformado num conjunto de 12 dimensões - $\{\mathcal{W}_i'\}$ – ao remover o vetor de média - $\hat{x}_k^-$ - de cada elemento e depois convertendo a parte de rotação (quaternião) em vetor de rotação;
6. A covariância do processo *a priori* - P_k^- - é calculada a partir de $\{\mathcal{W}_i'\}$;
7. É aplicado o modelo de medição – H – aos pontos sigma $\{\mathcal{Y}_i\}$ para os projetar no espaço de medição;
8. A média de $\{\mathcal{Z}_i\}$ é calculada, dada a estimativa de medição z_k^-. Esta é comparada com a medição atual - z_k, sendo a sua diferença v_k – a inovação;
9. A covariância da inovação - P_{vv} – é determinada adicionando o ruído de medição – R – à covariância - P_{zz} – do conjunto $\{\mathcal{Z}_i\}$;
10. A matriz de correlação cruzada - P_{xz} – é calculada através dos conjuntos $\{\mathcal{W}_i'\}$ e $\{\mathcal{Z}_i\}$;
11. O ganho de Kalman - K_k – é calculado a partir de P_{xz} e P_{vv} e depois utilizado para calcular a estimativa *a posteriori* - $\hat{x}_k$- e estimar a covariância do erro - P_k.

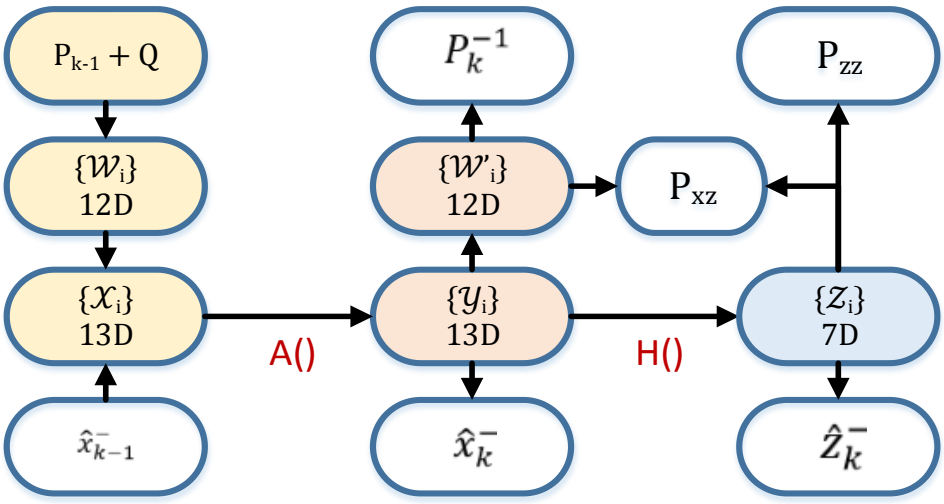


Figura 115 – Representação esquemática geral do filtro utilizado.

3.3.4. RESUMO GERAL – ARQUITECTURA DE TRACKING

A arquitetura de *tracking* proposta visa a combinação de um filtro de partículas com um UKF, utilizando assim um filtro de partículas *unscented* adaptado (Van Der Merwe, Doucet et al. 2000, Rui and Chen 2001, Li, Zhang et al. 2003, Guo and Qin 2007, Guo, Han et al. 2007, Zhou, Yang et al. 2011).

Neste momento o objetivo é introduzir filtragem temporal entre *frames*, introduzindo filtragem através de um UPF adaptado estudando o seu modo de funcionamento e implementação. Podemos resumir o conteúdo descrito nos subcapítulos anteriores da seguinte forma:

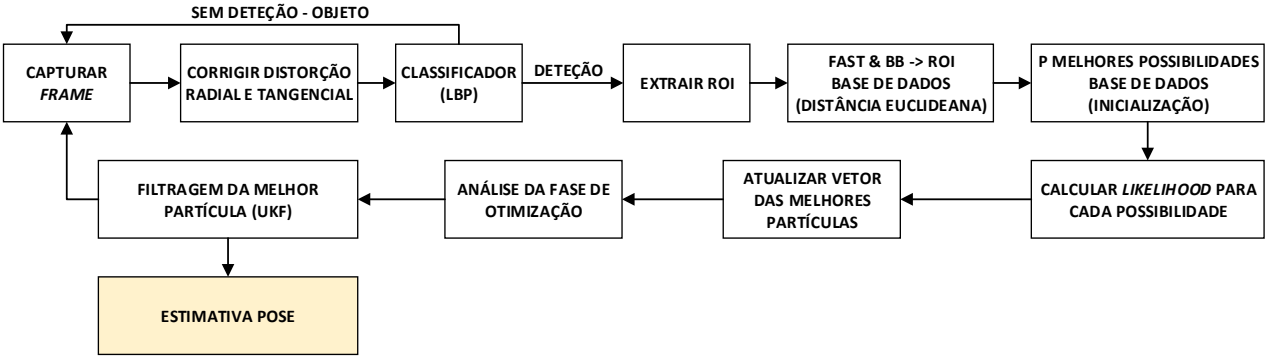


Figura 116 – Arquitetura Geral de *tracking*.

O algoritmo é descrito de forma simplificada ao longo do apêndice D – subcapítulo D.2. – *Tracking*.

3.4. CONCLUSÕES OU SÍNTESE

Este capítulo faz uma abordagem à descrição geral do sistema, que foi dividido em duas fases distintas: Deteção de pose e arquitetura de *tracking*. Devido à importância deste capítulo para a completa compreensão do trabalho desenvolvido, procurou-se fazer uma descrição a mais completa possível do sistema desenvolvido.

Foi descrita a fase de deteção de pose, dividindo o seu funcionamento em 4 fases: Deteção da aeronave, inicialização de partículas, avaliação de partículas e otimização de pose. Na fase de deteção da aeronave é efetuada uma segmentação utilizando um classificador, para poder assim detetar a localização provável do UAV para a segunda fase de inicialização das partículas e do filtro implementado. Na fase de avaliação das partículas a

sua *likelihood* é calculada por duas formas distintas (diferença entre histogramas interior e exterior e uma abordagem hibrida – contornos e a descrita anteriormente) consoante a distância do UAV à câmara. Na fase de otimização de pose é adotado um esquema de reamostragem baseada em algoritmos genéticos, evitando assim mínimos locais e falta de diversidade no filtro implementado.

A arquitetura de *tracking* implementada vai buscar muito à fase descrita anteriormente, sendo que a alteração mais substancial é a possibilidade de utilização de informação entre sequências de *frames* melhorando assim a estimativa do sistema. O tipo de filtragem escolhida é baseada num filtro de Kalman *Unscented* pois a relação de não linearidade entre a estimativa de orientação e a medição esperada impede a utilização do filtro de Kalman clássico. A principal diferença face ao filtro de Kalman estendido (EKF) é que este aproxima a distribuição de probabilidade Gaussiana através de um conjunto de pontos (pontos sigma) enquanto o EKF recorre à linearização das equações do modelo não-linear do sistema. Isto leva a que o UKF leve a resultados mais precisos (são utilizadas as equações originais) e um menor tempo de processamento (pois não necessita de calcular a matriz Jacobiano – matriz de derivadas parciais). Recorre-se também à representação da rotação utilizando quaterniões devido à sua simplicidade de implementação quando combinados com o UKF.

CAPÍTULO IV
RESULTADOS EXPERIMENTAIS

4.1. INTRODUÇÃO

Neste capítulo são apresentados alguns pormenores de implementação, tempos de execução e análise de imagens reais e sintéticas (criadas artificialmente utilizando o modelo CAD 3D do UAV). Nas imagens reais obtidas não existe informação de referência – *ground truth*, sendo os resultados avaliados qualitativamente através da observação do modelo CAD projetado nas imagens. Para avaliar quantitativamente a performance das arquiteturas desenvolvidas é efetuada uma análise estatística da estimação de pose para um grande número de imagens sintéticas geradas com referência - *ground truth.* Os métodos que vão ser descritos foram implementados em C/C++ num 2,40 GHz Intel i7 CPU com uma NVIDIA *GeForce* GT 750M. Todos os tempos de processamento e resultados apresentados este capítulo vão ser referentes a esta plataforma.

No subcapítulo 4.2 é descrito o processo de *rendering* do objeto utilizado – modelo CAD de um UAV – abordando assim os métodos utilizados e o seu desempenho específico, no subcapítulo 4.3 é analisada a arquitetura de deteção de pose descrita ao longo do subcapítulo 3.2 analisando o seu desempenho em cada uma das fases do algoritmo e no subcapítulo 4.4 é analisada a arquitetura de *tracking* descrita ao longo do subcapítulo 3.3 analisando também aqui o seu desempenho.

4.2. *RENDERING* DO OBJETO

Uma das particularidades do tipo de abordagem utilizada é a necessidade de efetuar a projeção do modelo CAD 3D para cada hipótese a testar - partícula, sendo esta uma ação que tem de ser efetuada de forma eficiente. Foram abordadas algumas formas de efetuar

esta projeção nomeadamente utilizando unicamente o CPU e utilizando o GPU (Figura 117).

O formato de ficheiro CAD utilizado – ".obj" – é *human-readable data*[53] facilitando a leitura e compreensão da informação possibilitando assim uma utilização fácil e intuitiva. Neste é possível encontrar informação sobre os vértices 3D do objeto, as normais e as faces que o constituem. Utilizando um modelo de UAV com 64368 vértices, 25889 normais e 57275 faces é necessário um tempo de carregamento para memória e posterior envio para a GPU de aproximadamente 430 ms.

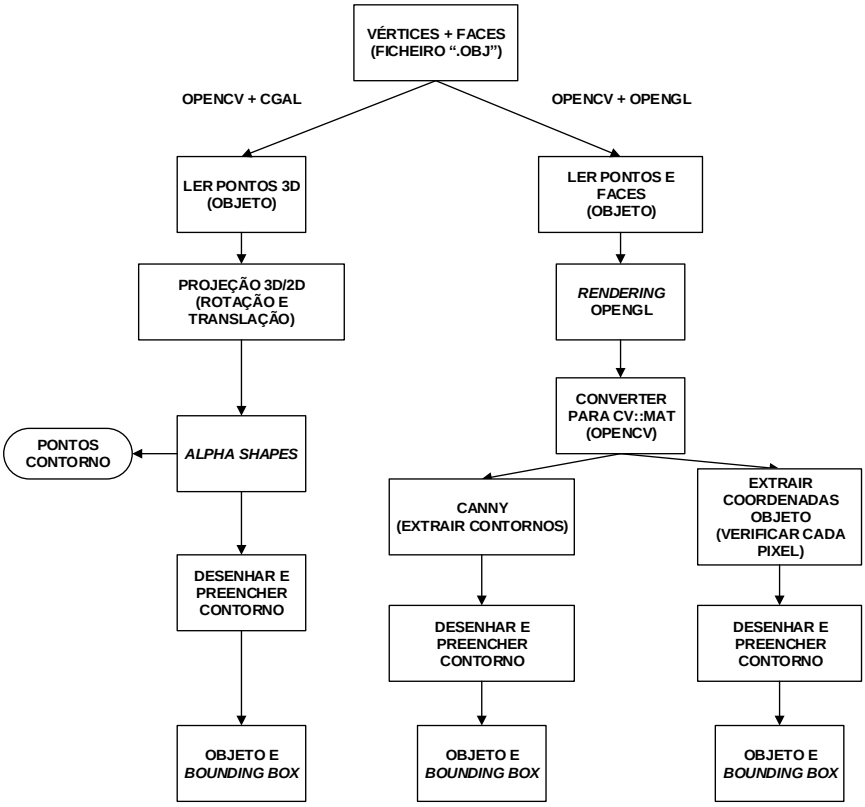


Figura 117 – *Rendering* objeto CPU e GPU – OPENGL.

Por forma a testar o desempenho da CPU nesta situação, por forma a comparar o seu desempenho com o obtido pela GPU, foram lidos os pontos do objeto 3D e de seguida projetados (Figura 117). Por forma a obter o respetivo contorno do objeto foi utilizada uma técnica descrita por *Alpha Shape* (Edelsbrunner, Kirkpatrick et al. 1983).

Esta técnica permite obter os pontos que constituem o contorno do objeto podendo posteriormente preencher o seu interior, é porém visível que a utilização de poucos pontos

[53] Expressão utilizada para referir um formato que é facilmente lido e interpretado por um ser humano.

torna o contorno pouco preciso. A utilização de mais pontos resolve a situação, no entanto o tempo de cálculo necessário também aumenta.

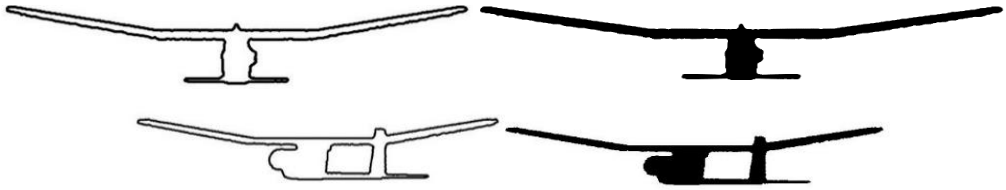


Figura 118 – Exemplo de utilização de *Alpha Shape* ao conjunto de pontos que constituem o objeto.

Outras técnicas poderiam também ser utlizadas como e.g. calcular o ponto mais próximo *(k-means*) e desenhar um círculo com esse raio. O que de acordo com a análise da Figura 119 se percebe que a criação do contorno é ineficiente, não correspondendo o obtido ao real (existe uma dilatação do contorno). Mas muitas outras abordagens foram exploradas e que não apresentaram resultados válidos.

Figura 119 – Objeto preenchido (Esq.) e objeto ao desenhar um círculo preenchido em cada ponto (raio = distância ao ponto mais próximo) (Dir.)

O tempo médio para a projeção de 18144 pontos é de aproximadamente 1 ms e a aplicação de *Alpha Shapes* para obter o respetivo pontos de contorno demora aproximadamente 175 ms. Isto faz com que para 100 projeções sejam necessários 1750 ms – ≅ 0,67 fps – o que torna esta técnica impossível de aplicar em tempo real. Para conseguir efetuar *rendering* em tempo real é necessário recorrer à GPU (foi utilizado a estrutura de imagem da biblioteca OpenCV - Figura 117), por forma a obter tempos compatíveis com uma aplicação que deve correr em tempo real.

Inicialmente foi utilizada a biblioteca OpenGL por forma a efetuar o *rendering* do objeto na GPU para um *buffer* denominado – *frame buffer object* (FBO) – e posteriormente ler o objeto representado para uma matriz imagem na CPU para posterior processamento (Figura 120). O rendering propriamente dito do objeto em causa, num FBO de 1280x720 *pixels*, demora aproximadamente 0 ms utilizando uma NVIDIA *GeForce* 750M e 20 ms utilizando uma Intel HD *Graphics* 4600 para uma simples projeção. Esta diferença deve-se maioritariamente devido à diferente capacidade de processamento, pois a Intel HD *Graphics* 4600 é uma placa gráfica desenvolvida para apresentar um baixo consumo energético.

É importante ter em consideração que o OpenGL não contempla a distorção presente na câmara, mas deve contemplar os parâmetros intrínsecos da câmara por forma a fazer a projeção do objeto no sítio correto. E foi exatamente isso que foi feito, os parâmetros obtidos através da calibração foram introduzidos e testados em *frames* reais verificando-se assim o seu correto funcionamento Figura 121.

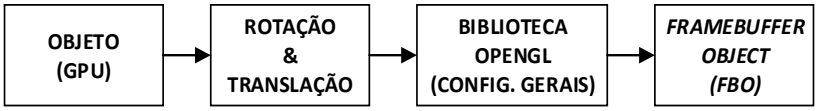


Figura 120 – Esquema simplificado – FBO.

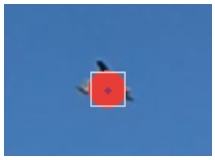


Figura 121 – Exemplo de projeção – *rendering* de esfera em OpenGL.

O tempo de leitura médio do FBO (três canais BGR – *GL_BGR* – *glReadPixels()*) é de aproximadamente 3 ms (333 fps) por projeção utilizando uma NVIDIA *GeForce* 750M e 5 ms (200 fps) utilizando uma Intel HD *Graphics* 4600, sendo esta leitura feita de forma síncrona. A primeira projeção apresenta sempre um *overhead*, pois a placa gráfica ainda está a efetuar processamento específico (neste caso toda a configuração é efetuada antes da primeira projeção), sendo de 46 ms para a NVIDIA e 29 ms para a Intel. Por forma a contornear esta situação pode utilizar-se o comando OpenGL – *glFinish*() – antes de iniciar o processo, por forma a garantir que todos os processos presentes na GPU são executados antes de prosseguir. É possível diminuir o tempo de transferência fazendo o *rendering* e transferência para a CPU de apenas uma gama de cor (e.g. azul – *GL_BLUE*), obtendo um tempo médio de 2 ms utilizando a NVIDIA e 4 ms utilizando a Intel.

Uma das vantagens de utilização da biblioteca OpenGL é eliminar as superfícies sobrepostas apresentando apenas as faces visíveis, utilizando o algoritmo *Z-buffer*. O seu funcionamento é muito simples, cada vez que um polígono é processado e tem uma coordenada Z inferior à existente em *buffer* o buffer é atualizado com o novo polígono.

Para o restante cálculo presente na Figura 117, em que os *pixels* pertencentes ao objeto e ao exterior são precisos em média 1 ms para obter o vetor de *pixels* pertencentes ao objeto e 3 ms para obter os *pixels* da região exterior. O cálculo dos respetivos histogramas demora em média aproximadamente 1 ms.

Por forma a acelerar o processo foi explorada a utilização da biblioteca OpenMP (Chandra 2001, Chapman, Jost et al. 2008) por forma a paralelizar na CPU as funções existentes para o cálculo dos histogramas e distância para cada partícula. Como cada partícula tem um número de *pixels* nos seus histogramas variável verificava-se um tempo de processamento maior em casos em que num determinado processador precisa-se de mais tempo em relação aos restantes gerando uma espera que atrasa o processamento global. No entanto esta paralelização não resolve o principal problema que é o atraso existente na transferência de dados entre a GPU e a CPU. Para resolver esta lacuna no *rendering* foi explorada a arquitetura CUDA, que através de programação dedicada permite tirar proveito da unidade de processamento gráfico NVIDIA utilizada. Também foi utilizado o - *pixel buffer object* (PBO) – que permite transferência de *pixels* de forma assíncrona.

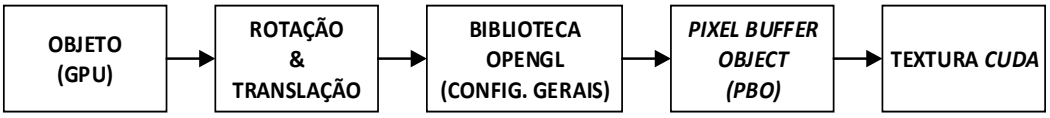


Figura 122 – Esquema simplificado – PBO & Textura CUDA.

O *rendering* para o PBO (1280x720) e posterior leitura para uma matriz imagem na CPU (formato OpenCV) em escala de cinzentos demora em média 2 ms (500 fps) muito semelhante à leitura direta síncrona de um canal proveniente do FBO (Figura 122 e Figura 123). A projeção para PBO, transferência para textura CUDA e posteriormente para PBO (Figura 123) novamente demora em média 2,32 ms (430 fps).

A grande vantagem de utilizar esta arquitetura é a capacidade de efetuar processamento específico na GPU, sendo mais rápido do que na CPU (múltiplos processadores – paralelização) e podendo diminuir o que se envia da GPU para a CPU diminuindo o tempo de processamento necessário nesta tarefa. O rendering para PBO e respetiva cópia para textura CUDA (1280x720) para poder efetuar processamento leva um tempo médio de 1,47 ms (680 fps). E é esta arquitetura que tem de ser explorada por forma a obter os tempos de processamento adequados para este tipo de aplicação funcionar em tempo real.

Figura 123 – PBO/Textura CUDA/PBO (esquerda) e PBO/Textura CUDA/CPU (direita).

O esquema de funcionamento da GPU (como visto na Figura 123) é representado por:

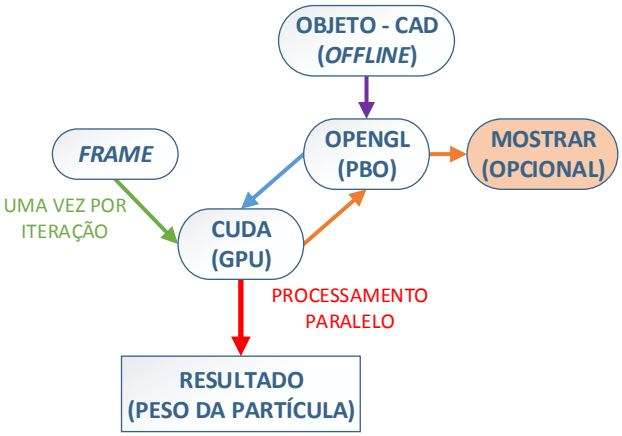


Figura 124 – Utilização GPU (CUDA) – Esquemático geral.

Sendo atualmente o processamento do peso de cada partícula efetuada na CPU, sendo que um dos objetivos finais do projeto em que esta dissertação se encontra inserido a utilização da GPU (Figura 125) para esse efeito.

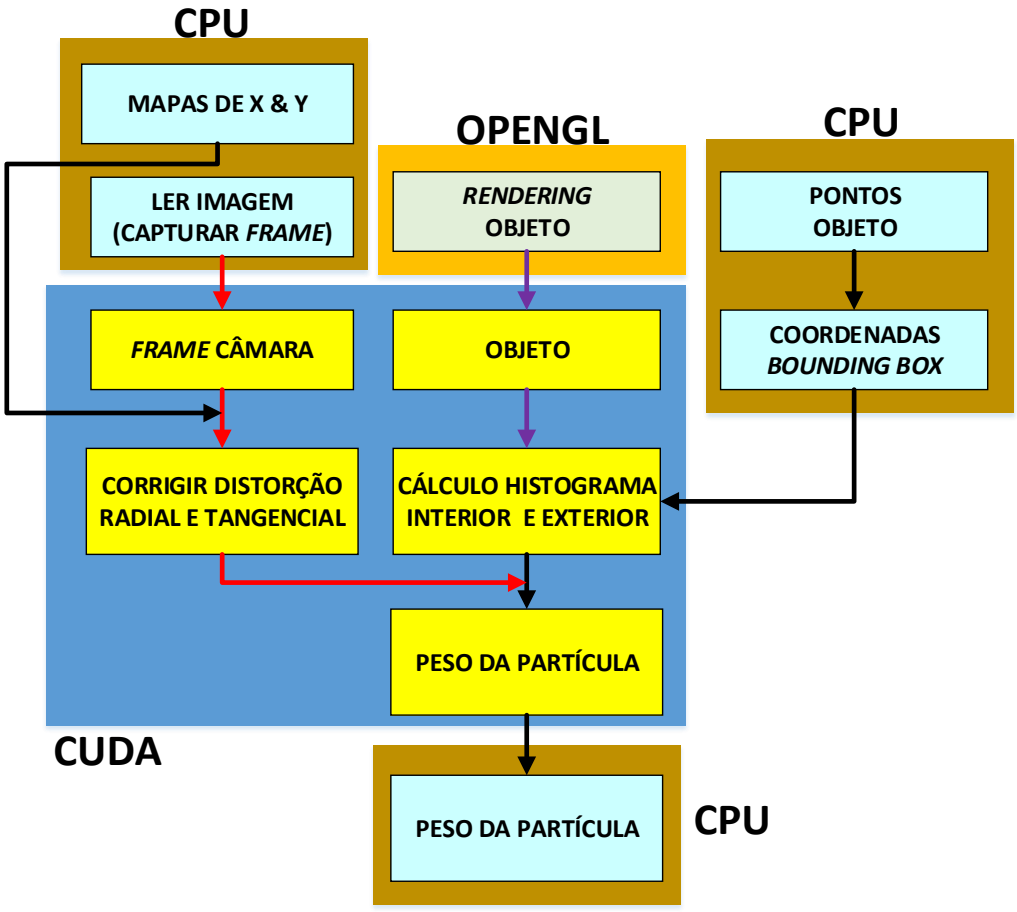


Figura 125 – Arquitetura Geral de cálculo da *likelihood* usando a GPU – arquitetura CUDA.

A correção da distorção na imagem é atualmente feita na GPU com sucesso, através da seguinte expressão:

$$Frame_{corrigida}(x,y) = Frame_{capturada}(mapa_x(x,y), mapa_y(x,y)) \quad (4.1)$$

em que $Frame_{capturada}(x,y)$ corresponde à frame obtida pela câmara, $mapa_x$ e $mapa_y$ aos mapas da coordenada X e Y respetivamente calculados utilizando a CPU (cerca de 20 ms para o seu cálculo) para possibilitar o remapeamento da imagem capturada (correção da distorção radial e tangencial) e $Frame_{corrigida}(x,y)$ corresponde à imagem final com a correção da distorção radial e tangencial efetuada. Ao calcular os mapas para remapeamento na CPU de acordo com os parâmetros de distorção obtidos através de calibração, é possível fazer com que esta correção seja rápida e eficiente. O envio dos mapas para a GPU (1280 x 720 números do tipo *float* por mapa) demora aproximadamente 954 ms, sendo estes mapas enviados unicamente uma vez – durante a inicialização do sistema. Após a inicialização do sistema o envio de uma *frame* capturada (1280 x 720) para a GPU demora cerca de 1 ms e a correção da distorção demora cerca de 1,46 ms no caso de utilização da aproximação bilinear (cerca de 5 ms na CPU) e 0,77 ms no caso de aproximação linear (cerca de 3 ms na CPU). O *rendering* do objeto e correspondente transferência para a GPU (CUDA) demora sensivelmente 1,45 ms. Sendo que ainda faltam terminar alguns pormenores da implementação utilizando a arquitetura CUDA, nomeadamente a forma eficiente de cálculo dos *pixels* que constituem o histograma exterior (cálculo da *likelihood*) mas as vantagens da sua utilização são bastante evidentes. Foi testada uma abordagem em que seria calculada a *bounding box* do objeto (para obter o histograma exterior) na CPU (Figura 125) utilizando uma amostragem de pontos 3D igualmente espaçados ao longo da superfície do UAV sendo posteriormente enviado para a GPU os resultados deste cálculo. Para tal foi utilizado um conjunto de referência de 7500 pontos (Figura 126) e amostrados pontos uniformemente espaçados entre si (Tabela 8) com distâncias entre si de:0,03; 0,025; 0,02; 0,015; 0,01; 0,009; 0,006; 0,004 e 0,002 (correspondendo a distâncias em metros no Mundo real).

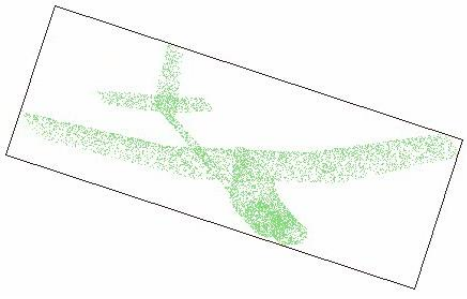


Figura 126 – Conjunto de referência para teste – 7500 pontos.

Distância entre pontos (metros):	Número de pontos 3D objeto:
0,002	6522
0,004	6032
0,006	5247
0,009	4044
0,01	3630
0,015	2181
0,02	1365
0,025	862
0,03	592

Tabela 8 – Número de pontos 3D utilizados para representar o UAV em função do espaçamento entre pontos adjacentes.

Seguidamente foi calculada a *bounding box* resultante para 2000 diferentes poses do UAV (à semelhança do efetuado para o cálculo da base de dados – subcapítulo 3.2.2.), tendo sido registadas as posições do centro, a área e o tempo de cálculo da respetiva *bounding box*. Tendo sido comparados os resultados obtidos para cada caso com o obtido utilizando o conjunto de referência (Figura 126).

Podemos verificar pela análise do Gráfico 3 que o erro no cálculo do centro da *bounding box* decresce com a diminuição da distância entre pontos (maior número de pontos utilizados), obtendo um valor médio de 0 quando é utilizada uma distância entre pontos de 0,002 (como apresentado na Tabela 9).

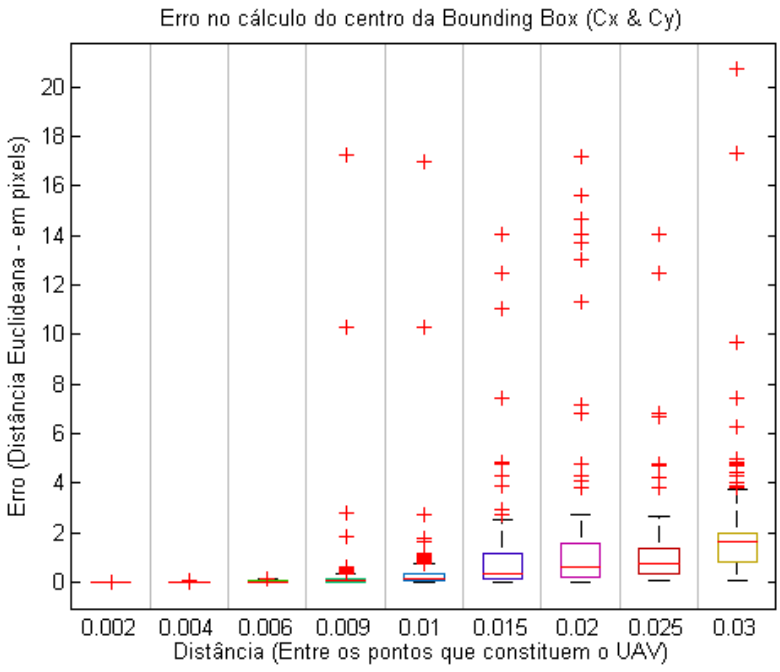


Gráfico 3 – Erro no cálculo do centro da Bounding Box.

Distância entre pontos (metros):	Valor médio (*pixels*):	Mediana (*pixels*):
0,002	0	0
0,004	0,002	0
0,006	0,03	0
0,009	0,19	0,09
0,01	0,33	0,16
0,015	0,78	0,35
0,02	1,21	0,63
0,025	0,99	0,78
0,03	1,67	1,63

Tabela 9 – Erro no cálculo do centro da Bounding Box - *pixels*.

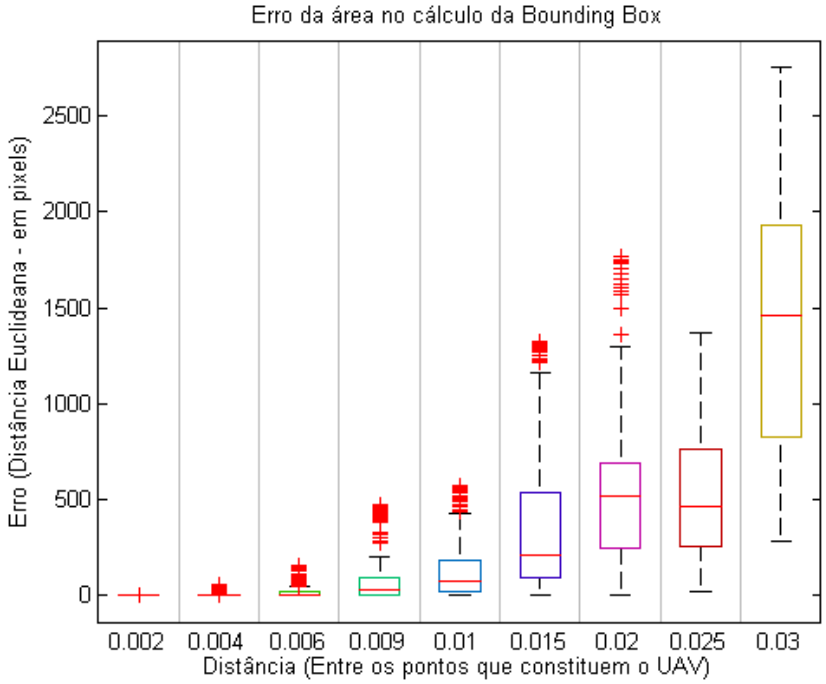


Gráfico 4 - Erro no cálculo da área da Bounding Box.

Distância entre pontos (metros):	Valor médio (*pixels*):	Mediana (*pixels*):
0,002	0,05	0
0,004	1,5	0
0,006	16	0,1
0,009	80	31
0,01	138	73
0,015	345	213
0,02	540	515
0,025	547	467
0,03	1417	1458

Tabela 10 – Erro no cálculo da área da Bounding Box – *pixels*.

Podemos analogamente também verificar pela análise do Gráfico 4 que o erro no cálculo da área da *bounding box* decresce com a diminuição da distância entre pontos (maior número de pontos utilizados), obtendo um valor médio de 0,05 quando é utilizada uma distância entre pontos de 0,002 (como apresentado na Tabela 10). A utilização de uma distância menor entre pontos sucessivos apresenta no entanto a principal desvantagem de terem de ser utilizados mais pontos 3D, o que leva a um aumento do tempo de processamento necessário (Gráfico 5) obtendo um valor médio de 0,76 ms quando é utilizada uma distância entre pontos de 0,002 (como apresentado na Tabela 11). Se tivermos de executar este cálculo e.g. 100 vezes por iteração para uma distância entre pontos de 0,002 vamos demorar em média 76 ms por iteração só no cálculo dos pontos constituintes da *bounding box.* Devem portanto ser abordadas alternativas a esta abordagem, por forma a obter um maior rendimento do sistema.

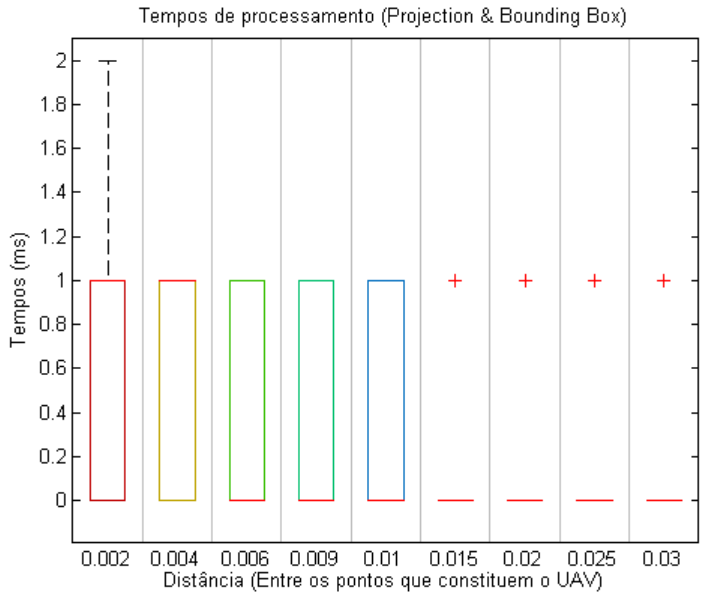


Gráfico 5 – Tempos de processamento necessário para cada exemplo de teste (ms).

Distância entre pontos (metros):	Valor médio (ms):	Mediana (ms):
0,002	0,76	1
0,004	0,56	1
0,006	0,47	0
0,009	0,38	0
0,01	0,35	0
0,015	0,20	0
0,02	0,16	0
0,025	0,11	0
0,03	0,08	0

Tabela 11 – Tempos de processamento necessário para cada exemplo de teste (ms).

Durante o normal funcionamento do filtro é possível que durante algum dos processos que o constituem as partículas geradas possam estar fora da *frame* de observação, logo é preciso validar cada partícula verificando a sua localização em tempo real conforme descrito na Figura 127.

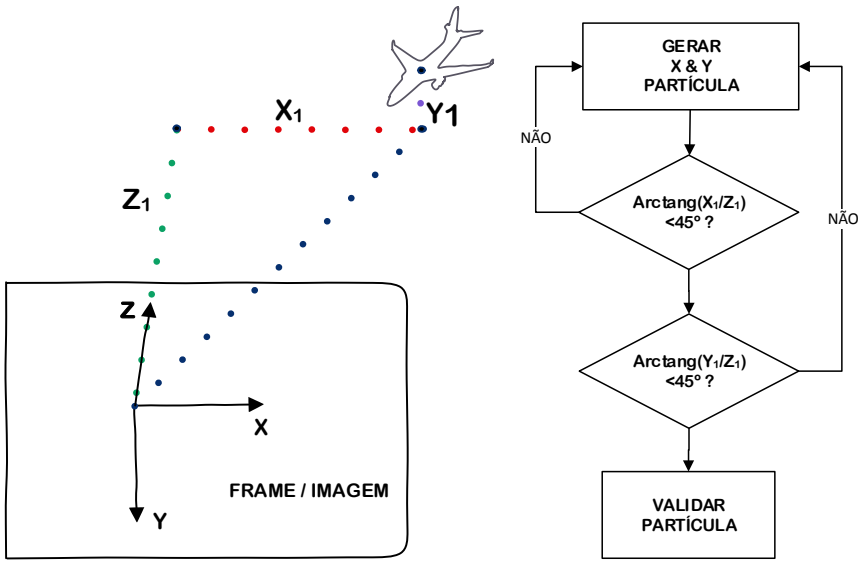


Figura 127 – Verificação se a partícula criada está dentro da *frame* de observação.

4.3. ANÁLISE ARQUITETURA DE DETEÇÃO DE POSE

4.3.1. DETEÇÃO DO OBJETO

A deteção inicial do objeto é feita utilizando um classificador em cascata (LBP) que identifica a localização da ROI que inclui o objeto, como demonstrado na Figura 128. O tempo computacional médio deste classificador numa *frame* de 1280x720 é de 59 ms. Por forma a tentar obter um tempo computacional inferior foi utilizado a interface CUDA existente na biblioteca OpenCV para aplicar este classificador, para uma frame de 1280x720 o tempo computacional médio é de 151 ms. Este aumento temporal é facilmente explicado pela pouca otimização da interface (está a dar os seus primeiros “passos”) e o grande atraso que existe entre a transferência de dados entre o GPU e a CPU – devido à largura de banda disponível.

Depois de obter a ROI, foi aplicado o algoritmo FAST à *frame* e os *keypoints* que estão dentro dessa ROI são utilizados para calcular a BB do objeto. O tempo computacional médio do algoritmo FAST numa *frame* de 1280x720 é inferior a 1 ms.

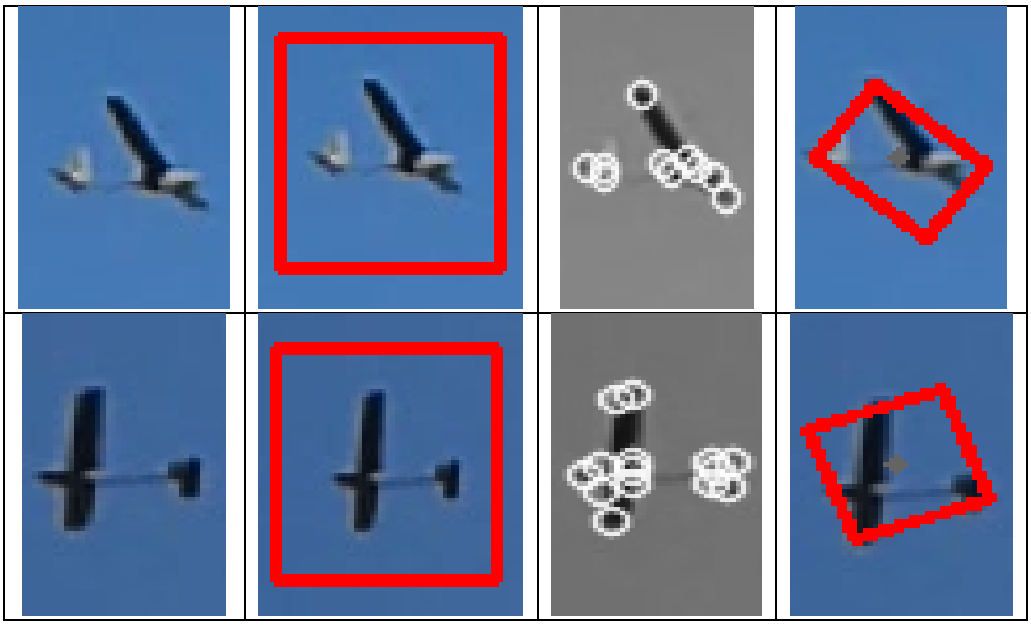


Figura 128 – Deteção da BB numa *frame* Real. Da esquerda para a direita: (i) Imagem original; (ii) ROI obtida pelo Classificador LBP; (iii) *Keypoints* obtidos através do algoritmo FAST; (iv) BB orientada obtida.

4.3.2. INICIALIZAÇÃO DAS PARTÍCULAS

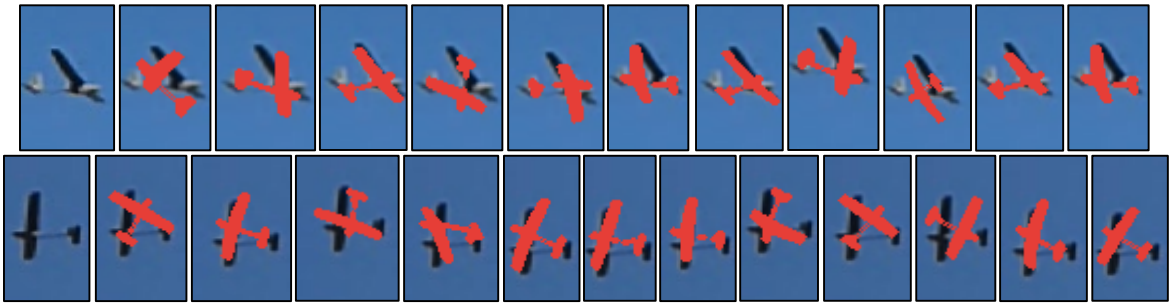


Figura 129 – Projeção na imagem real (vermelho) de possibilidades para a inicialização – Base de dados.

Para a BB obtida o ângulo e o *aspect ratio* foram calculados e comparados com a base de dados, como explicado no Subcapítulo 3.2.2. Como esta base de dados é carregada para a memória no início do programa, o tempo computacional médio desta comparação é inferior a 1 ms.

As melhores possibilidades (Figura 129) são usadas para iniciar a fase de otimização de pose. Existem algumas possibilidades com a mesma relação de BB e respetivo ângulo, que não correspondem à pose real do UAV, mas essas hipóteses vão ser filtradas na

primeira iteração. Esta filtragem é feita substituindo todas as partículas com um valor de *likelihood* muito baixo por partículas com valor elevado adicionando algum ruído gaussiano para garantir diversidade.

4.3.3. OTIMIZAÇÃO DAS PARTÍCULAS

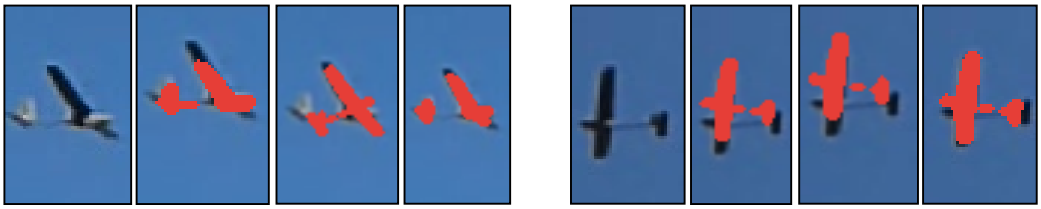


Figura 130 – Melhores partículas obtidas (3 primeiras iterações) para duas poses diferentes (Partículas = 100).

O método descrito no capítulo III – subcapítulo 3.2 - para deteção de pose foi aplicado em cenários reais (Figura 130). Através de testes práticos chegou-se à conclusão que 4 iterações com 100 partículas é o suficiente para obter uma boa aproximação da pose apresentada pelo objeto. Isto é apenas possível pois o procedimento de inicialização proporciona boas aproximações para a pose do objeto, aumentando assim a taxa de convergência da fase de otimização.

A *likelihood* para o cenário real da Figura 130 foi calculado (Gráfico 6 e Gráfico 7) fazendo variar o número de partículas (N) e o número de poses da base de dados (D) utilizadas na inicialização. Quando o valor de D é inferior a N as restantes possibilidades são criadas acrescentado ruído gaussiano às melhores possibilidades obtidas. Nos gráficos as duas linhas horizontais correspondem ao limiar selecionado (linha superior) e ao limiar mínimo (linha inferior) que controlam a otimização grosseira e fina do algoritmo de otimização de pose como representado na Figura 109. Abaixo do limiar mínimo estamos na fase de inicialização (melhores possibilidades da base de dados e algum ruído gaussiano dependendo dos parâmetros selecionados), no meio estamos na fase de otimização grosseira e acima do limiar estamos na fase de otimização fina.

Pela análise dos Gráfico 6 e Gráfico 7 é possível verificar que a convergência para uma estimativa aceitável/próxima da realidade (valor de *likelihood* superior ao valor de limiar – neste caso obtido experimentalmente e definido como 0,25 – conforme descrito na Figura 109) é bastante rápida (em média 2 iterações), demonstrando a importância do processo de inicialização na convergência para o resultado final. É também demonstrado que mais possibilidades provenientes da base de dados não corresponde obrigatoriamente melhores

resultados, e.g. 200 partículas utilizadas no Gráfico 7 para 150 e 200 possibilidades da base de dados. O valor de limiar também tem de ser ajustado para um valor que corresponda a um valor de *likelihood* aceitável para a pose do objeto, sendo um compromisso entre velocidade e precisão. A melhor relação entre velocidade e precisão foi obtida para 100 partículas utilizando 100 poses provenientes da base de dados para inicialização.

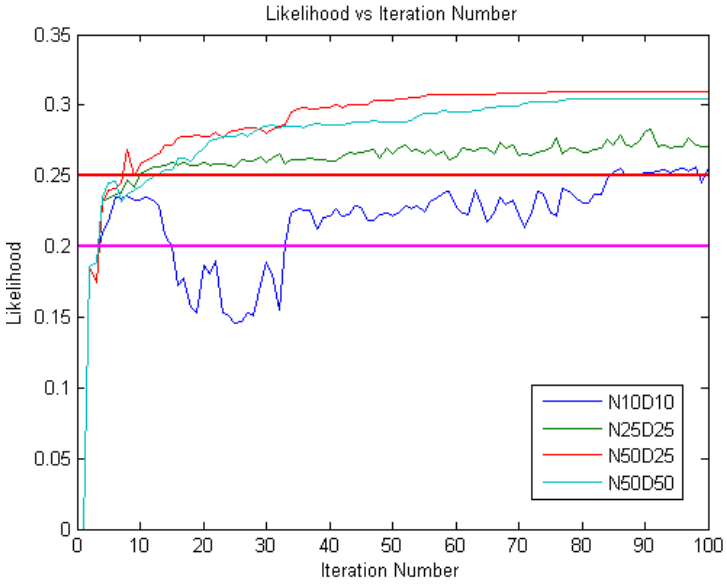


Gráfico 6 - *Likelihood* Média VS Número da iteração – Exemplo I.

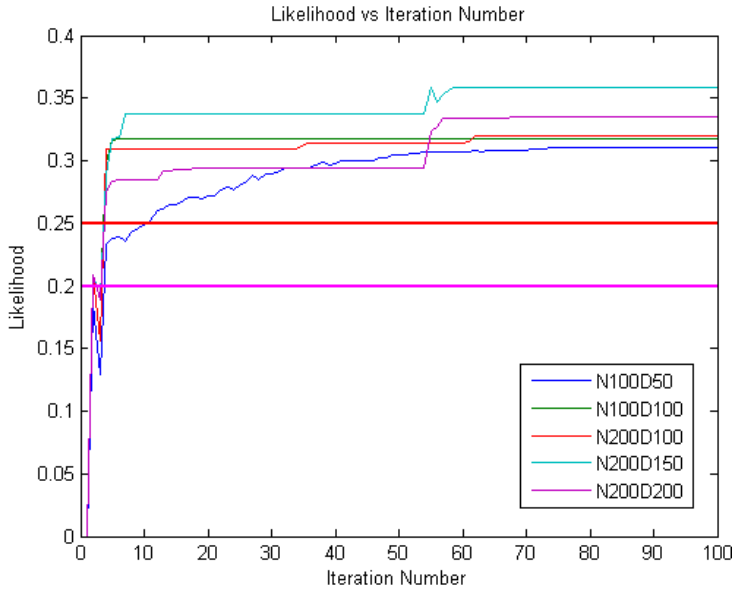


Gráfico 7 - *Likelihood* Média VS Número da iteração – Exemplo II.

Número de Partículas (N):	Tempo (ms):	*Frames* por segundo (fps):
1	7,5	133,3
10	72	13,8
20	148	6,7
50	364	2,7
100	710	1,4
200	1380	0,7

Tabela 12 – Tempo de execução (ms).

O tempo de execução médio para cada iteração é representado na Tabela 12. Para calcular a magnitude e ângulo do gradiente em cada *frame* (utilizado para a função de *likelihood* baseada em contornos) é necessário adicionar um tempo de computação médio de 25 ms. Ou seja, para 100 partículas, obtemos um tempo médio de computação por iteração de 790 ms (aproximadamente 1,26 fps).

4.3.4. AVALIAÇÃO DO DESEMPENHO QUANTITATIVO

Foi criado um conjunto de teste para diferentes distâncias do UAV: 5, 10, 15, 20, 25, 30, 35, 40, 45 e 50 metros, usando 850 imagens sintéticas (Figura 131) para cada distância. A pose final estimada foi obtida utilizando a partícula com maior peso ao correr o algoritmo proposto com 100 partículas e 100 poses provenientes da base de dados para inicialização.

Utilizando a função de *likelihood* baseada em textura para testes em todas as distâncias podemos verificar pela análise do Gráfico 8 que o erro de translação decresce com a distância obtendo um valor médio aos 5 metros (como representado na Tabela 13) de 0,33 metros. A área de aterragem é uma área irregular de 5x6 metros, logo precisamos de garantir um erro mínimo de translação de 1 metro para garantir a fiabilidade do sistema no momento da aterragem. Como estamos a operar num ambiente exterior com uma plataforma móvel, variações súbitas nas condições atmosféricas podem levar a falha. A resolução em translação obtida na gama de distâncias estudada é claramente suficiente para garantir uma boa estimativa da posição do UAV para aterragem. O número de *outliers* obtidos (corresponde a menos de 5% dos casos) pode ser claramente reduzido ao utilizar e.g. filtragem temporal.

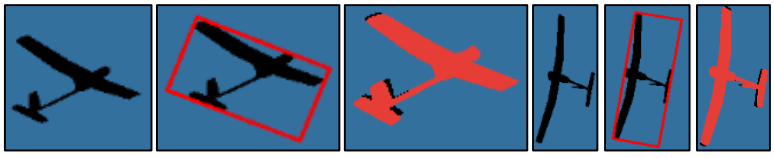


Figura 131 – Exemplos de *frames* de imagens sintéticas criadas para teste quantitativo.

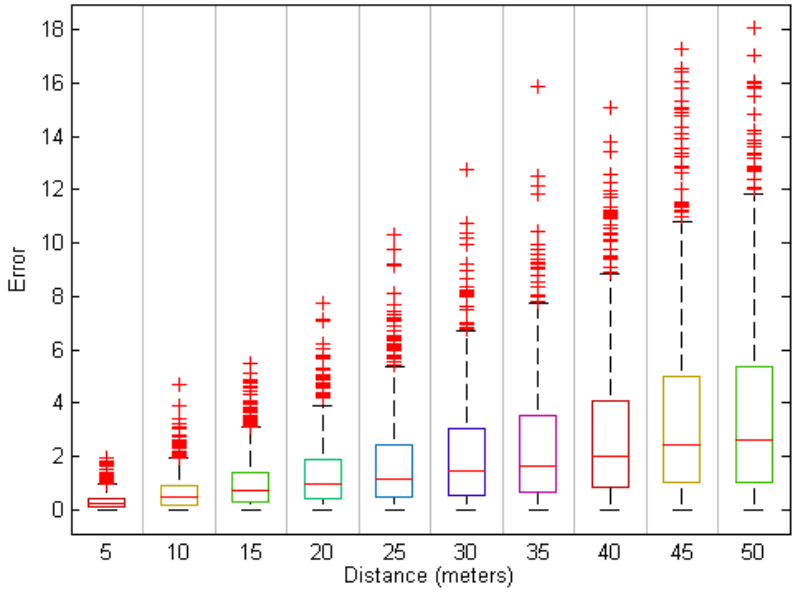


Gráfico 8 - Erro em translação – *likelihood* textura (metros).

Distância (metros):	Valor médio (metros):	Mediana (metros):
5	0,33	0,22
10	0,65	0,46
15	1,00	0,73
20	1,30	0,94
25	1,73	1,17
30	2,07	1,45
35	2,35	1,66
40	2,88	1,98
45	3,45	2,42
50	3,69	2,59

Tabela 13 – Erro de translação – *likelihood* textura (metros).

O erro de rotação também decresce com a proximidade mas menos que o obtido no caso da translação. Tem um valor mediano de 32,4 graus (Tabela 14). Este erro acontece maioritariamente porque o UAV tem a grande maioria dos seus *pixels* situados nas asas (Figura 132) e a função de *likelihood* baseada em textura utilizada é baseada na maximização da diferença entre duas áreas de *pixels*, originando uma menor sensibilidade para variações que acontecem no resto do seu corpo. Esta função de *likelihood* apresenta uma grande robustez a variações de iluminação, mas essa robustez tem um preço que é o elevado erro em rotação obtido. Este erro fica claramente evidente em poses onde o corpo do UAV está parcialmente ocludido pelas suas asas, gerando algumas situações em que a partícula se encontra deslocada aproximadamente 180 graus da pose observada.

Para tentar resolver esta ambiguidade e melhorar o resultado obtido, foi desenvolvida uma função de *likelihood* hibrida cujos resultados obtidos serão abordados seguidamente.

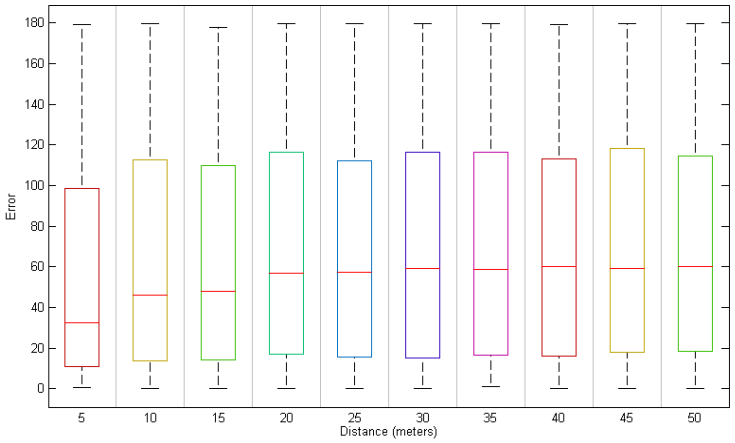


Gráfico 9 - Erro em rotação – likelihood textura (graus).

O erro de rotação é obtido de acordo com a seguinte expressão (Lee, McClamroch et al. 2006):

$$Erro_{Rotação} = \frac{1}{\sqrt{2}}\left\|R_{log}\right\|_F = \frac{1}{\sqrt{2}}\left\|log(R_1 \times R_2^T)\right\|_F \quad (4.2)$$

com R_1 a corresponder à matriz de rotação dos ângulos medidos e R_2' à matriz de rotação transposta dos ângulos de referência – imagem sintética gerada. A norma de Frobenius $\left\|R_{logm}\right\|_F$ é a norma da matriz $m \times n$, definida como a raiz quadrada da soma dos quadrados absolutos dos seus elementos da seguinte forma (Golub and Van Loan 1996):

$$\left\|R_{logm}\right\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|a_{ij}\right|^2} \quad (4.3)$$

Distância (metros):	Valor médio (graus):	Mediana (graus):
5	56,7	32,4
10	64,7	46,1
15	63,5	47,9
20	68,6	57,1
25	66,7	57,5
30	68,1	59,1
35	68,5	58,8
40	68,3	60,1
45	69,7	59,3
50	69,0	60,1

Tabela 14 – Erro de Rotação – *likelihood* textura (graus).

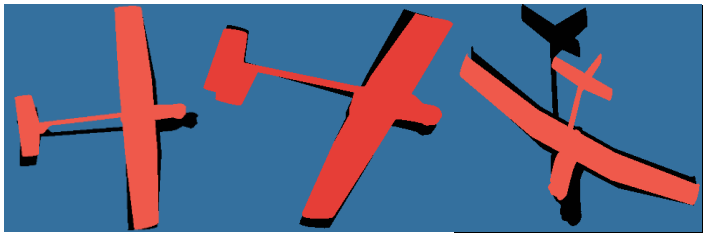


Figura 132 – Exemplo de erro obtido no uso da *likelihood* baseada em textura.

Através de testes práticos conseguiu-se chegar à conclusão que a utilização da função *likelihood* hibrida (descrita no Subcapítulo 3.2.3.) só faria sentido para distâncias inferiores a 25 metros. O erro de translação mais uma vez decresce com a distância (Gráfico 10) obtendo um valor médio aos 5 metros de 0,27 metros (Tabela 15), obtendo assim uma resolução superior ao obtido anteriormente ao utilizar apenas a função de *likelihood* baseada em textura.

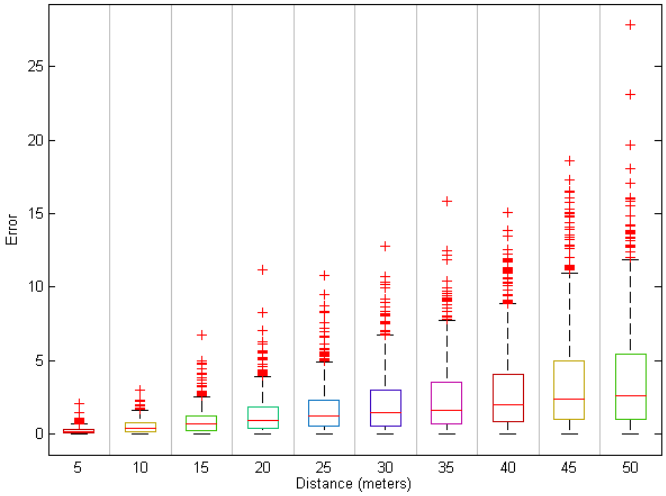


Gráfico 10 - Erro em translação – *likelihood* híbrida (metros).

Distância (metros):	Valor médio (metros):	Mediana (metros):
5	0,27	0,19
10	0,54	0,40
15	0,92	070
20	1,32	0,92
25	1,68	1,28
30	2,1	1,45
35	2,4	1,66
40	2,9	1,99
45	3,5	2,43
50	3,8	2,61

Tabela 15 – Erro de translação – *likelihood* híbrida (metros).

O erro em rotação aos 5 metros também decresce com a proximidade (Gráfico 11) mas mais uma vez menos acentuadamente que a translação. Tem um valor mediano de 9,4 graus (Tabela 16), sendo este valor obtido pela utilização de uma função de *likelihood* híbrida para valores de distância inferiores a 25 metros.

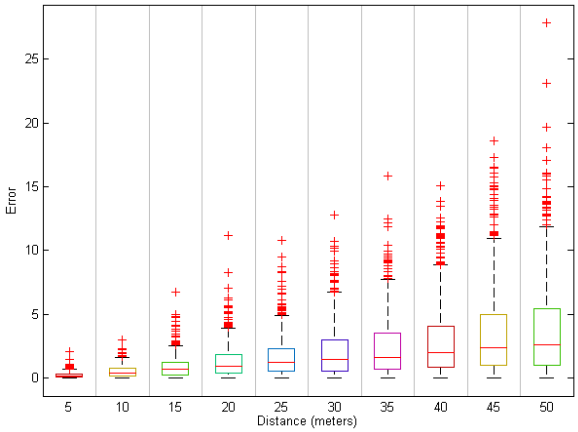


Gráfico 11 - Erro em rotação – *likelihood* híbrida (graus).

Distância (metros):	Valor médio (graus):	Mediana (graus):
5	37,2	9,4
10	52,3	14,6
15	60,5	22,1
20	62,5	23,3
25	63,3	27,0
30	75,4	52,2
35	75,1	46,6
40	77,2	56,5
45	74,9	41,8
50	79,3	61,7

Tabela 16 – Erro de Rotação – *likelihood* híbrida (graus).

Distância (metros)	Histograma interior	Histograma exterior
5	26365	81963
10	6451,2	20159
15	2834,7	8903,7
20	1613,4	5008,6
25	1027,1	3179,6
30	715,3	2197,3
35	526,6	1613,3
40	399,5	1208,9
45	317,6	950,6
50	257,8	762,1

Tabela 17 – Número médio de *pixels* obtidos – *likelihood* textura.

No entanto os erros obtidos por ambiguidades (partícula deslocada aproximadamente 180 graus) mantêm-se, devendo recorrer a filtragem temporal para eliminar estas ambiguidades.

A Tabela 17 representa o número médio de *pixels* obtidos no histograma interior e exterior em função da distância, ao utilizar a *likelihood* de textura. Quanto mais *pixels* existem nomeadamente na zona das asas, e como referido anteriormente, maior vai ser o erro obtido em rotação daí a utilização da função de *likelihood* híbrida.

4.4. ANÁLISE ARQUITETURA DE *TRACKING*

4.4.1. REAMOSTRAGEM

Para dar diversidade às possibilidades geradas e evitar mínimos locais, foi explorada a possibilidade de utilizar a base de dados de inicialização (conforme descrita no subcapítulo 3.2.2) em cada *frame* sem a estratégia de otimização local desenvolvida. Seria então reservado um número n de partículas em cada iteração que seriam provenientes deste método. Para verificar o seu desempenho foram geradas imagens sintéticas conforme representado na Figura 132.

Figura 133 – Exemplo da melhor partícula (vermelho) obtida por inicialização – base de dados.

Foi gerada uma sequência de imagens em que o UAV tinha uma variação de 0,01 metros em Z entre *frames*, variava a sua posição em X e Y de acordo com uma distribuição normal de média nula e desvio padrão de 0,05 metros e variava a sua atitude em α, β e γ de acordo com uma distribuição normal de média nula e de desvio padrão de 25 graus. A posição inicial do UAV na *frame* corresponde a: $Z = 65, X = 0, Y = 0, \alpha = 0, \beta = 0$ e $\gamma = 180$. Foi assim criado um conjunto de teste de 6201 imagens sintéticas (Figura 133), correspondendo a uma sequência de aterragem de acordo com os parâmetros descritos anteriormente. Utilizando somente a função de *likelihood* baseada em textura para testes

em todas as distâncias verifica-se que tendo em conta os erros de rotação (valor médio de 88,8 graus e mediana de 87,5 graus) de e translação (valor médio de 16,5 m e mediana de 13,2 metros) obtidos a melhor escolha será usar 5 possibilidades provenientes da base de dados. Estes erros são mais elevados do que os descritos no subcapítulo 4.3.4 pois após o processo de inicialização ainda é feito uma otimização local baseada em três fases (subcapítulo 3.2.4) o que possibilita a obtenção de melhores resultados. E nem sempre a melhor partícula (maior peso) naquela iteração está situada na zona em que se encontra a solução, principalmente em distribuições multimodais.

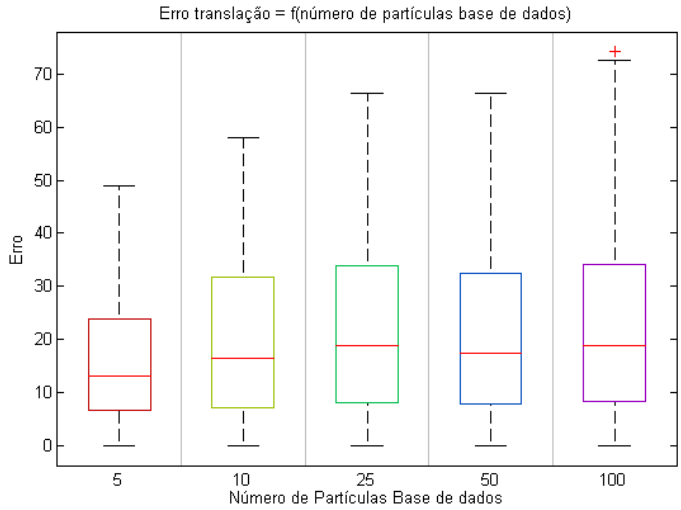


Gráfico 12 - Erro em translação (metros) – *likelihood* textura (metros).

Número de possibilidades (base de dados):	Valor médio (metros):	Mediana (metros):
100	21,9	18,92
50	20,8	17,6
25	21,6	18,9
10	20,1	16,6
5	16,5	13,2

Tabela 18 – Erro em translação (metros) inicialização – *likelihood* textura (metros).

Número de possibilidades (base de dados):	Valor médio (graus):	Mediana (graus):
100	111,6	116,4
50	109,3	113,4
25	103,9	107,9
10	93,7	93,5
5	88,8	87,5

Tabela 19 – Erro em rotação (graus) inicialização – *likelihood* textura (metros).

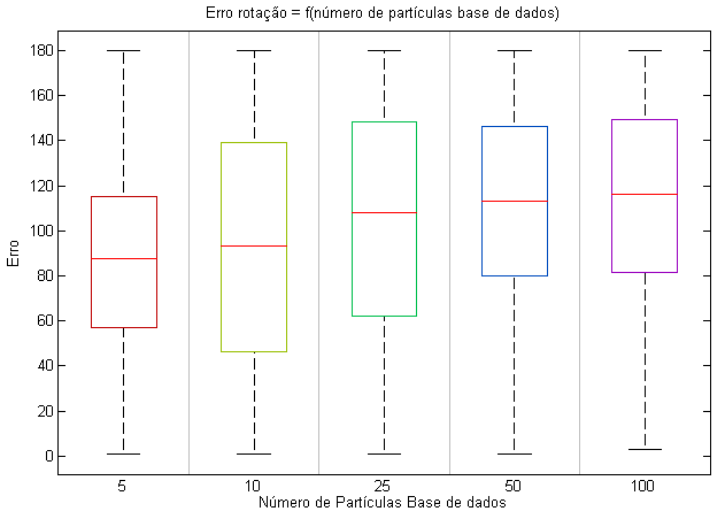


Gráfico 13 - Erro em rotação (graus) – *likelihood* textura (metros).

Utilizando a função de *likelihood* híbrida (Figura 134) para esta inicialização, e utilizando 100 partículas para esta inicialização temos que a média de erro de translação é de sensivelmente 21,2 metros e o valor médio do erro de rotação é de 98,41 graus.

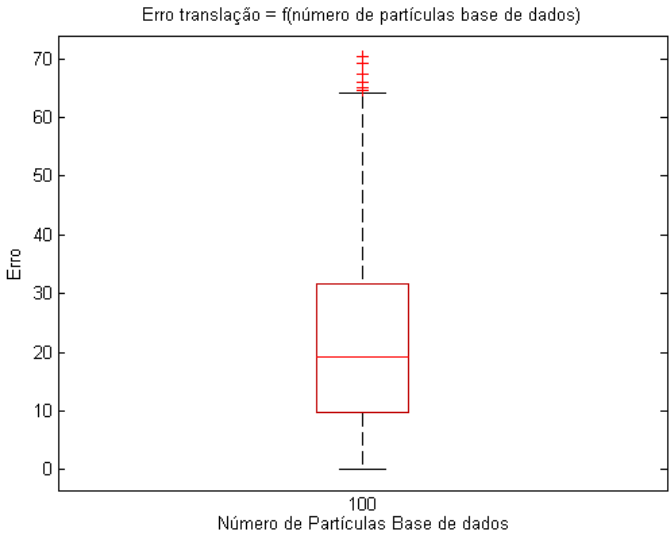


Gráfico 14 - Erro em rotação (graus) – *likelihood* híbrida (metros).

Figura 134 – Exemplo de filtro de Sobel - Imagem sintética da sequência de aterragem.

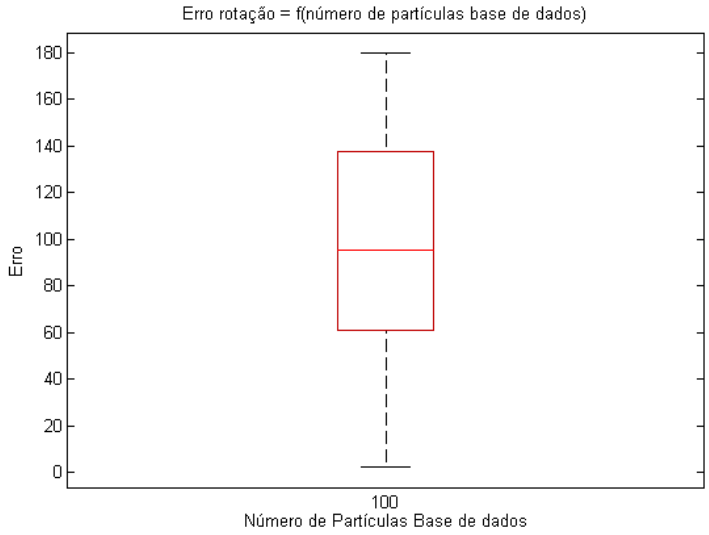


Gráfico 15 - Erro em rotação (graus) – *likelihood* híbrida (metros).

É possível também verificar que utilizar mais partículas da base de dados nem sempre apresenta melhores resultados, devido a existirem poses com pesos muito próximos (distribuição multimodal) o que faz com que escolhendo apenas a melhor o resultado não seja satisfatório. Deve-se optar por métodos de otimização local em cada iteração caso se pretenda utilizar este método de reamostragem, por forma a obter bons resultados.

4.4.2. AVALIAÇÃO DO DESEMPENHO

Para a análise do desempenho foram criadas sequências de vídeo sintético em que foi aplicado o algoritmo desenvolvido para a deteção de pose (subcapítulo 3.2.), em que a melhor partícula obtida foi filtrada entre *frames* recorrendo ao filtro de Kalman *Unscented* (subcapítulo 3.3.3.) procurando assim minimizar o erro obtido na estimativa de pose (posição 3D e atitude).

Para esta análise foram criadas duas sequências de vídeo sintético distintas, procurando assim quantificar os diferentes erros obtidos, quer em rotação e em translação entre a realidade, a medição e estimativa obtida através da filtragem.

Na primeira sequência criada o UAV é inicializado com o vetor de estado, de acordo com o descrito na Tabela 20, mantendo a aproximação com pose constante e a uma velocidade de 9,45 m/s (conforme descrito no subcapítulo 3.3.). Toda a parte do processamento do

filtro de partículas é feita em ângulos de Euler (devido às funções utilizadas recorrendo à biblioteca OpenGL) conforme descrito no subcapítulo 3.2., sendo a melhor partícula convertida para quaternião (subcapítulo 2.2.) e a seguir filtrada utilizando um UKF.

Vetor de Estado:	Valor:
X	0 m
Y	0 m
Z	25 m
Alfa	0 graus
Beta	0 graus
Gama	190 graus

Tabela 20 – Vetor de estado inicial – Sequência de vídeo sintético 1.

Como é possível ver pela análise do Gráfico 16, Gráfico 17 e Gráfico 18 (Posição em X,Y e Z respetivamente) existe algum erro de medição sendo a trajetória do UAV suavizada através da utilização de filtragem temporal. Permitindo assim uma melhor estimativa da posição do UAV, do que utilizando diretamente a medição efetuada. Faz todo o sentido a utilização de filtragem temporal, tendo em conta que a arquitetura de deteção de pose implementada possui erros de medição e estes podem ser atenuados com a devida filtragem. Os erros obtidos nesta sequência encontram-se quantificados na Tabela 21.

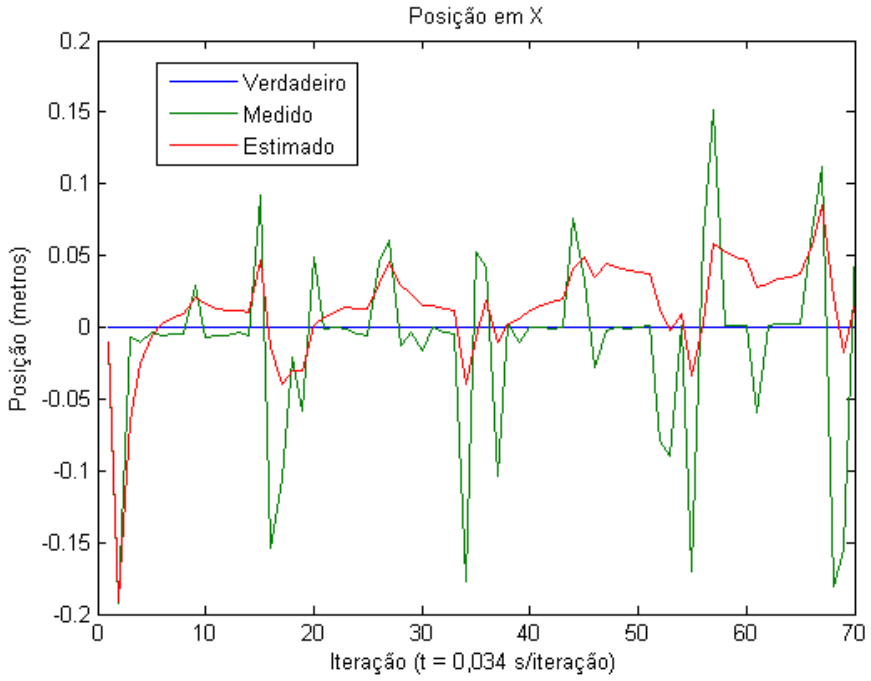


Gráfico 16 – Posição em X – Verdadeira, Medição e Estimada (Sequência I).

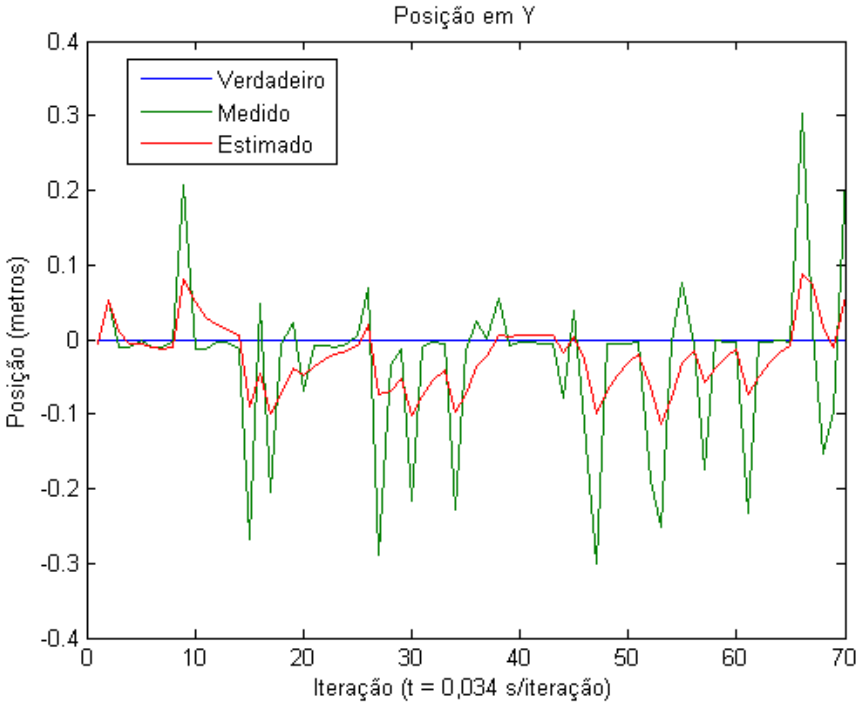


Gráfico 17 – Posição em Y – Verdadeira, Medição e Estimada (Sequência I).

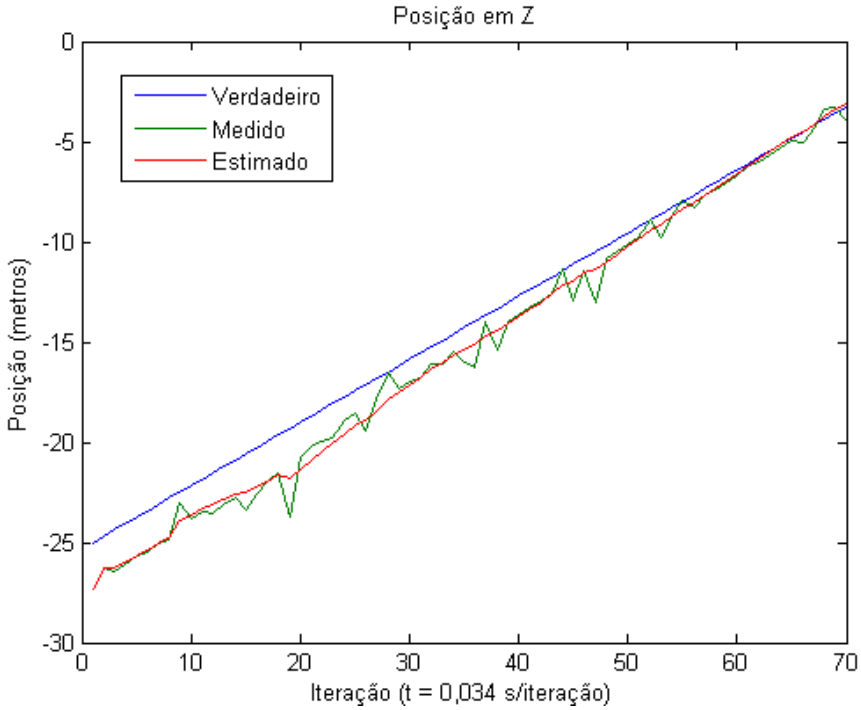


Gráfico 18 – Posição em Z – Verdadeira, Medição e Estimada (Sequência I).

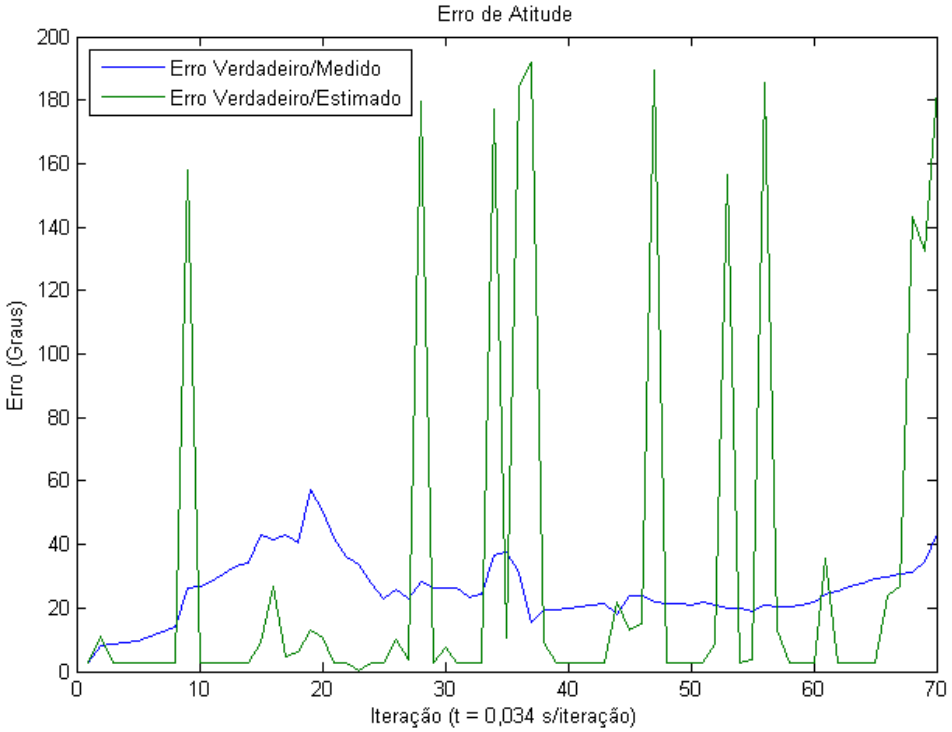


Gráfico 19 – Erro de atitude (Graus) (Sequência I).

	Medição		Sequência 1 (Estimativa)	
	Média	**Mediana**	**Média**	**Mediana**
X (m):	0,012	0,002	-0,012	-0,013
Y (m):	0,029	0,007	0,023	0,019
Z (m):	1,149	1,069	1,099	1,096
Erro de atitude verdadeiro / medido (graus)	32,39	2,89		
Erro de atitude verdadeiro / Estimado (graus)			24,87	23,91

Tabela 21 – Quantificação dos erros obtidos – Sequência 1.

No que concerne à estimação de atitude, é calculado o quaternião erro entre o quaternião correspondente ao erro entre a realidade e o medido e o quaternião erro correspondente ao erro entre a realidade e a atitude estimada (conforme descrito no subcapítulo 3.3.3. – Filtragem – Filtro de Kalman *Unscented*). Sendo que o ângulo – θ_{erro} – que permite criar uma rotação na outra (erro) a ser calculado da seguinte forma (Kraft 2003, Cheon and Kim 2007):

$$\theta_{erro} = 2\arccos(q_{erro_w}) \tag{4.4}$$

em que q_{erro_w} corresponde à parte escalar do quaternião erro calculado. Verificando-se um melhor comportamento (Gráfico 19) no caso em que existe redundância de atitude (mais 180 graus). Que era uma das grandes limitações verificadas, no cálculo da deteção de pose (subcapítulo 4.3.4.) em que não era utilizada qualquer informação temporal.

Para uma melhor análise das vantagens de utilização de filtragem temporal, a pose foi feita variar de acordo com duas distribuições normais, uma para o parâmetro X e Y ($\mathcal{N}(0,0{,}25)$) e outra para os ângulos alfa, beta e gama ($\mathcal{N}(0,5)$). O parâmetro Z manteve os valores iniciais e variação estabelecidos no teste anterior.

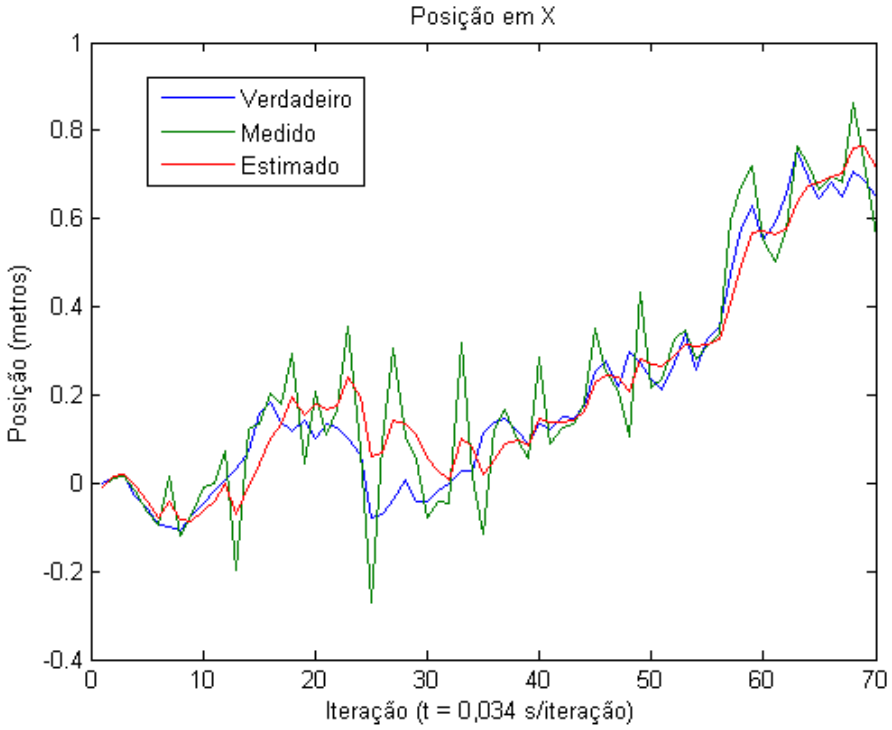


Gráfico 20 – Posição em X – Verdadeira, Medição e Estimada (Sequência II).

	Medição		**Sequência 2 (Estimativa)**	
	Média	**Mediana**	**Média**	**Mediana**
X (m):	- 0,021	- 0,009	-0,011	-0,009
Y (m):	0,026	0,018	0,037	0,034
Z (m):	1,015	0,621	0,904	0,212
Erro de atitude verdadeiro / medido (graus)	72,79	23,30		
Erro de atitude verdadeiro / Estimado (graus)			31,94	29,54

Tabela 22 – Quantificação dos erros obtidos – Sequência 2.

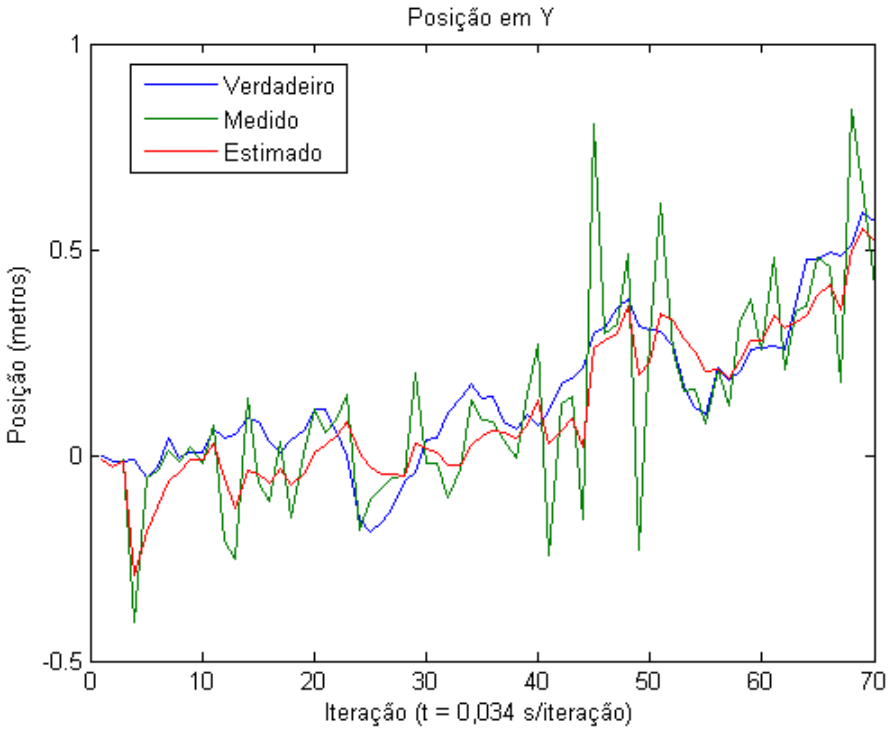


Gráfico 21 – Posição em X – Verdadeira, Medição e Estimada (Sequência II).

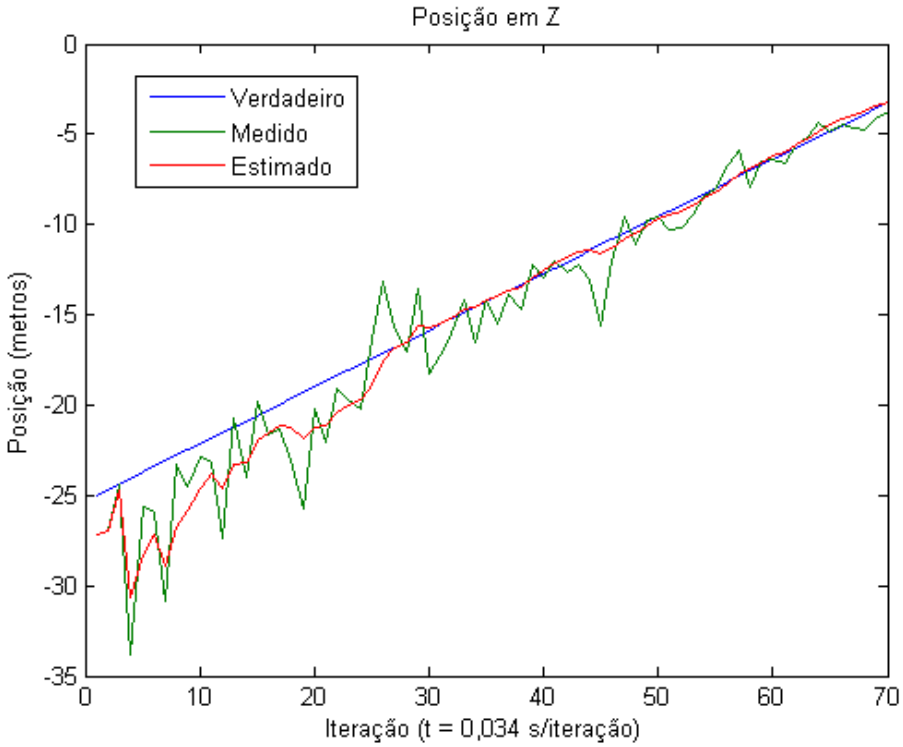


Gráfico 22 – Posição em Z – Verdadeira, Medição e Estimada (Sequência II).

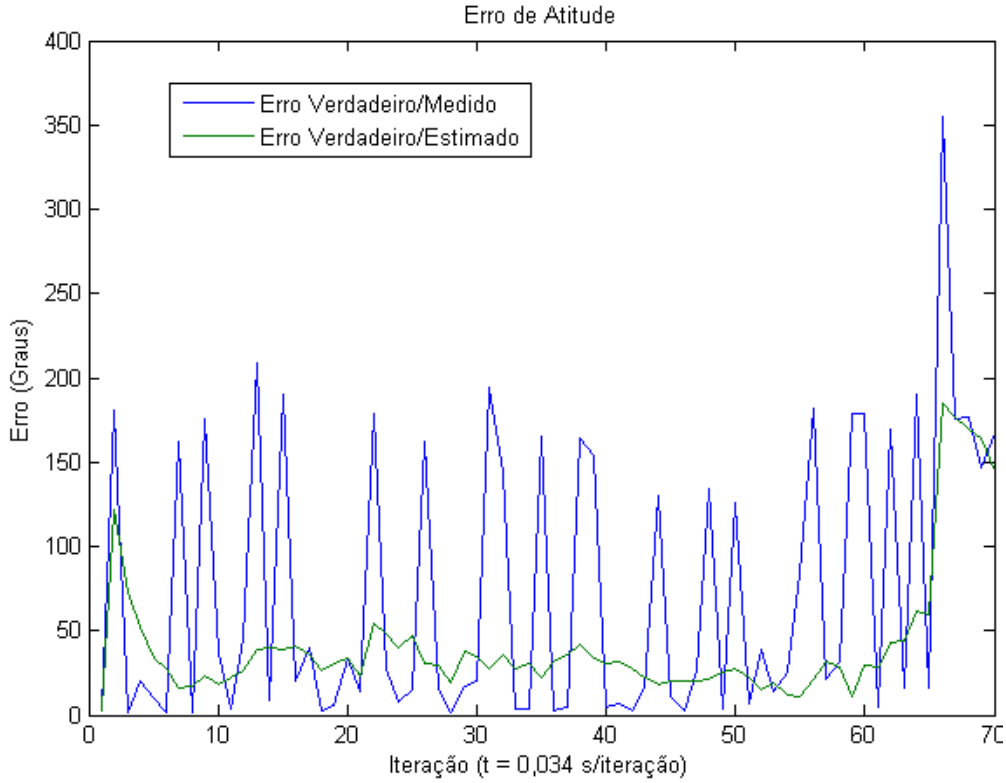


Gráfico 23 – Erro de atitude (Graus) (Sequência II).

No caso da posição em X,Y e Z (Gráfico 20, Gráfico 21 e Gráfico 22), nota-se um suavizar evidente da trajetória do UAV através da utilização da filtragem temporal, mais evidente agora devido à variação de X e Y mais acentuada entre iterações. No caso da análise de atitude (Gráfico 23) existe claramente uma diminuição do erro na estimativa face ao medido em que se verificam bastantes iterações em que a pose oposta (≅ 180 graus) ou poses próximas são selecionadas de forma errada. Neste caso concreto ainda não foram introduzidas restrições de estado, baseado e.g. em considerações físicas do objeto o que melhoraria a estimativa ao remover *outliers* na medição. Estas restrições ainda não foram introduzidos devido à falta de testes práticos, em que a dinâmica do UAV pode ser analisada e devidamente caracterizada para a introdução destas restrições. Os erros obtidos nesta sequência encontram-se quantificados na Tabela 22.

Um exemplo de *frame* sintética pode ser visto na Figura 135, em que se optou por colocar o céu a azul e o UAV em preto, sendo que a função de *likelihood* utilizada neste teste foi a baseada em textura. Esta foi selecionada pois era a que apresentava o pior desempenho nas distâncias de 25 metros e inferiores, sendo que as vantagens de utilização da filtragem temporal seriam mais evidentes.

Figura 135 – Exemplo de *frame* sintética criada.

4.5. CONCLUSÕES OU SÍNTESE

Este capítulo faz uma abordagem aos resultados experimentais obtidos, por forma a aferir o desempenho qualitativo e quantitativo do sistema. A inexistência de sequências de imagens reais com *ground truth* não possibilitou a análise do erro de rotação e translação tendo em conta imagens reais, sendo que para contornar essa situação foram utilizadas imagens geradas de forma sintética o mais aproximado da realidade possível para análise da arquitetura de deteção de pose e de *tracking*.

Foram exploradas estratégias de reamostragem, como a utilização da base de dados em cada iteração sem qualquer outro auxílio de otimização procurando assim quantificar o seu desempenho como estratégia de reamostragem. Esta estratégia de reamostragem mostrou por si só um fraco desempenho devendo, em caso de utilização, ser utilizado para dar diversidade às possibilidades testadas e não como origem principal de possibilidades a testar.

Foi explorada a filtragem temporal como complemento da arquitetura de deteção de pose desenvolvida, procurando assim utilizar um método de filtragem temporal em que fossem obtidos melhores resultados. Esta filtragem deve ser testada com sequências reais, procurando assim aferir corretamente os parâmetros de filtragem para obter o desempenho máximo.

Não houve uma preocupação com o tempo de processamento na análise da filtragem, pois esse processamento não envolve qualquer projeção de pose do UAV sendo puramente multiplicação de quaterniões, vetores e matrizes (tempo de processamento medido para cada iteração aproximadamente zero – ≅ 0,049 ms).

CAPÍTULO V
CONCLUSÕES E RECOMENDAÇÕES

5.1. SÍNTESE FINAL

Nesta dissertação de mestrado foi proposta a elaboração de um sistema de visão para aterragem automática de UAV. Assim, durante esta dissertação, verificou-se o estado da arte no que concerne ao modelo geométrico da câmara, à visão estereoscópica, ao classificador utilizado, deteção de cantos, métodos de deteção baseada em modelos 3D, filtragem temporal (filtro de Kalman *unscented*), arquitetura CUDA (GPU) e os sistemas de aterragem existentes atualmente de forma a compreender os conceitos teóricos subjacentes à elaboração deste trabalho.

Foram também descritas as arquiteturas implementadas de deteção de pose e tracking, descrevendo de forma pormenorizada cada um dos seus constituintes. Com vista a comprovar as hipóteses colocadas durante a descrição da metodologia utilizada, foram efetuados variadíssimos testes, após apreendidos os fundamentos teóricos essenciais para o fazer.

5.2. HIPÓTESES E OBJECTIVOS CUMPRIDOS

Os objetivos e hipóteses apresentadas no subcapítulo 1.3.1 são analisados neste subcapítulo, pois foi com esse objetivo que se desenvolveu este trabalho.

Além do descrito em todos os subcapítulos – Conclusões ou síntese – presentes em cada capítulo desta dissertação torna-se necessário efetuar uma reflexão final sobre os resultados obtidos.

A câmara a utilizar neste caso concreto pode ser uma câmara do espectro do visível, pois consegue-se um bom contraste entre o céu e o UAV, obtendo assim uma boa discriminação recorrendo a funções de *likelihood* criadas para o efeito. Para possibilitar a colocação da estação de comando em terra longe da câmara e principalmente para garantir IP67 (essencial a uma câmara destinada a operar no exterior e especialmente em ambiente marítimo) foi utilizada uma câmara com protocolo *Gigabit Ethernet*.

Devido ao baixo poder de processamento disponível na aeronave, o melhor sistema a utilizar será situado no navio. Sendo que em caso de possibilidade o melhor seria que o UAV o pudesse também fazer, aumentando assim a fiabilidade do sistema como um todo.

De acordo com os testes efetuados a imagem capturada pela câmara não deverá ter uma resolução muito elevada, pois quanto maior a resolução maior será o número de *pixels* que necessitamos de processar o que nesta aplicação em concreto em que o tempo de processamento é uma limitação não é aconselhado. A resolução que permite uma deteção e estimativa eficaz a pelo menos 50 metros é a de 1 Megapixels, garantindo assim um compromisso entre tempo de processamento e capacidade de deteção e estimativa.

A *frame rate* neste caso de estudo não se encontra limitada pelo *hardware* mas sim pelo algoritmo desenvolvido, em que a necessidade de projeção do objeto em múltiplas poses (partículas) torna o algoritmo computacionalmente pesado. Para tal, o cálculo da *likelihood* deve ser feito na GPU conforme sugerido no subcapítulo 5.3.

A tolerância de erro em translação aceitável para a área de aterragem é de aproximadamente 1 metro, tendo obtido através da arquitetura de deteção de pose e utilizando a fórmula de *likelihood* híbrida a 5 metros uma mediana de erro de translação de 0,19 metros e de rotação de 9,4 graus.

O UAV utilizado garante as dimensões necessárias para que possa ser estimado com sucesso a uma distância de pelo menos 50 metros, um dos objetivos iniciais. Sendo assim detetado com diferentes translações, rotações e intensidade luminosa através da arquitetura desenvolvida.

Foi introduzido e testado um método para estimação de pose de um UAV conhecido baseado num classificador em cascata, uma nova metodologia de inicialização para o filtro de partículas recorrendo a uma base de dados pré-treinada de poses e um processo de otimização local baseada nos métodos de evolução presentes nos algoritmos genéticos. A combinação de algoritmos genéticos com filtros de partículas demonstrou propriedades de convergência interessantes e que devem continuar a ser explorados. Foi também

introduzida filtragem temporal entre *frames*, procurando assim utilizar a informação da pose obtida na *frame* anterior para estimar a pose atual minimizando assim as diferenças que existem por poderem existir erros elevados de estimativa de pose entre *frames*.

A correção da distorção radial e tangencial é efetuada totalmente na GPU obtendo um menor tempo de processamento devido à enorme capacidade de paralelismo existente, diminuindo assim também a carga existente na CPU que pode ser utilizada para outras operações de forma assíncrona enquanto a GPU processa as imagens capturadas através da câmara utilizada.

5.3. RECOMENDAÇÕES E SUGESTÕES

Neste subcapítulo vão ser apresentadas algumas ideias para trabalho futuro nesta área, sendo a maior parte delas baseadas na resolução de certos problemas encontrados e otimização de processos.

As funções de *likelihood* estudadas e desenvolvidas têm parâmetros que têm de ser otimizados tendo em conta imagens reais, considerando mesmo parâmetros adaptativos consoante a distância do objeto a detetar. Outras funções de *likelihood* devem mesmo ser estudadas, procurando otimizações de desempenho.

A arquitetura de otimização de pose é baseada numa estratégia de três fases de otimização distintas baseadas em dois valores de limiar, que fazem com que exista uma variação ao processo de reamostragem utilizado. Estes limiares de deteção atualmente são fixos mas deve ser considerada a sua variação consoante a distância ao objeto a detetar.

O método de inicialização desenvolvido recorrendo a uma base de dados de poses apresentou resultados bastante satisfatórios, no entanto outros métodos devem ser abordados por forma a efetuar um correta comparação de desempenho. Um exemplo de inicialização poderá ser uma base de dados de máscaras de contornos, procurando assim uma optimização de processo.

O cálculo da *likelihood* da partícula deverá ser totalmente feito na GPU e não apenas o *rendering* do objeto, pois o atraso de transferência entre a GPU e a CPU torna a implementação em tempo real muito difícil. Foi iniciado este processo, tendo sidos registados os tempo de operação e de todas as transferências de dados entre dispositivos

(CPU e GPU) apenas não foi implementado o cálculo final utilizando a função de *likelihood* utilizada.

Implementar um filtro de partículas *unscented* adaptado individual para as melhores partículas encontradas, procurando assim através de uma estrutura em árvore garantir que em cada iteração obtemos sempre filtragem temporal de várias partículas possibilitando assim uma melhor estimativa.

Neste momento apenas foi estudada a implementação e funcionamento de um filtro UPF adaptado recorrendo a imagens sintéticas, ajustando assim os seus parâmetros de funcionamento a imagens geradas de forma sintética. Devem ser obtidas imagens reais por forma a conseguir estimar corretamente os parâmetros a utilizar em caso real.

Por forma a não perder o avião devido ao balanço da embarcação (factor este que não foi contemplado nesta dissertação) deve-se acoplar a câmara a uma plataforma que se possa movimentar (*Pan-Tilt unit*) mas que ao mesmo tempo faça a medição da sua orientação relativa por forma a não perder o UAV (sair da *frame*) e conseguir estimar a sua atitude.

5.4. LIMITAÇÕES E PROBLEMAS ENCONTRADOS

Sem dúvida uma das grandes limitações encontradas foi a falta de oportunidade de experimentar o sistema numa situação real, tendo falhado esse que seria um dos objetivos principais. Ao aplicar o sistema num caso real daria uma outra perspetiva das necessidades e problemas existentes, sendo que com certeza iriam surgir situações não contempladas teoricamente. A falta de legislação específica nesta área levou a que através da Autoridade Aeronáutica Nacional – AAN – fosse emitida a circular Nº1/13 (23 de Setembro de 2013) que visa estabelecer os requisitos e procedimentos para a emissão de licenças especiais de aeronavegabilidade – LEA – para sistemas de aeronaves não tripuladas no domínio da Defesa Nacional. O carácter experimental desta circular levou a que houvesse uma demora excessiva na implementação de requisitos, o que impossibilitou janela temporal para que testes reais fossem levados a cabo.

A arquitetura desenvolvida visa detetar um objeto assumindo que o cenário de fundo é o céu, para cenários distintos (com outro tipo de fundos mais heterogéneos) este algoritmo deve ser testado e devidamente adaptado em caso de necessidade.

Todo o processamento referente ao *rendering* e cálculo da *likelihood* do objeto deve ser feita na GPU e não na CPU, pois o grande atraso existente na transferência de dados entre a GPU e a CPU faz com que nesta aplicação concreta em que a velocidade de processamento é um requisito base – tempo real. Apesar desse atraso se situar nos milissegundos, ao aumentarmos o número de partículas por iteração este número vai-se tornar relevante no final.

(Página intencionalmente deixada em branco)

BIBLIOGRAFIA

Ahonen, T., A. Hadid and M. Pietikäinen (2004). Face recognition with local binary patterns. Computer vision-eccv 2004, Springer**:** 469-481.

Anderson, B. D. and J. B. Moore (2012). Optimal filtering, Courier Dover Publications.

Arulampalam, M. S., S. Maskell, N. Gordon and T. Clapp (2002). "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking." Signal Processing, IEEE Transactions on **50**(2): 174-188.

Azevedo, M. (2006). "Teses, relatórios e trabalhos escolares." Lisboa: Universidade Católica Editora.

Barrera, P., J. M. Cañas and V. Matellán (2005). Visual Object Tracking in 3D with Color Based Particle Filter. WEC (2).

Bell, G., J. Gray and A. Szalay (2006). Petascale Computational Systems. Computer, IEEE Computer Society. **39:** 110-112.

Bhat, K. S., S. M. Seitz, J. Popović and P. K. Khosla (2002). Computing the physical parameters of rigid-body motion from video. Computer Vision—ECCV 2002, Springer**:** 551-565.

Bigun, J. (2006). Vision with Direction: A Systematic Introduction to Image Processing and Computer Vision, Springer.

Boli, M., P. M. Djuri and S. Hong (2004). "Resampling algorithms for particle filters: a computational complexity perspective." EURASIP J. Appl. Signal Process. **2004**: 2267-2277.

Bradski, G. and A. Kaehler (2008). Learning OpenCV: Computer vision with the OpenCV library, " O'Reilly Media, Inc.".

Bradski, G. and A. Kaehler (2013). Learning OpenCV: Computer Vision in C++ with the OpenCV Library, O'Reilly Media, Inc.

Brandão, M., A. Bernardino and J. Santos-Victor (2011). Image Driven Generation of Pose Hypotheses for 3D Model-based Tracking. MVA.

Brown, J. A. (2010). GPU-Accelerated Particle Filtering for 3D Model-Based Visual Tracking. Master of Applied Science, University of New Brunswick, Fredericton, Canada.

Cagnoni, S., E. Lutton and G. Olague (2007). Genetic and evolutionary computation for image processing and analysis, Hindawi Publishing Corporation.

Carmi, A., S. J. Godsill and F. Septier (2009). Evolutionary MCMC particle filtering for target cluster tracking. Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop, 2009. DSP/SPE 2009. IEEE 13th, IEEE.

Cesetti, A., E. Frontoni, A. Mancini, P. Zingaretti and S. Longhi (2010). A vision-based guidance system for UAV navigation and safe landing using natural landmarks. Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009, Springer.

Cha, S.-H. and S. N. Srihari (2002). "On measuring the distance between histograms." Pattern Recognition **35**(6): 1355-1370.

Challa, S. (2011). Fundamentals of Object Tracking, Cambridge University Press.

Chandra, R. (2001). Parallel programming in OpenMP, Morgan Kaufmann.

Chang, P. and J. Krumm (1999). Object recognition with color cooccurrence histograms. Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on., IEEE.

Chapman, B., G. Jost and R. Van Der Pas (2008). Using OpenMP: portable shared memory parallel programming, MIT press.

Chaves, S. M., R. W. Wolcott and R. M. Eustice (2013). "NEEC Research: Toward GPS-denied Landing of Unmanned Aerial Vehicles on Ships at Sea."

Cheon, Y.-J. and J.-H. Kim (2007). Unscented filtering in a unit quaternion space for spacecraft attitude estimation. Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on, IEEE.

Choi, C. and H. I. Christensen (2011). Robust 3D visual tracking using particle filtering on the SE (3) group. Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE.

Cook, S. (2013). CUDA programming: a developer's guide to parallel computing with GPUs, Newnes.

Corke, P. (2011). Robotics, vision and control: fundamental algorithms in MATLAB, Springer.

Corporation, I. (2014). "Intel® Core™ i7-4700HQ Processor (6M Cache, up to 3.40 GHz)." Retrieved July,2014, 2014, from http://ark.intel.com/products/75116/Intel-Core-i7-4700HQ-Processor-6M-Cache-up-to-3_40-GHz.

Corporation, N. (2014). CUDA C Programming Guide NVIDIA Corporation**:** 241.

Cortes, B. (2005). "Sistemas de suporte à decisão." FCA-Editora Informática, Lisboa R Modelo de Gestão da Rota do Românico do Vale do Sousa| Rosário Correia Machado.

Crassidis, J. L. and F. L. Markley (2003). "Unscented filtering for spacecraft attitude estimation." Journal of guidance, control, and dynamics **26**(4): 536-542.

Cruz, j. E. C., e. h. shiguemori and L. N. F. Guimarães (2013) "Comparação entre HOG+SVM e Haar-like em cascata para a detecção de campos de futebol em imagens aéreas e orbitais." Simpósio brasileiro de sensoriamento remoto.

Cyganek, B. and J. P. Siebert (2011). An Introduction to 3D Computer Vision Techniques and Algorithms, Wiley.

Dalal, N. and B. Triggs (2005). Histograms of oriented gradients for human detection. Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, IEEE.

Davies, E. R. (2012). Computer and machine vision: theory, algorithms, practicalities, Academic Press.

Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision with OpenCV, John Wiley & Sons.

Dhindsa, K. S. and G. Babbar (2011). "Development of an Improved Feature based Algorithm for Image Matching." Development **14**(8).

Doucet, A., N. De Freitas and N. Gordon (2001). An introduction to sequential Monte Carlo methods. Sequential Monte Carlo methods in practice, Springer**:** 3-14.

Doucet, A., N. de Freitas and N. Gordon (2001). Sequential Monte Carlo Methods in Practice, Springer.

Doucet, A. and A. M. Johansen (2009). "A tutorial on particle filtering and smoothing: Fifteen years later." Handbook of Nonlinear Filtering **12**: 656-704.

Dua, S. and X. Du (2014). Data mining and machine learning in cybersecurity, CRC press.

Dubuisson, S. (2010). The computation of the Bhattacharyya distance between histograms without histograms. Image Processing Theory Tools and Applications (IPTA), 2010 2nd International Conference on, IEEE.

Edelsbrunner, H., D. Kirkpatrick and R. Seidel (1983). "On the shape of a set of points in the plane." Information Theory, IEEE Transactions on **29**(4): 551-559.

Fahmy, A. A., O. Ismail and A. K. Al-Janabi (2013). "Stereo Vision Based Depth Estimation Algorithm In Uncalibrated Rectification." International Journal of Video & Image Processing & Network Security **13**(2).

Faragher, R. (2012). "Understanding the basis of the Kalman filter via a simple and intuitive derivation." IEEE Signal Processing Magazine **29**(5): 128-132.

Farber, R. (2011). CUDA application design and development, Elsevier.

Ficocelli, M. and F. Janabi-Sharifi (2001). Adaptive filtering for pose estimation in visual servoing. Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, IEEE.

Fitzgerald, D. L., R. A. Walker and D. A. Campbell (2005). "A vision based emergency forced landing system for an autonomous uav."

Flach, P. (2012). Machine learning: the art and science of algorithms that make sense of data, Cambridge University Press.

Forsyth, D. and J. Ponce (2012). Computer Vision: A Modern Approach, Pearson.

Forsyth, D. A. and J. Ponce (2011). Computer Vision: A Modern Approach, Pearson Education, Limited.

Freund, Y. and R. E. Schapire (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. Computational learning theory, Springer.

Freund, Y. and R. E. Schapire (1996). Experiments with a new boosting algorithm. ICML.

Garratt, M., H. Pota, A. Lambert, S. ECKERSLEY-MASLIN and C. Farabet (2009). "Visual tracking and lidar relative positioning for automated launch and recovery of an unmanned rotorcraft from ships at sea." Naval Engineers Journal **121**(2): 99-110.

Georgoulas, C. and I. Andreadis (2011). "A real-time fuzzy hardware structure for disparity map computation." Journal of Real-Time Image Processing **6**(4): 257-273.

Gevers, T. and H. Stokman (2004). "Robust histogram construction from color invariants for object recognition." Pattern Analysis and Machine Intelligence, IEEE Transactions on **26**(1): 113-118.

Goldberg, D. E. and K. Deb (1991). "A comparative analysis of selection schemes used in genetic algorithms." Urbana **51**: 61801-62996.

Golub, G. H. and C. F. Van Loan (1996). Matrix Computations, Johns Hopkins University Press.

Gonçalves-Coelho, A. M., L. C. Veloso and V. J. A. S. Lobo (2007). Tests of a light UAV for naval surveillance. IEEE/OES Oceans'2007. Aberdeen, UK.

Gui, Y., P. Guo, H. Zhang, Z. Lei, X. Zhou, J. Du and Q. Yu (2013). "Airborne Vision-Based Navigation Method for UAV Accuracy Landing Using Infrared Lamps." Journal of Intelligent & Robotic Systems **72**(2): 197-218.

Guo, R.-h. and Z. Qin (2007). "An unscented particle filter for ground maneuvering target tracking." Journal of Zhejiang University SCIENCE A **8**(10): 1588-1595.

Guo, W., C. Han and M. Lei (2007). Improved unscented particle filter for nonlinear Bayesian estimation. Information Fusion, 2007 10th International Conference on, IEEE.

Guo, Z. and S. Osher (2011). "Template matching via l1 minimization and its application to hyperspectral data." Inverse Problems and Imaging **5**(1): 19-35.

Gupta, N. (2012). "Texture Memory in CUDA - What is texture Memory in CUDA Programming." Retrieved November,2014, 2014, from http://cuda-programming.blogspot.pt/2013/02/texture-memory-in-cuda-what-is-texture.html.

Hadid, A., M. Pietikainen and T. Ahonen (2004). A discriminative feature space for detecting and recognizing faces. Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, IEEE.

Haralick, B. M., C.-N. Lee, K. Ottenberg and M. Nölle (1994). "Review and analysis of solutions of the three point perspective pose estimation problem." International Journal of Computer Vision **13**(3): 331-356.

Hartley, R. and A. Zisserman (2003). Multiple View Geometry in Computer Vision, Cambridge University Press.

Hartley, R. I. (1997). "In defense of the eight-point algorithm." Pattern Analysis and Machine Intelligence, IEEE Transactions on **19**(6): 580-593.

Haug, A. J. (2012). Bayesian Estimation and Tracking: A Practical Guide, John Wiley & Sons.

Haykin, S. S., S. S. Haykin and S. S. Haykin (2001). Kalman filtering and neural networks, Wiley Online Library.

Hazeldene, A., A. Sloan, C. Wilkin and A. Price (2004). In-flight orientation, object identification and landing support for an unmanned air vehicle. Proceedings of the IEEE International Conference on Autonomous Robots and Agents.

Hérissé, B., T. Hamel, R. Mahony and F.-X. Russotto (2012). "Landing a VTOL unmanned aerial vehicle on a moving platform using optical flow." Robotics, IEEE Transactions on **28**(1): 77-89.

Higham, N. J. (1990). "Analysis of the Cholesky decomposition of a semi-definite matrix."

Hornberg, A. (2007). Handbook of Machine Vision, Wiley.

Hubbard, D., B. Morse, C. Theodore, M. Tischler and T. McLain (2007). Performance evaluation of vision-based navigation and landing on a rotorcraft unmanned aerial vehicle. Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on, IEEE.

Jähne, B., H. Haussecker and P. Geissler (1999). Handbook of computer vision and applications, Citeseer.

Jähne, B., H. Haussecker and P. Geissler (1999). Handbook of Computer Vision and Applications: Sensors and imaging, Academic Press.

Jalal, A. S. and V. Singh (2012). "The State-of-the-Art in Visual Object Tracking." Informatica (Slovenia) **36**(3): 227-248.

Janabi-Sharifi, F. and M. Marey (2010). "A kalman-filter-based method for pose estimation in visual servoing." Robotics, IEEE Transactions on **26**(5): 939-947.

Julier, S. J. (2002). The scaled unscented transformation. American Control Conference, 2002. Proceedings of the 2002, IEEE.

Kahn, A. D. (2010). Vision-based recovery and guidance for small unmanned air vehicles. Toronto, Ontario Canada: AIAA Guidance, Navigation, and Control Conference.

Kirk, D. B. and W. H. Wen-mei (2012). Programming massively parallel processors: a hands-on approach, Newnes.

Kong, W., D. Zhang, X. Wang, Z. Xian and J. Zhang (2013). Autonomous landing of an UAV with a ground-based actuated infrared stereo vision system. Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE.

Kotecha, J. H. and P. M. Djuric (2003). "Gaussian particle filtering." Signal Processing, IEEE Transactions on **51**(10): 2592-2601.

Kotecha, J. H. and P. M. Djuric (2003). "Gaussian sum particle filtering." Signal Processing, IEEE Transactions on **51**(10): 2602-2612.

Kraft, E. (2003). A quaternion-based unscented Kalman filter for orientation tracking. Proceedings of the Sixth International Conference of Information Fusion.

Kwok, N. M., G. Fang and W. Zhou (2005). Evolutionary particle filter: re-sampling from the genetic algorithm perspective. Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on, IEEE.

Lamoureux, P. Numerical Stability of the 8-Point Algorithm, ECSE.

Lange, S., N. Sunderhauf and P. Protzel (2009). A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments. Advanced Robotics, 2009. ICAR 2009. International Conference on, IEEE.

Lange, S., N. Sünderhauf and P. Protzel (2008). Autonomous landing for a multirotor UAV using vision. International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2008).

Lee, T., N. H. McClamroch and M. Leok (2006). Attitude maneuvers of a rigid spacecraft in a circular orbit. American Control Conference, 2006, IEEE.

Lepetit, V. and P. Fua (2005). "Monocular Model-Based 3D Tracking of Rigid Objects: A Survey." Foundations and Trends® in Computer Graphics and Vision **1**(1): 1-89.

Lewis, F. L., L. Xie and D. Popa (2008). Optimal and robust estimation: with an introduction to stochastic control theory, CRC.

Li, P., T. Zhang and A. E. Pece (2003). "Visual contour tracking based on particle filters." Image and Vision Computing **21**(1): 111-123.

Lienhart, R., A. Kuranov and V. Pisarevsky (2003). Empirical analysis of detection cascades of boosted classifiers for rapid object detection. Pattern Recognition, Springer**:** 297-304.

Lienhart, R. and J. Maydt (2002). An extended set of haar-like features for rapid object detection. Image Processing. 2002. Proceedings. 2002 International Conference on, IEEE.

Lima, J. P., F. Simões, L. Figueiredo and J. Kelner (2010). "Model based markerless 3D tracking applied to augmented reality." SBC **1**.

Lowe, D. G. (2004). "Distinctive image features from scale-invariant keypoints." International journal of computer vision **60**(2): 91-110.

Luong, Q.-T. and O. D. Faugeras (1996). "The fundamental matrix: Theory, algorithms, and stability analysis." International Journal of Computer Vision **17**(1): 43-75.

Ma, Y., S. Soatto, J. Kosecka and S. S. Sastry (2004). An Invitation to 3-D Vision: From Images to Geometric Models, Springer.

Mahalakshmi, T., R. Muthaiah and P. Swaminathan (2012). "Review Article: An Overview of Template Matching Technique in Image Processing." Research Journal of Applied Sciences **4**.

Marakas, G. M. (2003). Decision support systems in the 21st century, Prentice Hall ^ eNew Jersey New Jersey.

Martínez, C., P. Campoy, I. Mondragón and M. A. Olivares-Méndez (2009). Trinocular ground system to control UAVs. Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, Ieee.

Martins Junior, J. (2013). Como escrever trabalhos de conclusão de curso: instruções para planejar e montar, desenvolver, concluir, redigir e apresentar trabalhos monográficos e artigos. Como escrever trabalhos de conclusão de curso: instruções para planejar e montar, desenvolver, concluir, redigir e apresentar trabalhos monográficos e artigos, Vozes.

Matloff, N. (2011). "Programming on parallel machines." University of Calfornia, Davis.

Mejias, L., D. L. Fitzgerald, P. C. Eng and L. Xi (2009). "Forced landing technologies for unmanned aerial vehicles: towards safer operations." Aerial Vehicles: 415-442.

Merz, T., S. Duranti and G. Conte (2006). Autonomous landing of an unmanned helicopter based on vision and inertial sensing. Experimental Robotics IX, Springer**:** 343-352.

Miller, A., M. Shah and D. Harper (2008). Landing a UAV on a runway using image registration. Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, IEEE.

Mohan, A., C. Papageorgiou and T. Poggio (2001). "Example-based object detection in images by components." Pattern Analysis and Machine Intelligence, IEEE Transactions on **23**(4): 349-361.

Moore, R., S. Thurrowgood, D. Bland, D. Soccol and M. V. Srinivasan (2009). A stereo vision system for UAV guidance. Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, IEEE.

Morvan, Y. (2009). Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video. PhD thesis, Eindhoven University of Technology.

Navega, S. (2002). "Princípios essenciais do data mining." Anais do Infoimagem.

Ng, K. K. and E. J. Delp (2009). New models for real-time tracking using particle filtering. IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics.

Nixon, M. (2008). Feature Extraction & Image Processing, Elsevier Science.

Nummiaro, K., E. Koller-Meier and L. Van Gool (2003). "An adaptive color-based particle filter." Image and vision computing **21**(1): 99-110.

Ojala, T., M. Pietikainen and T. Maenpaa (2002). "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns." Pattern Analysis and Machine Intelligence, IEEE Transactions on **24**(7): 971-987.

Okuma, K., A. Taleghani, N. De Freitas, J. J. Little and D. G. Lowe (2004). A boosted particle filter: Multitarget detection and tracking. Computer Vision-ECCV 2004, Springer**:** 28-39.

Okutomi, M. and T. Kanade (1993). "A multiple-baseline stereo." Pattern Analysis and Machine Intelligence, IEEE Transactions on **15**(4): 353-363.

Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips (2008). "GPU computing." Proceedings of the IEEE **96**(5): 879-899.

Papageorgiou, C. P., M. Oren and T. Poggio (1998). A general framework for object detection. Computer vision, 1998. sixth international conference on, IEEE.

Park, S., J. Hwang, K. Rou and E. Kim (2007). "A New Particle Filter Inspired by Biological Evolution: Genetic Filter." International Journal of Electrical, Robotics, Electronics and Communications Engineering **9**: 1259 - 1263.

Park, S., J. P. Hwang, E. Kim and H.-J. Kang (2009). "A new evolutionary particle filter for the prevention of sample impoverishment." Trans. Evol. Comp **13**(4): 801-809.

Pe, U. S. N. B. o. N. and F. A. Carson (1969). Basic Optics and Optical Instruments, Dover Publications.

Periquito, D., J. C. Nascimento, A. Bernardino and J. Sequeira (2013). Vision-based Hand Pose Estimation-A Mixed Bottom-up and Top-down Approach. VISAPP (1).

Pervin, E. and J. A. Webb (1982). "Quaternions in computer vision and robotics."

Pietikäinen, M., A. Hadid, G. Zhao and T. Ahonen (2011). Computer Vision Using Local Binary Patterns. Computer Vision Using Local Binary Patterns, Springer London. **40:** E1-E2.

Porto, I. (2014). "UMS 2014, 3rd Workshop on European Unmanned Maritime Systems." Retrieved 07, August, 2014, from http://ums2014.inescporto.pt/.

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1996). Numerical recipes in C, Citeseer.

Prince, S. J. (2012). Computer vision: models, learning, and inference, Cambridge University Press.

Quinlan, J. R. (1986). "Induction of decision trees." Machine learning **1**(1): 81-106.

Radke, R. J. (2012). Computer Vision for Visual Effects, Cambridge University Press.

Ribeiro, M. I. (2004). "Kalman and extended kalman filters: Concept, derivation and properties." Institute for Systems and Robotics: 43.

Riola, J., J. Diaz and J. Giron-Sierra (2011). The prediction of calm opportunities for landing on a ship: Aspects of the problem. OCEANS, 2011 IEEE-Spain, IEEE.

Rosten, E. and T. Drummond (2005). Fusing points and lines for high performance tracking. Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, IEEE.

Rosten, E. and T. Drummond (2006). Machine Learning for High-Speed Corner Detection. Computer Vision – ECCV 2006. A. Leonardis, H. Bischof and A. Pinz, Springer Berlin Heidelberg. **3951:** 430-443.

Rosten, E., R. Porter and T. Drummond (2010). "Faster and Better: A Machine Learning Approach to Corner Detection." IEEE Trans. Pattern Anal. Mach. Intell. **32**(1): 105-119.

Rosten, E., R. Porter and T. Drummond (2010). "Faster and better: A machine learning approach to corner detection." Pattern Analysis and Machine Intelligence, IEEE Transactions on **32**(1): 105-119.

Ruetsch, G. and M. Fatica (2011). "CUDA Fortran for scientists and engineers." Nvidia Corporation, Santa Clara, CA.

Rui, Y. and Y. Chen (2001). Better proposal distributions: Object tracking using unscented particle filter. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, IEEE.

Sammut, C. and G. I. Webb (2011). Encyclopedia of machine learning, Springer.

Sanchez-Lopez, J. L., S. Saripalli, P. Campoy, J. Pestana and C. Fu (2013). Toward visual autonomous ship board landing of a VTOL UAV. Unmanned Aircraft Systems (ICUAS), 2013 International Conference on, IEEE.

Sanders, J. and E. Kandrot (2010). CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley Professional.

Santos, N. P., F. Melício, V. Lobo and A. Bernardino (2014). A Ground-based vision system for UAV autonomous landing. Poster presented at the 3rd Workshop on European Unmanned Maritime Systems. Porto, Portugal.

Saripalli, S. (2009). Vision-based autonomous landing of an helicopter on a moving target. Proceedings of AIAA Guidance, Navigation, and Control Conference, Chicago, USA.

Saripalli, S., J. F. Montgomery and G. Sukhatme (2002). Vision-based autonomous landing of an unmanned aerial vehicle. Robotics and automation, 2002. Proceedings. ICRA'02. IEEE international conference on, IEEE.

Saripalli, S., J. F. Montgomery and G. Sukhatme (2003). "Visually guided landing of an unmanned aerial vehicle." Robotics and Automation, IEEE Transactions on **19**(3): 371-380.

Särkkä, S. (2013). Bayesian filtering and smoothing, Cambridge University Press.

Sarmento, M. (2008). Guia Prático sobre a Metodologia Científica para Elaboração, Escrita e Apresentação de Teses de Doutoramento, Dissertações de Mestrado e Trabalhos de Investigação Aplicada. Lisboa, Portugal, Universidade Lusíada de Lisboa.

Schapire, R. E. (1990). "The strength of weak learnability." Machine learning **5**(2): 197-227.

Shakernia, O., Y. Ma, T. J. Koo and S. Sastry (1999). "Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control." Asian journal of control **1**(3): 128-145.

Sharp, C. S., O. Shakernia and S. S. Sastry (2001). A vision system for landing an unmanned aerial vehicle. Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, IEEE.

Simon, D. (2006). Optimal state estimation: Kalman, H infinity, and nonlinear approaches, John Wiley & Sons.

Stenger, B., P. R. Mendonça and R. Cipolla (2001). Model-Based Hand Tracking Using an Unscented Kalman Filter. BMVC.

Sturm, P. F. and S. J. Maybank (1999). On plane-based camera calibration: A general algorithm, singularities, applications. Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on., IEEE.

Subramanyam, M. (2013). "Feature Based Image Mosaic Using Steerable Filters and Harris Corner Detector." International Journal of Image, Graphics & Signal Processing **5**(6).

Sugandi, B., H. Kim, J. K. Tan and S. Ishikawa (2009). "A moving object tracking based on color information employing a particle filter algorithm." Artificial Life and Robotics **14**(1): 39-42.

Sur, F., N. Noury and M.-O. Berger (2008). Computing the uncertainty of the 8 point algorithm for fundamental matrix estimation. 19th British Machine Vision Conference-BMVC 2008.

Szeliski, R. (2010). Computer Vision: Algorithms and Applications, Springer.

Taiana, M., J. C. Nascimento, J. A. Gaspar and A. Bernardino (2008). Sample-Based 3D Tracking of Colored Objects: A Flexible Architecture. BMVC.

Taiana, M., J. Santos, J. Gaspar, J. Nascimento, A. Bernardino and P. Lima (2010). "Tracking objects with generic calibrated sensors: an algorithm based on color and 3D shape features." Robotics and autonomous systems **58**(6): 784-795.

TECHPOWERUP. (2014). "NVIDIA GeForce GT 750M." Retrieved June,2014, 2014, from http://www.techpowerup.com/gpudb/2224/geforce-gt-750m.html.

Thrun, S., W. Burgard and D. Fox (2005). Probabilistic robotics, MIT press.

Trawny, N. and S. I. Roumeliotis (2005). "Indirect Kalman filter for 3D attitude estimation." University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep **2**.

Turban, E., R. Sharda, D. Delen and T. Efraim (2007). Decision support and business intelligence systems, Pearson Education India.

Turner, R. D. (2012). Gaussian processes for state space models and change point detection, University of Cambridge.

Uosaki, K., Y. Kimura and T. Hatanaka (2003). Nonlinear state estimation by evolution strategies based particle filters. Evolutionary Computation, 2003. CEC'03. The 2003 Congress on, IEEE.

Van Der Merwe, R., A. Doucet, N. De Freitas and E. Wan (2000). The unscented particle filter. NIPS.

Van Verth, J. M. and L. M. Bishop (2008). Essential Mathematics for Games and Interactive Applications: A Programmer's Guide, CRC Press.

Viola, P. and M. Jones (2001). Rapid object detection using a boosted cascade of simple features. Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, IEEE.

Wan, E. A. and R. Van Der Merwe (2000). The unscented Kalman filter for nonlinear estimation. Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000, IEEE.

Wenzel, K. E., A. Masselli and A. Zell (2011). "Automatic take off, tracking and landing of a miniature UAV on a moving carrier vehicle." Journal of intelligent & robotic systems **61**(1-4): 221-238.

Williams, P. and M. Crump (2012). Intelligent landing system for landing UAVs at unsurveyed airfields. Proceedings of the 28th International Congress of the Aeronautical Sciences.

Wilt, N. (2013). The cuda handbook: A comprehensive guide to gpu programming, Pearson Education.

Wöhler, C. (2009). 3D Computer Vision: Efficient Methods and Applications, Springer.

Wu, A. D., E. N. Johnson, M. Kaess, F. Dellaert and G. Chowdhary (2013). "Autonomous flight in GPS-denied environments using monocular vision and inertial sensors."

Wu, F., Z. Hu and F. Duan (2005). 8-point algorithm revisited: Factorized 8-point algorithm. Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, IEEE.

Xiang, W., Y. Cao and Z. Wang (2012). Automatic take-off and landing of a quad-rotor flying robot. Control and Decision Conference (CCDC), 2012 24th Chinese, IEEE.

Xiaowei, Z., L. Hong and S. Xiaohong (2013). "Object Tracking with an Evolutionary Particle Filter Based on Self-Adaptive Multi-Features Fusion." Int J Adv Robotic Sy **10**(61).

Xu, G., X. Chen, B. Wang, K. Li, J. Wang and X. Wei (2011). A search strategy of UAV's automatic landing on ship in all weathe. Electrical and Control Engineering (ICECE), 2011 International Conference on, IEEE.

Xu, G., Y. Zhang, S. Ji, Y. Cheng and Y. Tian (2009). "Research on computer vision-based for UAV autonomous landing on a ship." Pattern Recognition Letters **30**(6): 600-605.

Yang, H. and G. Welch (2005). Model-based 3d object tracking using an extended-extended kalman filter and graphics rendered measurements. Computer Vision for Interactive and Intelligent Environment, 2005, IEEE.

Yang, Z.-F. and W.-H. Tsai (1998). "Using parallel line information for vision-based landmark location estimation and an application to automatic helicopter landing." Robotics and Computer-Integrated Manufacturing **14**(4): 297-306.

Zhang, Y., L. Shen, Y. Cong, D. Zhou and D. Zhang (2013). Ground-based visual guidance in autonomous UAV landing.

Zhang, Z. (2000). "A flexible new technique for camera calibration." Pattern Analysis and Machine Intelligence, IEEE Transactions on **22**(11): 1330-1334.

Zhao, Y. J. and H. L. Pei (2013). "Improved Vision-Based Algorithm for Unmanned Aerial Vehicles Autonomous Landing." Applied Mechanics and Materials **273**: 560-565.

Zhou, J., Y. Yang, J. Zhang and E. Edwan (2011). "Applying quaternion-based unscented particle filter on INS/GPS with field experiments." Proceedings of the ION GNSS, Portland: 1-14.

APÊNDICE A

UMS 2014

A.1. WORKSHOP ON UNMANNED MARITIME SYSTEMS

- *Poster* apresentado na Exponor, Porto em 29 e 30 de Maio de 2014 (Porto 2014, Santos, Melício et al. 2014).

A GROUND-BASED VISION SYSTEM FOR UAV AUTONOMOUS LANDING

GENERAL DESCRIPTION

In this study a vision system for autonomous landing of an existing commercial aerial vehicle (UAV) named AR4 aboard a ship, based on a single standard RGB digital camera is proposed. The vision system is located on the ship's deck and is used to estimate the UAV pose (3D position and orientation) during the landing process. Using a vision system located on the ship makes it possible to use an UAV with less processing power, decreasing its size and weight. Due to the non-linear nature of this implementation a 3D markerless framework is used based on a particle filter. The UAV will operate from the Portuguese Navy fast patrol boats (FPB), which implies the capability of landing in 27 m length, 5.9 m breadth vessels, with a 5x6 m small and irregular landing zone located at the boat's stern. The implementation of a completely autonomous system is very important in real scenarios, since this ships have only a small crew and UAV pilots are not usually available. Moreover a vision based system is more robust in an environment where GPS jamming can occur.

OBJECTIVES

- Develop an automatic landing system (AUTOmatic LANDing) for small UAV onboard ships (landing in very small areas);
- Increase the operational capability and ease of operation in adverse sea conditions;
- Operational validation of the concept.

Figure 1 - Approximation tests to the ship stern, after several landing tests the chosen landing method (because of its reliability) was a simple cord net (photos taken from previous tests with a different model).

SYSTEM ARCHITECTURE

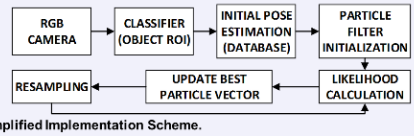


Figure 2 – Simplified Implementation Scheme.

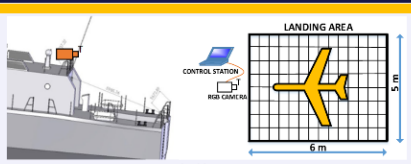


Figure 3 – Fast Patrol boat stern section (left) and the system Architecture and landing area illustration (right).

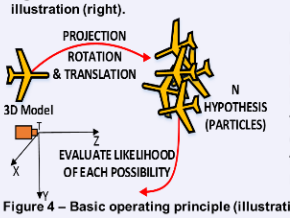


Figure 4 – Basic operating principle (illustration).

In every frame multiple object pose hypothesis are being tested, evaluating their proximity with the observed image. It's very important to use a likelihood function robust to illumination changes, especially in outdoor environments. The particle filter framework is mainly used due to it's nonlinear nature.

RESULTS

Figure 5 – Best possibilities obtained in the first 3 iterations of the filter for two different poses (Number of particles = 100).

Due to the initialization method used (database) we found that 4 iterations with 100 particles (tested possibilities at a frame) are enough for good detection performance. At 5 meters the mean error in translation is about 0.33 meters and in rotation is about 37,2 degrees for a single frame. We expect to further increase the accuracy of the system by using the UAV dynamic model in a temporal filtering and data association framework. The likelihood function used tries to be tolerant to illumination changes, but that robustness implies that it has poor accuracy in the initial rotation estimation.

FUTURE WORK

- Make more "real tests" to increase system performance and to identify implementation problems;
- Focus on using temporal filtering frameworks and dynamic constrains to complete the tracking system.

This work was funded by the EU Commission within the National Strategic Reference Framework (QREN) under grant agreement 23260 (AUTOLAND).

APÊNDICE B

ICIUS 2014

B.1. 10TH INTERNATIONAL CONFERENCE ON INTELLIGENT UNMANNED SYSTEMS

A Ground-Based Vision System for UAV Pose Estimation

***Abstract*—**We present a vision system based on a single frame of standard RGB digital camera to estimate the pose of an unmanned aerial vehicle (UAV). The envisaged application is of ground-based automatic landing, where the vision system is located on the ship's deck and is used to estimate the UAV pose (3D position and orientation) during the landing process. Using a vision system located on the ship makes it possible to use an UAV with less processing power, decreasing its size and weight. The proposed method uses a 3D model based pose estimation approach that requires the 3D CAD model of the UAV. Pose is estimated in a particle filtering framework. The implemented particle filter is inspired in the evolution strategies present in the genetic algorithms avoiding sample impoverishment. Results show position and angular errors are compatible with automatic landing system requirements, even without temporal filtering. The algorithm is suitable for real time implementation in standard workstations with graphical processing units.

***Keywords*—** Computer Vision, Model Based Pose Estimation, Autonomous Vehicles, Military Systems, Particle Filters.

I. INTRODUCTION

In the past several years, research on Autonomous vehicles has augmented the number of possible civilian and military applications. The key requirement for most of these systems, especially in military environment, is reliability, guaranteeing a very low failure rate in normal operation.

Portugal has the 10th largest exclusive economic zone (EEZ) in the world and this makes it difficult to control the territorial waters (approximately 41335 km2) with limited resources. Nowadays, fast patrol boats (FPB) have an important role in this context, but their efficacy can be significantly improved by the support of unmanned aerial vehicles (UAVs). Because this kind of ships has a small crew and UAV pilots are not usually available, the automation of all the UAV operations that still require human intervention (e.g. landing) is a priority. The available landing site in this kind of ships is a small and irregular area with size of around 5x6m (stern section). A vision based system is also more robust in environments where GPS jamming can occur, increasing the system performance in real operation scenarios [1].

The available landing area limits the maximum UAV payload. Most of the research made in this area are based on on-board systems [2, 3], whereas ground systems [4, 5] are not usually considered.

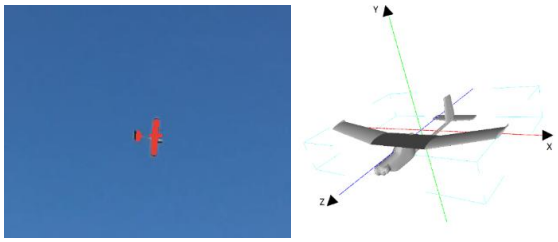


Figure 1 - Pose estimation in a real scenario (left) and the used UAV Model (right).

Nuno Pessanha Santos – nuno.pessanha.santos@marinha.pt / nuno.pessanha.santos@gmail.com

Using a system located on the ship's deck makes it possible to use an UAV with less processing power, decreasing its size and weight. The Portuguese navy has started first tests with low cost commercial off-the-shelf UAVs in 2005, testing several hardware and landing configurations [6].

For the vision based system we adopt a 3D model based approach (as seen in Figure 1). Nowadays all UAVs have their 3D CAD model available, so their pose (3D position and orientation) can be estimated using a class of methods for 3D model-based tracking based on particle filters. These represent the distribution of an object's 3D pose as a set of weighted hypotheses (particles) [7, 8]. Hypotheses are tested by explicitly projecting the object model in the image, with a certain rotation and translation, and comparing the actual image pixel information. The particle filter, in contrast with other methods like Kalman Filter, can be used in non-linear situations for instance in rotation estimation and multimodal distributions typical of pose estimation methods [9-11].

Initialization is a common problem in particle filters when no a-priori information of object location is known. In this case the particles are usually scattered randomly in space around a specific position [11]. This method is usually slow to converge, affecting the filter performance and robustness. Another solution may be to increase the particle number, but this increases the required processing power, which is usually limited. In this paper we propose a method to make a simple and efficient initialization, based on an initial database of known object poses.

The resampling strategies for the created particles are inspired in the evolution strategies present in the genetic algorithms [12-14], avoiding sample impoverishment [15]. Crossover and mutation operators are adopted, increasing the filter performance and decreasing the number of needed particles for object tracking.

In Section 2 is described the 3D model-based pose estimation system architecture, which is divided in the following major components: (1) Airship detection, (2) Particle initialization, (3) Particle evaluation and (4) Pose optimization. In section 3 some experiments concerning synthetic and real scenarios are presented. Finally, in section 4 we present the conclusions of the paper and provide directions for further research work.

II. 3D MODEL-BASED POSE ESTIMATION SYSTEM

This section introduces the proposed 3D model based pose estimation system. In this study, we consider UAV detection on an outdoor environment by a moving platform (ship). The state vector $\boldsymbol{P_t}$ is defined by the 3D position (X, Y, Z), and the object orientation represented by X, Y, Z Euler angles (γ, β, α). The main goal is to estimate the pose of the vehicle at each time $\{\boldsymbol{P_t}; \boldsymbol{t} \boldsymbol{\epsilon} \mathbf{N}\}$ and send this information to a control station. In this paper we do not consider information fusion between different time steps (tracking) but solely the performance evaluation of the detection and pose estimation algorithms. Tracking will be addressed in future work.

The proposed method is divided in 4 parts:

- **Airship detection** – In this stage, the image is scanned in the search for regions that may contain aerial vehicles with the appearance of the UAV to land;
- **Particle initialization** – In this stage we try to match the regions found in the previous stage to a pre-trained database of UAV bounding boxes in multiple poses. All poses with sufficient match score will initialize particles for the pose estimation procedure;
- **Particle evaluation** – Each of the particles generated in the previous stage will be ranked by likelihood, according to the distance of the particle two different metrics are used;
- **Pose optimization** – Based on the likelihood metrics, particles are resamples and optimized to best fit the image appearance.

A. AIRSHIP DETECTION

The initial region of interest (ROI) detection is very important since we will use feature points from the object to make the particle initialization. Since we are operating in outdoor environments it is very important to achieve luminosity invariance, guaranteeing a high object detection rate.

The initial UAV detection is made using a trained local binary pattern cascade classifier [16],. This initial image segmentation is very important since we are operating in the exterior and the presence of other objects in the frame (for instance clouds) affects the performance of the proposed system architecture.

B. PARTICLE INITIALIZATION

The airship detection stage results in an oriented bounding box (BB) that indicates the image position and rough posture of the UAV. To initialize particles close to the real posture of the UAV, we compare the detected bounding box characteristics with synthetically generated bounding boxes of the UAV in many possible poses. This database can be created offline and indexed in an efficient way for fast run-time access. Since we use a perspective camera model, the database (training stage) can be made independent of object position in the image.

The detected bounding boxes are obtained applying the FAST [17, 18] feature detector on the observation frame, selecting the key points that are inside the obtained ROI. This ROI is simply obtained in the airship detection phase, making it possible to select the feature points belonging to the UAV.

The database is created by rendering images of the UAV 3D CAD model at a fixed position but varying rotation according to a Gaussian distribution with $\mathcal{N}(0,90)$ for the γ, β and α parameters. The created database represents 20000 orientation possibilities: 13620 possibilities (68.1%) are contained in the frontal semi sphere and the rest (6380) are contained in the back semi sphere. This gives an average sample difference of 1.23º for the front possibilities and 1.8º for the back semi sphere possibilities. We favour the front hemisphere because in a landing situation the UAV pose is more likely located on the frontal semi sphere.

For each generated possibility is obtained the BB that better fits the projected object and is stored in a database indexed by angle (θ) and aspect ratio (AR). In order to estimate the initial object pose of the UAV detected in the real image (3D position and orientation), we compare the bounding box angle and BB aspect ratio between the object in the observation frame and the database.

$$AR = \frac{BB_{Widht}}{BB_{Height}} \tag{1}$$

The difference between the observation (obs) and the database (data) is calculated online using the Euclidean distance as in:

$$d(\theta, AR) = \sqrt{(\theta_{obs} - \theta_{data})^2) + (AR_{obs} - AR_{data})^2} \tag{2}$$

For the best possibilities we assume that the object's points are all in the same depth (Z coordinate), projected in a plane parallel to the image. The Z coordinate can be computed by the relationship between the BB areas and depth.

$$Z = Z_{Database}\sqrt{\frac{Area_{Database}}{Area_{Observation}}} \tag{3}$$

The X and Y coordinates are calculated by the relation between the coordinate of the centre of the observation BB, the obtained Z coordinate and the camera intrinsic parameters – focal length and camera centre coordinates (obtained by calibration).

$$X = \frac{Z(BB_X - C_X)}{f_x} \tag{4}$$

$$Y = \frac{Z(BB_Y - C_Y)}{f_y} \tag{5}$$

It is important to guarantee a correct camera calibration to ensure precision in system performance. Figure 2 shows a block diagram of the particle initialization procedure.

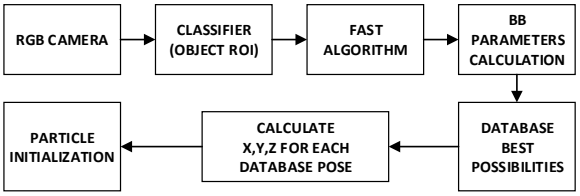


Figure 2 - Particle initialization architecture.

C. PARTICLE EVALUATION

Each particle corresponds to a 3D pose. By projecting the 3D CAD model of a particle in the image frame, we obtain an expectation of its location (hypothesis). To check the quality of this hypothesis, we will sample

the current image at the expected location and evaluate a likelihood function. To decrease the estimation error two different likelihood functions were used:

- Difference between the inner (object) and the outer histogram limited by a bounding box (above 25 meters estimation);
- Hybrid approach combining the likelihood function described before with a contour based method (under 25 meters estimation).

Above 25 meters the particle with the highest weight is the particle that maximizes the difference between the two histograms (See figure 3). Using this approach is possible to have a likelihood function invariant to illumination changes. This robustness is very important since this is an outdoor application, where the brightness variations are often unpredictable.

The histograms are obtained in the RGB colour space (12 bins for each colour – B = 12), and the distance between them are calculated using the Bhattacharyya similarity metric as in [19, 20]:

$$L_{texture} = 1 - \sum_{b=1}^{B} \sqrt{h^{inner}(b).h^{outer}(b)} \quad (6)$$

Where h^{inner} is the inner histogram, h^{outer} is the outer histogram and b is the respective histogram bin.

Figure 3 - Example of object inner (object) and outer (between the object and the bounding box) regions where color histograms are computed for the particle likelihood metric.

The hybrid likelihood function (used in under 25 meters estimation) combines the likelihood function described before, with contour information. The set of visible edges (3D CAD model) are projected in a 2D plane according to the currently tested pose hypothesis. The edge line segments are identified and a sample point (v) in the middle is generated. Then a 1D perpendicular search (as seen in Figure 4) is made to match the sample points with the nearest edge (m). After calculating the matches the contour likelihood is calculated as in [21]:

$$L_{contour} = exp^{\left(-\lambda_v \frac{(p_v - p_m)}{p_v}\right)} \times exp^{(-\lambda_e \bar{e})} \quad (7)$$

where $\bar{e}$ is the arithmetic average distance between the sample points and the edge points, the λ_v and λ_e are sensitivity terms used to tune the contour likelihood function, p_v is the number of visible sample points and p_m is the number of matched sample points.

To increase system performance when a frame is obtained the magnitude and orientation of the gradient (Sobel filter) is calculated and stored.

In order to obtain less error in the matching process, for each sample point the angle is calculated and compared to the gradient orientation at the matched point. If this angle diverges by 45 degrees the matched point is excluded minimizing the existing error.

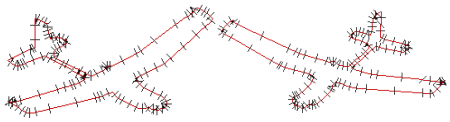


Figure 4 - Sample points and 1D search (black lines).

The hybrid likelihood is created by the junction of the two likelihood functions described before in equations (6) and (7), and is calculated as in:

$$L_{total} = L_{contour} + A \times L_{texture} \quad (8)$$

Where A is a relative sensitivity term used to fine tune the hybrid likelihood function. The simplified schematic of this calculation can be seen in Figure 5.

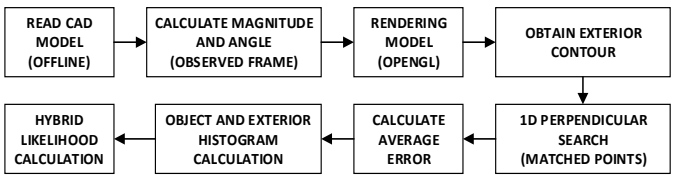


Figure 5 - Hybrid likelihood calculation – simplified schematic.

D. POSE OPTIMIZATION

Given a set of particles and a likelihood function, the optimization process (Figure 6) operates in three phases:

- Bootstrap;
- Coarse Optimization;
- Fine Optimization.

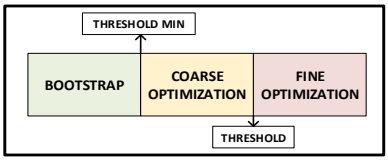


Figure 6 - Particle Filter optimization Phases.

In the Bootstrap phase the best 100 possibilities obtained by comparison with the database are collected in a list (Top 100). The likelihood of each particle is evaluated and stored in the list. The best M particles are stored in an auxiliary buffer (Top M). The particles with weight very close to zero (below $\delta = 0.01$) are eliminated and replaced with a random particle selected from the Top M buffer, added with Gaussian noise of covariance Σ_B. At this point, all particles have likelihood above δ.

Then, we run up to 10 improvement steps. In each step all particles are evaluated and compared to those in the Top M, if the obtained weight is higher the Top M is updated. If there are at least two particles in the Top M with likelihood bigger than "Threshold min", the bootstrap phase ends. Otherwise, each particle is perturbed with Gaussian noise with covariance Σ_B. If after 10 of these improvement steps no two particles are above "Threshold min", the bootstrap process is restarted up to a maximum of 3 restarts. In our experiments we have noticed that the occurrence of restarts is very rare.

The coarse optimization phase begins when at least two particles have a weight higher than "Threshold min". At any stage of the coarse and fine optimization phases, the best two particles have an important role in the optimization process because they will provide the "chromosomes" for an approach inspired on genetic algorithms [13, 14]. After extensive experimentation we have found such an approach much more effective than using only random perturbations and resampling as in conventional particle filters. The approach works as follows.

Each particle in the Top 100 list coming from the bootstrap process is analyzed. If the particle is the best one, it is perturbed with some Gaussian noise. If the particle weight is smaller, the best 2 particles are combined by crossover to create a new particle. The crossover operation consists in random selection of attributes (X, Y, Z, γ, β, α) of the original particles. To half of the particles generated by crossover is applied a soft mutation by adding Gaussian noise to the result. Together these rules allow a focused particle diversity, simultaneously converging to the best solution and avoiding possible local minima. The process stops when at least two of the particles are above value "Threshold". If this does not happen in 10 iterations, the pose optimization filter returns to the bootstrap phase automatically.

The fine optimization phase is analogous to the coarse phase but the Gaussian noise variance applied in mutation is lower, in order to make a fine-tuning to the estimated pose. The fine optimization phase ends after 5 iterations.

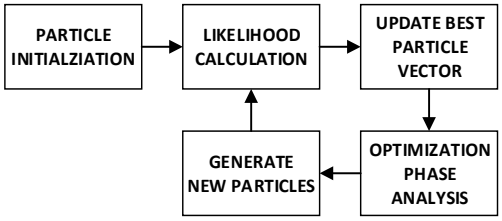


Figure 7 - Pose optimization architecture.

III. EXPERIMENTAL RESULTS

In this section we show results from UAV detection, particle initialization and pose estimation on real images. Because with real images we do not have ground truth information, results are qualitatively evaluated through observation of the CAD model projection on the images. To quantitatively evaluate the performance of our method we also show a statistical analysis of pose estimation on a large number of synthetically generated images with ground truth. The method was implemented in C++ on a 2.40 GHz Intel i7 CPU and NVIDIA GeForce GT 750M. All computational times and results presented in this section refer to this platform.

A. OBJECT DETECTION

The initial object detection is made using a cascade classifier that identifies the object localization ROI as seen in Figure 8. The average running time in a 1280x720 frame is about 59 ms.

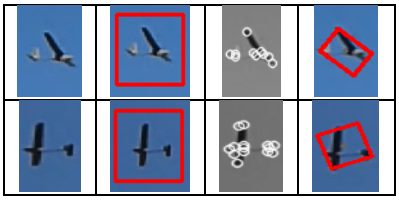


Figure 8 - Observation frame Bounding Box detection. From left to right: (i) original image; (ii) detected ROI by sliding window cascaded classifier trained on LBP features; (iii) FAST features detected inside previous ROI; and (iv) oriented bounding box enclosing detected features.

After the ROI was obtained, the FAST algorithm is applied to the frame and the points that are inside that ROI are used to calculate the object BB. The obtained average running time of the FAST algorithm in a 1280x720 frame is less than 1 ms.

B. PARTICLE INITIALIZATION

Figure 9 - In red we can see the projection on the real image of randomly selected possibilities for particle initialization from the best database matches.

For the obtained bounding box the angle and aspect ratio is calculated and compared with the database, as explained in Section II.D. Since the database is entirely loaded in the beginning of the program, the average running time for all database is less than 1 ms.

The best possibilities (as seen in Figure 9) are used to initialize the pose optimization procedure. There are some possibilities with the same relation of BB and angle that do not correspond to the real pose of the UAV, but those possibilities are filtered in the first iteration. This filtering is done replacing all the particles with very low likelihood with a best particle adding some Gaussian noise to guarantee particle diversity.

C. PARTICLE OPTIMIZATION

Figure 10 - Best Particles obtained in the first 3 iterations for two different poses (Particles = 100).

The described method was applied in real scenarios (as seen in Figure 10). We found that 4 iterations with 100 particles are enough for a good detection performance. This is only possible because the particle initialization procedure gives already good approximations to the pose, improving the convergence rate of the optimization phase.

The average likelihood for the real scenario in Figure 10 was calculated (Figure 11) changing the particle number (N) and the database poses (D) used for the initialization. When D is lower than N the remaining

possibilities are created adding Gaussian noise to the best possibilities. In the plots the two horizontal lines correspond to the selected "Threshold" (upper line) and "Threshold min" (lower line) that control the coarse and fine phases of the optimization algorithm depicted in Figure 6. Bellow "Threshold Min" we are at the bootstrap phase (best database possibilities and some Gaussian noise depending of the selected parameters), in the middle we are at the coarse optimization phase and above "Threshold" we are at the fine optimization phase.

As we can see from the Figure 11 the convergence to the coarse optimization is very fast (in average 2 iterations), demonstrating the importance of the initialization process in the convergence to the final result. It is also shown that more database possibilities do not necessarily correspond to better results, for example the 200 particles used in Figure 11 for 150 and 200 database possibilities. The "Threshold" parameter must be set to a value that corresponds to acceptable particle pose likelihood, being a compromise between speed and accuracy. The best speed vs performance parameters obtained are 100 particles with 100 poses used from the database.

The average computational time for each iteration is shown in Table I. To calculate the magnitude and angle of the gradient of each frame (used for the contour likelihood calculation) an average running time of 25 ms must be added. Thus, for 100 particles, we get an average time of 790 ms (approximately 1.26 fps).

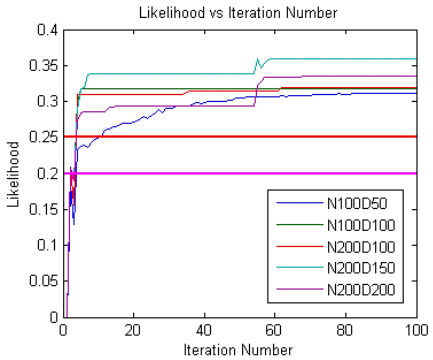


Figure 11 - Average likelihood vs Iteration number.

TABLE I

COMPUTATIONAL TIME

Particle Number (N)	Time (ms)	Frames per second (fps)
1	7.5	133.3
10	72	13.8
20	148	6.7
50	374	2.6
100	765	1.3
200	1659	0.6

D. QUANTITATIVE PERFORMANCE EVALUATION

A test set was created for several UAV distances: 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 meters, using 850 synthetic frames for each distance. The final pose estimate was obtained from the most likely particle running the proposed algorithm with 100 particles and 100 database poses.

As we can see from the Figure 12 the translation error decreases with proximity, obtaining a mean value at 5 meters (as seen in Table II) of 0.27 meters. The landing area is an irregular area of 5x6 meters, so we need to guarantee a minimum translation error of 1 meter in order to ensure a reliable landing.

As we are operating in an outdoor environment and with moving platforms, a sudden variation in the atmospheric conditions can lead to failure. This resolution in translation is clearly achieved, guaranteeing a UAV good estimation in 3D position across the range of distances covered in this test. We expect to further increase the accuracy of the system and reduce the amount of outliers (outliers are currently less than 5% of the cases) by using the UAV dynamic model in a temporal filtering and data association framework.

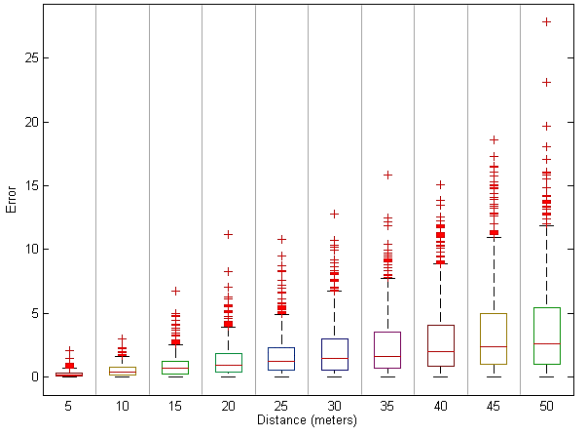


Figure 12. Translation Error (meters).

TABLE II

TRANSLATION ERROR

Distance (meters)	Mean value (meters)	Median value (meters)
5	0.27	0.19
10	0.54	0.40
15	0.92	0.70
20	1.32	0.92
25	1.68	1.28
30	2.1	1.45
35	2.4	1.66
40	2.9	1.99
45	3.5	2.43
50	3.8	2.61

The rotation error at 5 meters also decreases with proximity but less markedly than with translation. It has a median value of 9.4 degrees (as seen in Table III). This error values can be achieved by the combination of the hybrid likelihood formula as described in section II.C for distances under 25 meters. The higher error for distances above 25 meters happens mainly because the UAV model has the majority of its pixels concentrated in its wings and the used likelihood formula is based on the maximization of the difference between two pixel areas, originating a lower sensibility in variations that happen in the rest of its body. The rotation error becomes particularly evident in poses where the UAV body is partially occluded by its wings, generating some situations where the particle is shifted around 180 degrees from the observed pose. This ambiguity will be tackled in future work by using dynamic constraints between pose and velocity direction in a temporal filtering framework.

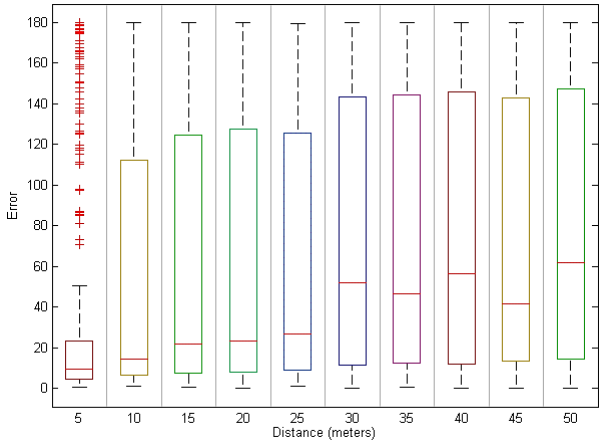


Figure 13. Rotation Error (Degrees).

TABLE III

ROTATION ERROR

Distance (meters)	Mean value (Degrees)	Median value (Degrees)
5	37.2	9.4
10	52.3	14.6
15	60.5	22.1
20	62.5	23.3
25	63.3	27.0
30	75.4	52.2
35	75.1	46.6
40	77.2	56.5
45	74.9	41.8
50	79.3	61.7

IV. CONCLUSIONS

A method was introduced and tested for estimating the pose of a known UAV in images acquired onboard a ship, for the purpose of automatic landing. The presented algorithms features a UAV detection method based on a cascaded classifier; a novel particle initialization methodology with a pre-trained database of images indexed by bounding box properties; and a particle optimization stage inspired in evolutionary methods that has shown interesting convergence properties. The implemented architecture allows a very accurate position estimation (about 4% median error at 5 m) and a reasonable attitude error (about 10º median error at 5m). We consider these precision levels suitable for the following stages of the work, which will focus on using temporal filtering frameworks and dynamic constraints to complete the tracking system.

ACKNOWLEDGMENT

This work was funded by the EU Commission within the National Strategic Reference Framework (QREN) under grant agreement 23260 (AUTOLAND) and by the Portuguese government through FEDER funds under project SEAGULL (SI IDT 34063) and by the FCT project [PEst-OE/EEI/LA0009/2013].

REFERENCES

1. Wu, A.D., E.N. Johnson, M. Kaess, F. Dellaert, and G. Chowdhary, *Autonomous flight in GPS-denied environments using monocular vision and inertial sensors.* AIAA J. of Aerospace Information Systems (JAIS), 2013. **10**(4): p. 14.
2. Cesetti, A., E. Frontoni, A. Mancini, P. Zingaretti, and S. Longhi, *A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks.* Journal of Intelligent and Robotic Systems, 2010. **57**(1-4): p. 233-257.
3. Xu, G., Y. Zhang, S. Ji, Y. Cheng, and Y. Tian, *Research on computer vision-based for UAV autonomous landing on a ship.* Pattern Recognition Letters, 2009. **30**(6): p. 600-605.
4. Kong, W., D. Zhang, X. Wang, Z. Xian, and J. Zhang. *Autonomous landing of an UAV with a ground-based actuated infrared stereo vision system*. in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. 2013. IEEE.
5. Martínez, C., P. Campoy, I. Mondragón, and M.A. Olivares-Méndez. *Trinocular ground system to control UAVs*. in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. 2009. Ieee.
6. Gonçalves-Coelho, A.M., L.C. Veloso, and V.J.A.S. Lobo, *Tests of a light UAV for naval surveillance*, in *IEEE/OES Oceans'2007*. 2007: Aberdeen, UK.
7. Doucet, A., N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. 2001: Springer.
8. Haug, A.J., *Bayesian Estimation and Tracking: A Practical Guide*. 2012: Wiley.
9. Lepetit, V. and P. Fua, *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey.* Foundations and Trends® in Computer Graphics and Vision, 2005. **1**(1): p. 1-89.
10. Challa, S., *Fundamentals of Object Tracking*. 2011: Cambridge University Press.
11. Forsyth, D.A. and J. Ponce, *Computer Vision: A Modern Approach*. 2011: Pearson Education, Limited.
12. Boli, M., P.M. Djuri, and S. Hong, *Resampling algorithms for particle filters: a computational complexity perspective.* EURASIP J. Appl. Signal Process., 2004. **2004**: p. 2267-2277.

13. Park, S., J.P. Hwang, E. Kim, and H.-J. Kang, *A new evolutionary particle filter for the prevention of sample impoverishment.* Trans. Evol. Comp, 2009. **13**(4): p. 801-809.

14. Kwok, N.M., G. Fang, and W. Zhou. *Evolutionary particle filter: re-sampling from the genetic algorithm perspective*. in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. 2005. IEEE.

15. Simon, D., *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. 2006: Wiley.

16. Pietikäinen, M., A. Hadid, G. Zhao, and T. Ahonen, *Computer Vision Using Local Binary Patterns*, in *Computer Vision Using Local Binary Patterns*. 2011, Springer London. p. E1-E2.

17. Rosten, E. and T. Drummond, *Machine Learning for High-Speed Corner Detection*, in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Editors. 2006, Springer Berlin Heidelberg. p. 430-443.

18. Rosten, E., R. Porter, and T. Drummond, *Faster and Better: A Machine Learning Approach to Corner Detection.* IEEE Trans. Pattern Anal. Mach. Intell., 2010. **32**(1): p. 105-119.

19. Taiana, M., J. Santos, J. Gaspar, J. Nascimento, A. Bernardino, and P. Lima, *Tracking objects with generic calibrated sensors: an algorithm based on color and 3D shape features.* Robotics and autonomous systems, 2010. **58**(6): p. 784-795.

20. Taiana, M., J.C. Nascimento, J.A. Gaspar, and A. Bernardino. *Sample-Based 3D Tracking of Colored Objects: A Flexible Architecture*. in *BMVC*. 2008.

21. Choi, C. and H.I. Christensen. *Robust 3D visual tracking using particle filtering on the SE (3) group*. in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. 2011. IEEE.

APÊNDICE C

OCEANS' 15 MTS/IEE

C.1. EXTENDED ABSTRACT

A Ground-Based Vision System for UAV Tracking

We present a vision system based on a standard RGB digital camera to track an unmanned aerial vehicle (UAV) during the landing process aboard a ship. The vision system is located on the ship´s deck and is used to estimate the UAV pose (3D position and orientation) during the landing process.

Figure 1. UAV Ship landing – Practical tests.

Using a vision system located on the ship makes it possible to use an UAV with less processing power, decreasing its size and weight. The proposed method uses a combination of a boosted classifier for UAV detection, a fast pose hypotheses generator and particle based optimization for pose estimation and an unscented Kalman filter (UKF) approach for dynamic tracking. The implemented particle optimization method is inspired in the evolution strategies developed in the genetic algorithms' framework to avoiding sample impoverishment.

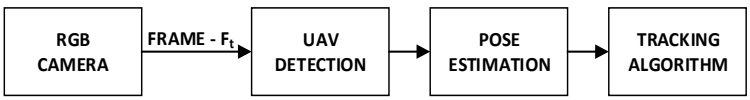


Figure2. Simplified system architecture.

The proposed method is divided in 5 parts:

- **Airship detection** – In this stage, the image is scanned in the search for regions that may contain aerial vehicles with the appearance of the UAV to land;

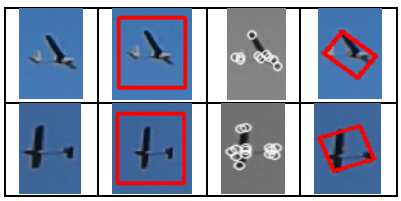


Figure3. UAV detection. From left to right: (i) original image; (ii) detected ROI by sliding window cascaded classifier trained on LBP features; (iii) FAST features detected inside previous ROI; and (iv) oriented bounding box enclosing detected features.

- **Particle initialization** – In this stage we try to match the regions found in the previous stage to a pre-trained database of UAV bounding boxes in multiple poses. All poses with sufficient match score will initialize particles for the pose estimation procedure;

Figure 4. In red we can see the projection on the real image of randomly selected possibilities for particle initialization from the best database matches.

- **Particle evaluation** – Each of the particles generated in the previous stage will be ranked by likelihood. To decrease the estimation error two different likelihood functions were used:
  - Difference between the inner (object) and the outer histogram limited by a bounding box (above 25 meters distance);
  - Hybrid approach combining the likelihood function described before with a contour based method (under 25 meters distance).
- **Pose optimization** – Based on the likelihood metrics, particles are resamples and optimized to best fit the image appearance;

Figure 5. Best Particles obtained in the first 3 iterations for two different poses (Number of Particles = 100).

A test set was created for several UAV distances: 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 meters, using 850 synthetic frames for each distance. The final pose estimate was obtained from the most likely particle running the proposed algorithm with 100 particles and 100 database poses.

Translation Error			Rotation Error		
Distance (meters)	**Mean value (meters)**	**Median value (meters)**	**Distance (meters)**	**Mean value (Degrees)**	**Median value (Degrees)**
5	0.27	0.19	5	37.2	9.4
10	0.54	0.40	10	52.3	14.6
15	0.92	0.70	15	60.5	22.1
20	1.32	0.92	20	62.5	23.3
25	1.68	1.28	25	63.3	27.0
30	2.1	1.45	30	75.4	52.2
35	2.4	1.66	35	75.1	46.6
40	2.9	1.99	40	77.2	56.5
45	3.5	2.43	45	74.9	41.8
50	3.8	2.61	50	79.3	61.7

Table 1. Translation and Rotation errors on a synthetic data set.

As we can see from the Table 1 the translation error decreases with proximity, obtaining a mean value at 5 meters of 0.27 meters. The landing area is an irregular area of 5x6 meters, so we need to guarantee a minimum translation error of 1 meter in order to ensure a reliable landing. The rotation error at 5 meters also decreases with proximity but less markedly than with translation. It has a median value of 9.4 degrees (as seen in Table 1). This error values can be achieved by the combination of the hybrid likelihood formula as described in the particle evaluation phase for distances under 25 meters.

- **Filtering** – The best particle in each iteration is filtered using an unscented particle filtering framework based on a state vector $\boldsymbol{x_t} = [x, y, z, v_x, v_y, v_z, q_w, q_x, q_y, q_z, w_x, w_y, w_z]$, where x, y and z are the UAV 3D position, v_x, v_y and v_z are the UAV linear velocities for each axis, q_w, q_x, q_y and q_z is the quaternion representation of the UAV rotation and w_x, w_y and w_z are the angular velocities around each axis.

The temporal filtering between frames is an essential step, allowing us to use information of the object between frames. A constant linear and angular velocity model is used to filter the individual pose estimates at each frame and decrease the obtained estimation error. An example of the filtering framework applied to the synthetic data set can be seen in the Figure 6, where the best particle of each iteration step (Airship detection, Particle initialization, Particle evaluation and pose optimization) is filtered using the unscented Kalman filter.

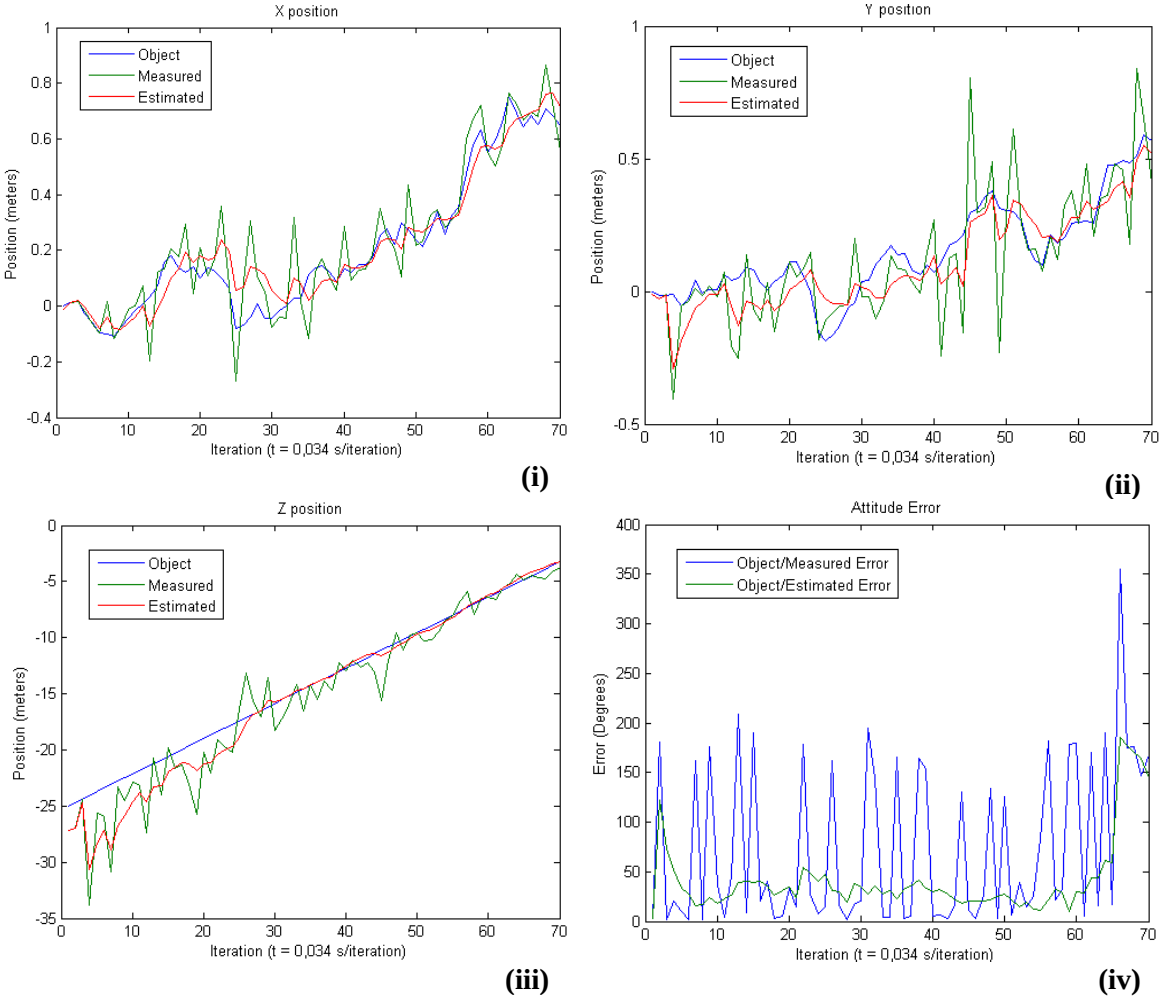


Figure 6. Temporal filtering – Unscented particle filter: (i) Translation in X; (ii) Translation in Y; (iii) Translation in Z; and (iv) Rotation error.

Results show position and angular errors are compatible with automatic landing system requirements. The implementation of a completely autonomous system is very important in real scenarios, since the ships have only a small crew and UAV pilots are not usually available. In the present more field tests are being made to increase the overall system performance.

APÊNDICE D
DESCRIÇÃO SIMPLIFICADA - ARQUITETURAS

D.1. ARQUITETURA DE DETEÇÃO DE POSE

Descrição Simplificada: Deteção de Pose – Arquitetura Final

```
1. MAIN: //Ciclo Principal
2. Objeto = Ler_Objeto("UAV.obj"); //Ler objeto para RAM
3. BaseDados = Ler_Base_de_dados("BaseDados.txt"); //Ler Base de dados para RAM
4. Classificador = Ler_Parâmetros_Classificador("LBP.xml"); //Ler Classificador treinado para RAM
5. Frame_Actual = Ler_Frame_Câmara();
6. Frame_Actual_sem_distorção = Distorção_Remap(Frame_Actual); //Corrigir distorção Radial & Tangencial
7. [Magnitude_Sobel, Ângulo_Sobel] = Sobel(Frame_Actual); //Sobel – Magnitude e Ângulo do Gradiente
8.do{ //Fica no loop até detetar um objecto utilizando o classificador
9. ROI_Objeto = Detetar_UAV(Frame_Actual_sem_distorção,Classificador);
10. } while(ROI_Objeto.size()==0);
11. Atual_Gray = Color_Gray(Frame_Actual_sem_distorção); //RGB para Gray
12. Keypoints_Frame = FAST_Keypoints(Atual _Gray); //Keypoints do algoritmo FAST
13. Keypoints_ROI = Selecionar_ROI(Keypoints_Frame); //Keypoints ROI (são os que interessam - objeto)
14. Bounding_Box = Rectangulo_min_Area(Keypoints_ROI); //Cálculo da Bounding Box
15. [Área,Ângulo,Largura,Altura,BB_X,BB_Y]=Parâmetros(Bounding_Box); //Cálculo dos parâmetros da BB obtida
16. double AR = Largura/Altura; //Cálculo do Aspect Ratio
17. for(int h=0;h< N_base_dados ;h++){ //Cálculo da distância euclidiana para cada valor da base de dados
18. Valor[h]=sqrt((pow((Ângulo-(BaseDadosAng[h])),2))+(pow((AR-(BaseDadosAR[h])),2)));
19. }
20. for(int q=0;q< N_inicialização ; q++){ //Cálculo dos parâmetros X,Y,Z, Alfa, Beta e Gama - inicialização
```

```
21. int índice_Valor_Maior = Maior(Valor);
21. Z_H = (sqrt((BaseDadosArea_Valor[índice_Valor_Maior])/Área)*4);
22. X_H = (Z_H*(BB_X-C_X))/f_x;
23. Y_H = (Z_H*(BB_Y-C_Y))/f_y;
24. P[0][q]= X_H;
25. P[1][q]= Y_H;
26. P[2][q]= Z_H;
27. P[3][q]= BaseDadosAlfa[índice_Valor_Maior];
28. P[4][q]= BaseDadosBeta[índice_Valor_Maior];
29. P[5][q]= BaseDadosGama[índice_Valor_Maior];
30. }
31. for(int p=0; p< Iteração; p++){ //Início do Filtro – Otimização da
pose
32. for(int i=0; i< N_Partículas; i++){ //Cálculo da likelihood para cada
partícula
33. Partícula =
rendering(P[0][i],P[1][i],P[2][i],P[3][i],P[4][i],P[5][i]);
33.
Peso[i]=Likelihood_Calc(Frame_Actual_sem_distorção,Partícula,Magnitude_So
bel,Ângulo_Sobel);
34. }
35. for(int z=0; z< N_Partículas;z++){ //Atualizar o vetor de Melhores
partículas – Top M
36. if(Peso[z]<=0) Peso[z]=0;
37. int índice_Menor = Distancia_menor(Peso_Melhores_Partículas);
//Índice cuja distância é menor no vetor – "Peso_Melhores_Partículas"
38. if(Peso[z]>= Peso_Melhores_Partículas[índice_Menor]){
39. Peso_Melhores_Partículas[0][índice_Menor]=P[0][z];
40. Peso_Melhores_Partículas[1][índice_Menor]=P[1][z];
41. Peso_Melhores_Partículas[2][índice_Menor]=P[2][z];
42. Peso_Melhores_Partículas[3][índice_Menor]=P[3][z];
43. Peso_Melhores_Partículas[4][índice_Menor]=P[4][z];
44. Peso_Melhores_Partículas[5][índice_Menor]=P[5][z];
45.  }
46. }
47. int fase = fase_análise(Peso_Melhores_Partículas); //Fase de
otimização de pose
48. if(fase==1){ //Fase de Inicialização
49. for(int z=0; z< N_Partículas;z++){
50. int índice_Maior = Distancia_maior(Peso_Melhores_Partículas);
51. if(número_iteração == 1 && Peso[z]<δ){ //Primeira iteração e valor
inferior a sigma
```

```
52.    P[0][z]    =    Peso_Melhores_Partículas    [0][índice_Maior]    +
RuídoGaussiano(inicial);
53.    P[1][z]    =    Peso_Melhores_Partículas    [1][índice_Maior]    +
RuídoGaussiano(inicial);
54.    P[2][z]    =    Peso_Melhores_Partículas    [2][índice_Maior]    +
RuídoGaussiano(inicial);
55.    P[3][z]    =    Peso_Melhores_Partículas    [3][índice_Maior]    +
RuídoGaussiano(inicial);
56.    P[4][z]    =    Peso_Melhores_Partículas    [4][índice_Maior]    +
RuídoGaussiano(inicial);
57.    P[5][z]    =    Peso_Melhores_Partículas    [5][índice_Maior]    +
RuídoGaussiano(inicial);
58. } else {
59. P[0][z] = P[0][z] + RuídoGaussiano(fase1);
60. P[1][z] = P[1][z] + RuídoGaussiano(fase1);
61. P[2][z] = P[2][z] + RuídoGaussiano(fase1);
62. P[3][z] = P[3][z] + RuídoGaussiano(fase1);
63. P[4][z] = P[4][z] + RuídoGaussiano(fase1);
64. P[5][z] = P[5][z] + RuídoGaussiano(fase1);
65. }
66.         if         ((atan(abs((P[0][z])/(P[2][z])))>0.35)         ||
(atan(abs((P[1][z])/(P[2][z])))>0.35) || (P[2][z]>=-0.5) || (P[2][z]<=-
100) ){ //Verifica se a partícula criada está dentro da frame
67. z--;
68.   }
69.  }
70.} else if(fase == 2){ //Fase de otimização grosseira
71. for(int z=0; z< NPartículas;z++){
72. int índice_Maior = Distancia_maior(Peso_Melhores_Partículas);
73. if(Peso[z]> Peso_Melhores_Partículas[índice_Maior]){
74. P[0][z] = P[0][z] + RuídoGaussiano(fase2);
75. P[1][z] = P[1][z] + RuídoGaussiano(fase2);
76. P[2][z] = P[2][z] + RuídoGaussiano(fase2);
77. P[3][z] = P[3][z] + RuídoGaussiano(fase2);
78. P[4][z] = P[4][z] + RuídoGaussiano(fase2);
79. P[5][z] = P[5][z] + RuídoGaussiano(fase2);
80. } else {
81. if(mutação <= Relação_mutação){ // cruzamento – Escolha aleatória
dos parâmetros do Vetor Top M
82. mutação++; //Variável auxiliar que controla se é para existir ou não
mutação
83. P[0][z] = P[0][rand()%3]
```

```
84. P[1][z] = P[1][rand()%3];
85. P[2][z] = P[2][rand()%3];
86. P[3][z] = P[3][rand()%3];
87. P[4][z] = P[4][rand()%3];
88. P[5][z] = P[5][rand()%3];
89. } else { //Mutação = cruzamento + Ruído Gaussiano(mutação_fase2)
90. mutação =0; //Variável auxiliar igual a zero
91. P[0][z] = Peso_Melhores_Partículas[0][rand()%3] +
RuídoGaussiano(mutação_fase2);
92. P[1][z] = Peso_Melhores_Partículas[1][rand()%3] +
RuídoGaussiano(mutação_fase2);
93. P[2][z] = Peso_Melhores_Partículas[2][rand()%3] +
RuídoGaussiano(mutação_fase2);
94. P[3][z] = Peso_Melhores_Partículas[3][rand()%3] +
RuídoGaussiano(mutação_fase2);
95. P[4][z] = Peso_Melhores_Partículas[4][rand()%3] +
RuídoGaussiano(mutação_fase2);
96. P[5][z] = Peso_Melhores_Partículas[5][rand()%3] +
RuídoGaussiano(mutação_fase2);
97.  }
98. }
99. if ((atan(abs((P[0][z])/(P[2][z])))>0.35) ||
(atan(abs((P[1][z])/(P[2][z])))>0.35) || (P[2][z]>=-0.5) ||
(P[2][z]<=-100) ){ //Verifica se a partícula criada está dentro da
frame
100. z--;
101.   }
102.  }
103. } else { //Fase de otimização Fina
104. for(int z=0; z< N_Partículas;z++){
105. int índice_Maior = Distancia_maior(Peso_Melhores_Partículas);
//Índice cuja distância é maior no vetor – "Peso_Melhores_Partículas"
106. if(Peso[z]> Peso_Melhores_Partículas[índice_Maior]){
107. P[0][z] = P[0][z] + RuídoGaussiano(fase3);
108. P[1][z] = P[1][z] + RuídoGaussiano(fase3);
109. P[2][z] = P[2][z] + RuídoGaussiano(fase3);
110. P[3][z] = P[3][z] + RuídoGaussiano(fase3);
111. P[4][z] = P[4][z] + RuídoGaussiano(fase3);
112. P[5][z] = P[5][z] + RuídoGaussiano(fase3);
113. } else {
114. if(mutação <= Relação_mutação){ // cruzamento – Escolha aleatória
dos parâmetros do Vetor Top M
```

```
115. mutação++; //Variável auxiliar que controla se é para existir ou
não mutação
116. P[0][z] = P[0][rand()%3]
117. P[1][z] = P[1][rand()%3];
118. P[2][z] = P[2][rand()%3];
119. P[3][z] = P[3][rand()%3];
120. P[4][z] = P[4][rand()%3];
121. P[5][z] = P[5][rand()%3];
122. } else { //Mutação = cruzamento + Ruído Gaussiano(mutação_fase3)
123. mutação =0; //Variável auxiliar igual a zero
124. P[0][z] = Peso_Melhores_Partículas[0][rand()%3] +
RuídoGaussiano(mutação_fase3);
125. P[1][z] = Peso_Melhores_Partículas[1][rand()%3] +
RuídoGaussiano(mutação_fase3);
126. P[2][z] = Peso_Melhores_Partículas[2][rand()%3] +
RuídoGaussiano(mutação_fase3);
127. P[3][z] = Peso_Melhores_Partículas[3][rand()%3] +
RuídoGaussiano(mutação_fase3);
128. P[4][z] = Peso_Melhores_Partículas[4][rand()%3] +
RuídoGaussiano(mutação_fase3);
129. P[5][z] = Peso_Melhores_Partículas[5][rand()%3] +
RuídoGaussiano(mutação_fase3);
130.  }
131. }
132. if ((atan(abs((P[0][z])/(P[2][z])))>0.35) ||
(atan(abs((P[1][z])/(P[2][z])))>0.35) || (P[2][z]>=-0.5) ||
(P[2][z]<=-100) ){ //Verifica se a partícula criada está dentro da
frame
133. z--;
134.   }
135.  }
136. }
137. número_iteração++; //Variável que guarda o número da iteração
atual
138. if(fase==3){ //Se estamos na fase de otimização fina – mostrar
resultado (output)
139. int Maior = Distancia_maior(Peso_Melhores_Partículas);
140. Resultado = Mostrar(P[0][Maior],P[1][Maior],P[2][
Maior],P[3][Maior],P[4][Maior],P[5][Maior];
141. Janela_Windows(Resultado);
142. LimparVetorPeso(Peso, N_Partículas);
143. }
144. return 22305; }}
```

D.2. ARQUITETURA DE *TRACKING*

Descrição Simplificada: *Tracking* – Arquitetura Final

```
1. MAIN: //Ciclo Principal
2. Inicializar UKF();//O filtro é inicializado – Vetor de estado, Matriz de covariância, conforme descrito no subcapítulo 3.3.3. - Filtragem
3. Objeto = Ler_Objeto("UAV.obj"); //Ler objeto para RAM
4. BaseDados = Ler_Base_de_dados("BaseDados.txt"); //Ler Base de dados para RAM
5. Classificador = Ler_Parâmetros_Classificador("LBP.xml"); //Ler Classificador treinado para RAM
6. Frame_Actual = Ler_Frame_Câmara();
7. Frame_Actual_sem_distorção = Distorção_Remap(Frame_Actual); //Corrigir distorção Radial & Tangencial
8. [Magnitude_Sobel, Ângulo_Sobel] = Sobel(Frame_Actual); //Sobel – Magnitude e Ângulo do Gradiente
9.do{ //Fica no loop até detetar um objecto utilizando o classificador
10. ROI_Objeto = Detetar_UAV(Frame_Actual_sem_distorção,Classificador);
11. } while(ROI_Objeto.size()==0);
12. Atual_Gray = Color_Gray(Frame_Actual_sem_distorção); //RGB para Gray
13. Keypoints_Frame = FAST_Keypoints(Atual _Gray); //Keypoints do algoritmo FAST
14. Keypoints_ROI = Selecionar_ROI(Keypoints_Frame); //Keypoints ROI (são os que interessam - objeto)
15. Bounding_Box = Rectangulo_min_Area(Keypoints_ROI); //Cálculo da Bounding Box
16. [Área,Ângulo,Largura,Altura,BBX,BBY]=Parâmetros(Bounding_Box); //Cálculo dos parâmetros da BB obtida
17. double AR = Largura/Altura; //Cálculo do Aspect Ratio
18. for(int h=0;h< Nbase_dados ;h++){ //Cálculo da distância euclidiana para cada valor da base de dados
19. Valor[h]=sqrt((pow((Ângulo-(BaseDadosAng[h])),2))+(pow((AR-(BaseDadosAR[h])),2)));
20. }
21. for(int q=0;q< Ninicialização ; q++){ //Cálculo dos parâmetros X,Y,Z, Alfa, Beta e Gama - inicialização
22. int índice_Valor_Maior = Maior(Valor);
23. Z_H = (sqrt((BaseDadosArea_Valor[índice_Valor_Maior])/Área)*4);
24. X_H = (Z_H*(BBX-CX))/fx;
25. Y_H = (Z_H*(BBY-CY))/fy;
26. P[0][q]= X_H;
27. P[1][q]= Y_H;
```

```
28. P[2][q]= Z_H;
29. P[3][q]= BaseDadosAlfa[índice_Valor_Maior];
30. P[4][q]= BaseDadosBeta[índice_Valor_Maior];
31. P[5][q]= BaseDadosGama[índice_Valor_Maior];
32. }
33. for(int p=0; p< Iteração; p++){ //Início do Filtro – Otimização da pose
34. for(int i=0; i< N_Partículas; i++){ //Cálculo da likelihood para cada partícula
35. Partícula = rendering(P[0][i],P[1][i],P[2][i],P[3][i],P[4][i],P[5][i]);
36. Peso[i]=Likelihood_Calc(Frame_Actual_sem_distorção,Partícula,Magnitude_Sobel,Ângulo_Sobel);
37. }
38. for(int z=0; z< N_Partículas;z++){ //Atualizar o vetor de Melhores partículas – Top M
39. if(Peso[z]<=0) Peso[z]=0;
40. int índice_Menor = Distancia_menor(Peso_Melhores_Partículas); //Índice cuja distância é menor no vetor – "Peso_Melhores_Partículas"
41. if(Peso[z]>= Peso_Melhores_Partículas[índice_Menor]){
42. Peso_Melhores_Partículas[0][índice_Menor]=P[0][z];
43. Peso_Melhores_Partículas[1][índice_Menor]=P[1][z];
44. Peso_Melhores_Partículas[2][índice_Menor]=P[2][z];
45. Peso_Melhores_Partículas[3][índice_Menor]=P[3][z];
46. Peso_Melhores_Partículas[4][índice_Menor]=P[4][z];
47. Peso_Melhores_Partículas[5][índice_Menor]=P[5][z];
48.  }
49. }
50. int fase = fase_análise(Peso_Melhores_Partículas); //Fase de otimização de pose
51. if(fase==1){ //Fase de Inicialização
49. for(int z=0; z< N_Partículas;z++){
50. int índice_Maior = Distancia_maior(Peso_Melhores_Partículas);
51. if(número_iteração == 1 && Peso[z]<δ){ //Primeira iteração e valor inferior a sigma
52. P[0][z] = Peso_Melhores_Partículas [0][índice_Maior] + RuídoGaussiano(inicial);
53. P[1][z] = Peso_Melhores_Partículas [1][índice_Maior] + RuídoGaussiano(inicial);
54. P[2][z] = Peso_Melhores_Partículas [2][índice_Maior] + RuídoGaussiano(inicial);
```

```
55.    P[3][z]    =    Peso_Melhores_Partículas    [3][índice_Maior]    +
RuídoGaussiano(inicial);
56.    P[4][z]    =    Peso_Melhores_Partículas    [4][índice_Maior]    +
RuídoGaussiano(inicial);
57.    P[5][z]    =    Peso_Melhores_Partículas    [5][índice_Maior]    +
RuídoGaussiano(inicial);
58. } else {
59. P[0][z] = P[0][z] + RuídoGaussiano(fase1);
60. P[1][z] = P[1][z] + RuídoGaussiano(fase1);
61. P[2][z] = P[2][z] + RuídoGaussiano(fase1);
62. P[3][z] = P[3][z] + RuídoGaussiano(fase1);
63. P[4][z] = P[4][z] + RuídoGaussiano(fase1);
64. P[5][z] = P[5][z] + RuídoGaussiano(fase1);
65. }
66.         if         ((atan(abs((P[0][z])/(P[2][z])))>0.35)         ||
(atan(abs((P[1][z])/(P[2][z])))>0.35) || (P[2][z]>=-0.5) || (P[2][z]<=-
100) ){ //Verifica se a partícula criada está dentro da frame
67. z--;
68.   }
69.  }
70.} else if(fase == 2){ //Fase de otimização grosseira
71. for(int z=0; z< NPartículas;z++){
72. int índice_Maior = Distancia_maior(Peso_Melhores_Partículas);
73. if(Peso[z]> Peso_Melhores_Partículas[índice_Maior]){
74. P[0][z] = P[0][z] + RuídoGaussiano(fase2);
75. P[1][z] = P[1][z] + RuídoGaussiano(fase2);
76. P[2][z] = P[2][z] + RuídoGaussiano(fase2);
77. P[3][z] = P[3][z] + RuídoGaussiano(fase2);
78. P[4][z] = P[4][z] + RuídoGaussiano(fase2);
79. P[5][z] = P[5][z] + RuídoGaussiano(fase2);
80. } else {
81. if(mutação <= Relação_mutação){ // cruzamento - Escolha aleatória
dos parâmetros do Vetor Top M
82. mutação++; //Variável auxiliar que controla se é para existir ou não
mutação
83. P[0][z] = P[0][rand()%3]
84. P[1][z] = P[1][rand()%3];
85. P[2][z] = P[2][rand()%3];
86. P[3][z] = P[3][rand()%3];
87. P[4][z] = P[4][rand()%3];
88. P[5][z] = P[5][rand()%3];
```

```
89. } else { //Mutação = cruzamento + Ruído Gaussiano(mutação_fase2)
90. mutação =0; //Variável auxiliar igual a zero
91. P[0][z] = Peso_Melhores_Partículas[0][rand()%3] +
RuídoGaussiano(mutação_fase2);
92. P[1][z] = Peso_Melhores_Partículas[1][rand()%3] +
RuídoGaussiano(mutação_fase2);
93. P[2][z] = Peso_Melhores_Partículas[2][rand()%3] +
RuídoGaussiano(mutação_fase2);
94. P[3][z] = Peso_Melhores_Partículas[3][rand()%3] +
RuídoGaussiano(mutação_fase2);
95. P[4][z] = Peso_Melhores_Partículas[4][rand()%3] +
RuídoGaussiano(mutação_fase2);
96. P[5][z] = Peso_Melhores_Partículas[5][rand()%3] +
RuídoGaussiano(mutação_fase2);
97.  }
98. }
99. if ((atan(abs((P[0][z])/(P[2][z])))>0.35) ||
(atan(abs((P[1][z])/(P[2][z])))>0.35) || (P[2][z]>=-0.5) || (P[2][z]<=-
100) ){ //Verifica se a partícula criada está dentro da frame
100. z--;
101.   }
102.  }
103. } else { //Fase de otimização Fina
104. for(int z=0; z< N_Partículas;z++){
105. int índice_Maior = Distancia_maior(Peso_Melhores_Partículas);
//Índice cuja distância é maior no vetor – "Peso_Melhores_Partículas"
106. if(Peso[z]> Peso_Melhores_Partículas[índice_Maior]){
107. P[0][z] = P[0][z] + RuídoGaussiano(fase3);
108. P[1][z] = P[1][z] + RuídoGaussiano(fase3);
109. P[2][z] = P[2][z] + RuídoGaussiano(fase3);
110. P[3][z] = P[3][z] + RuídoGaussiano(fase3);
111. P[4][z] = P[4][z] + RuídoGaussiano(fase3);
112. P[5][z] = P[5][z] + RuídoGaussiano(fase3);
113. } else {
114. if(mutação <= Relação_mutação){ // cruzamento – Escolha aleatória
dos parâmetros do Vetor Top M
115. mutação++; //Variável auxiliar que controla se é para existir ou não
mutação
116. P[0][z] = P[0][rand()%3]
117. P[1][z] = P[1][rand()%3];
118. P[2][z] = P[2][rand()%3];
119. P[3][z] = P[3][rand()%3];
```

```
120. P[4][z] = P[4][rand()%3];
121. P[5][z] = P[5][rand()%3];
122. } else { //Mutação = cruzamento + Ruído Gaussiano(mutação_fase3)
123. mutação =0; //Variável auxiliar igual a zero
124. P[0][z] = Peso_Melhores_Partículas[0][rand()%3] +
RuídoGaussiano(mutação_fase3);
125. P[1][z] = Peso_Melhores_Partículas[1][rand()%3] +
RuídoGaussiano(mutação_fase3);
126. P[2][z] = Peso_Melhores_Partículas[2][rand()%3] +
RuídoGaussiano(mutação_fase3);
127. P[3][z] = Peso_Melhores_Partículas[3][rand()%3] +
RuídoGaussiano(mutação_fase3);
128. P[4][z] = Peso_Melhores_Partículas[4][rand()%3] +
RuídoGaussiano(mutação_fase3);
129. P[5][z] = Peso_Melhores_Partículas[5][rand()%3] +
RuídoGaussiano(mutação_fase3);
130.  }
131. }
132. if ((atan(abs((P[0][z])/(P[2][z])))>0.35) ||
(atan(abs((P[1][z])/(P[2][z])))>0.35) || (P[2][z]>=-0.5) ||
(P[2][z]<=-100) ){ //Verifica se a partícula criada está dentro da
frame
133. z--;
134.   }
135.  }
136. }
137. número_iteração++; //Variável que guarda o número da iteração
atual
138. if(fase==3){ //Se estamos na fase de otimização fina - mostrar
resultado (output)
139. int Maior = Distancia_maior(Peso_Melhores_Partículas);
140. Resultado_Medição = Mostrar(P[0][Maior],P[1][Maior],P[2][
Maior],P[3][Maior],P[4][Maior],P[5][Maior]; //Mostrar resultado da
medição
141. Quaternião_medição_actual =
Converter_Quaternião(P[3][Maior],P[4][Maior],P[5][Maior]); //Converter
medição actual de Euler para Quaternião
142. Vetor_Estado = Filtragem_UKF();//Aplicar a filtragem - Filtro de
Kalman Unscented
143. Atitude_Estimada_Euler =
Conversão_Quaternião(Vetor_Estado.segmento<4>(3)); //Converter atitude
obtida de Quaterniões para Euler - OpenGL (mostrar resultado)
144. Resultado_Filtragem = Mostrar(Vetor_Estado[0], Vetor_Estado[1],
Vetor_Estado[2], Atitude_Estimada_Euler [0], Atitude_Estimada_Euler
[1], Atitude_Estimada_Euler [2]); //Mostrar Resultado da filtragem
145. Janela_Windows(Resultado);
```

```
146. LimparVetorPeso(Peso, N_Partículas);
147. }
148. return 22305; }}
```

(Página intencionalmente deixada em branco)