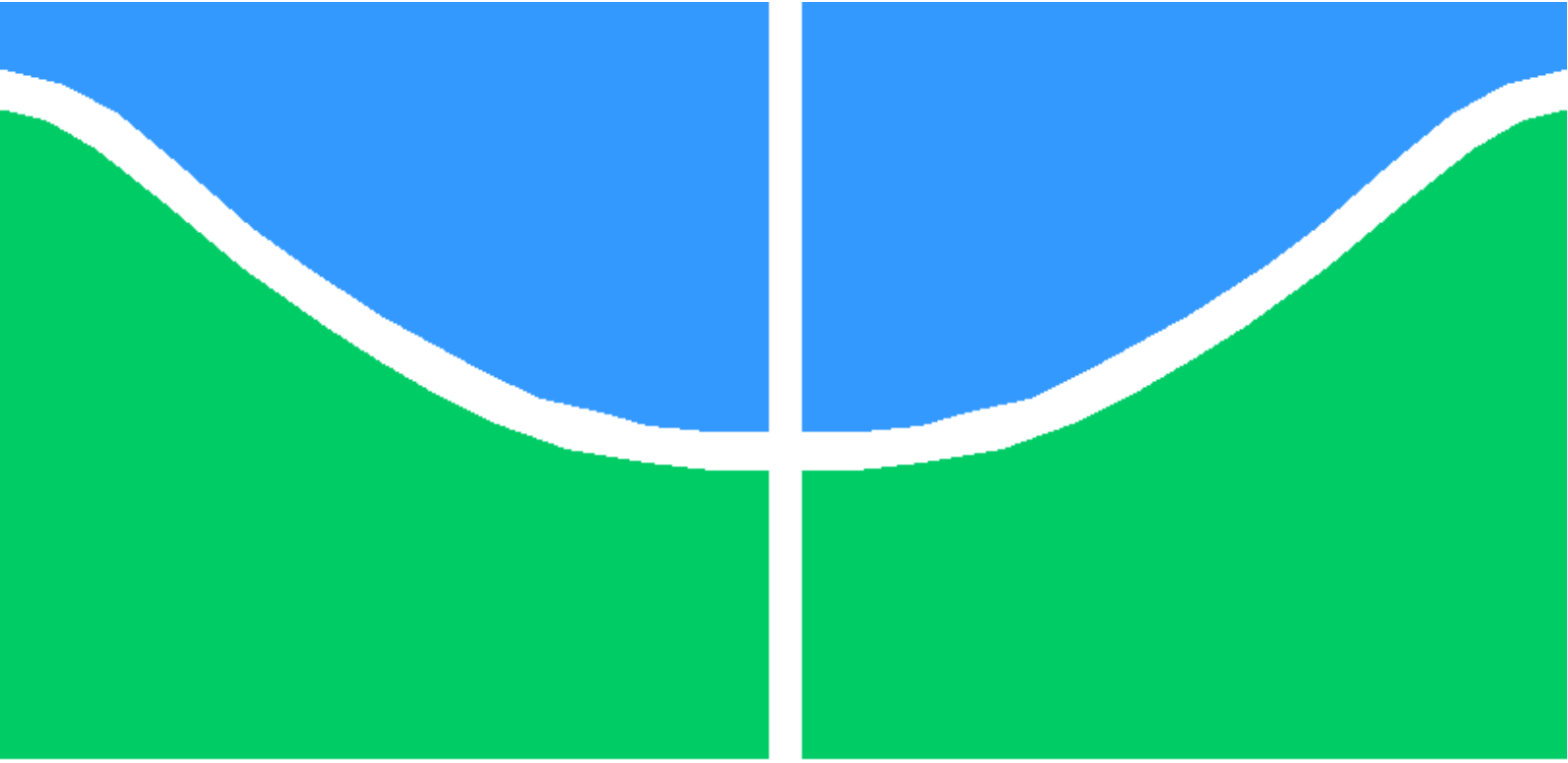


TRABALHO DE GRADUAÇÃO

ESTIMAÇÃO DE POSE POR VISÃO MONOCULAR PARA ATERRISSAGEM DE VANTs UTILIZANDO *ARTOOLKIT* COM MULTIMARCAS

Ronie Paulucio Porfirio

Brasília, junho de 2012

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

ESTIMAÇÃO DE POSE POR VISÃO MONOCULAR PARA ATERRISSAGEM DE VANTs UTILIZANDO *ARTOOLKIT* COM MULTIMARCAS

Ronie Paulucio Porfirio

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Banca Examinadora

PROF. JOÃO YOSHIYUKI ISHIHARA, ENE/UnB _____
Orientador

PROF. FLÁVIO DE BARROS VIDAL, CIC/UnB _____
Co-orientador

PROF. ADOLFO BAUCHSPIESS, ENE/UnB _____
Examinador interno

PROF. HENRIQUE CEZAR FERREIRA, ENE/UnB _____
Examinador interno

Dedicatória

À memória de meu avô Antonio Paulucio, referência de humildade e sabedoria, e meu avô Luis Afonso Porfirio, que conheci apenas quando criança e por meio das histórias de meus pais.

Ronie Paulucio Porfirio

Agradecimentos

A Deus, pela inspiração e por colocar em meu caminho pessoas que me auxiliaram nesta jornada, dessa maneira, ao meu lado constantemente.

A toda minha família, em especial aos meus pais, Rommel Ferreira Porfírio e Aparecida Paulucio Porfirio, pelo apoio e esforço na minha formação e o amor sem o qual eu não estaria realizando este sonho.

Aos professores João Ishihara, Flávio Vidal e Henrique Cezar pela paciência e orientações nas tantas reuniões semanais sempre me direcionando para o melhor caminho.

Aos amigos da Unb, em especial Hugo Tadashi e George Brindeiro, pelos esclarecimentos e contribuições de informações, sem os quais a realização deste se tornaria mais árdua.

E a todos amigos que fiz durante minha vida universitária e contribuíram direta ou indiretamente para a realização deste trabalho, os meus sinceros agradecimentos.

Ronie Paulucio Porfirio

RESUMO

Veículos aéreos não tripulados (VANTs) são sistemas robóticos que despertam o interesse de instituições civis e militares face as inúmeras aplicações dessa tecnologia. Dentre as atividades no uso de VANTs o processo de aterrissagem ainda é uma das atividades que envolvem cuidado especial. Neste trabalho é apresentada a proposta de uma metodologia para a estimação de pose usando visão monocular utilizando *ARToolKit* multimarcas para auxiliar no processo de pouso de VANTs. São apresentados resultados dos experimentos realizados comprovando a eficiência da metodologia proposta.

ABSTRACT

Unmanned aerial vehicles (UAVs) are robotic systems that arise great interest in civil and military institutions because of many possibilities of the applications of this technology. Among the many tasks performed during these applications, the landing process is either one of the that deserves special attention. In this paper we propose a methodology for height estimation based on a monocular vision system using a multimark ARToolKit resources to aid in UAVs landing process. We present experimental results which confirm the efficiency of the proposed methodology.

SUMÁRIO

1 Introdução 1
1.1 Contextualização 1
1.2 Objetivos 2
1.3 Resultados alcançados 3
1.4 Apresentação do manuscrito 3

2 Fundamentação Teórica 4
2.1 Câmeras e calibração 4
2.1.1 Modelo de Câmeras 4
2.1.2 Geometria de Projeção Básica 6
2.1.3 Distorção de Lentes 6
2.1.4 Calibração 6
2.2 Pose de VANT 7
2.3 Transformações de Coordenadas 8
2.3.1 Translação 3D 8
2.3.2 Rotações *Yaw*, *Pitch* e *Roll* 9
2.3.3 Determinando *Yaw*, *Pitch* e *Roll* a partir de uma matriz de rotação .. 10
2.3.4 Matriz de transformação homogenia para corpos rígidos 11
2.3.5 Transformações entre *frames* 12

3 Softwares utilizados 14
3.1 Introdução 14
3.2 A biblioteca *openCV* 14
3.3 A biblioteca *ARToolKit* 14
3.3.1 Limitações 16
3.4 Programas Auxiliares 17
3.4.1 *Câmera Calibration Toolbox for MATLAB* 17
3.4.2 Pattern Maker 17

4 Hardware utilizado 19
4.1 Requisitos Computacionais 19
4.2 VANT *AR.Drone* 19
4.2.1 Descrição do hardware do *AR.Drone* 20
4.2.2 Descrição do SDK do *AR.Drone* 21
4.2.3 Descrição do programa *AutoPylot* 22

5 Sistema de visão computacional desenvolvido 23
5.1 Pista de Pouso 23
5.2 Metodologia 24
5.2.1 Sistemas de Coordenadas 25
5.2.2 Estimando a pose 26
5.3 Fusão dos dados 29
5.3.1 Fusão multimarcas para todos os parâmetros da pose 29
5.3.2 Fusão geométrica para o parâmetro z 31
5.3.3 Fusão final para o parâmetro z 32

6 Resultados Experimentais 33
6.1 Classificação dos ensaios 33
6.2 Parâmetros x e y 34
6.3 Parâmetro z (Altura) 34
6.3.1 Resultados para ensaios estáticos 36
6.3.2 Resultados para ensaios contínuos 37
6.3.3 Resultados para ensaios durante voo 40
6.4 Parâmetro α (Guinada ou *YAW*) 40
6.4.1 Resultados para ensaios durante voo 41
6.5 Parâmetro β (Arfagem ou *PITCH*) 42
6.5.1 Resultados para ensaios estáticos 42
6.5.2 Resultados para ensaios durante voo 42
6.6 Parâmetro γ (Rolagem ou *ROLL*) 43
6.6.1 Resultados para ensaios estáticos 44
6.6.2 Resultados para ensaios durante voo 44

7 Conclusões 45

REFERÊNCIAS BIBLIOGRÁFICAS 46

Anexos 49

I Pista de pouso 50
I.1 Construção 50
I.2 Marcas de *ARToolKit* 54
I.3 Disposição geométrica das marcas 57
I.4 Configuração final 58

II Programa vantvision 59
II.1 Implementação do Software 59
II.1.1 Introdução 59
II.1.2 Metodologia 59
II.1.3 Estratégia de programação 60
II.2 Algoritmos desenvolvidos 61
II.2.1 vantvision, *autopilot* e *modVision* 61

II.2.2 *modDetectaM* ... 62
II.2.3 modProcessa ... 62
II.2.4 *modGraph* ... 63
II.2.5 *modShow* ... 66
II.2.6 *modSpeak* ... 67
II.2.7 *modShell* ... 67
II.3 Descrição do sistema de visão ... 68
II.3.1 Iniciado pelo processo *vantvision* ... 68
II.3.2 Iniciado pelo processo *autopilot* e pelo processo *modVision* ... 69
II.3.3 Redefinindo a prioridade dos processos ... 72
II.3.4 Finalização do sistema de visão ... 73
II.4 Conclusão ... 75

III Programa autopilot ... 76
III.1 Compilação e Execução ... 76
III.2 Controle ... 77

IV Conteúdo do CD ... 79
IV.1 Pasta doc ... 79
IV.2 Pasta prog ... 80
IV.2.1 Pasta criaCalibraDat ... 80
IV.2.2 Pasta criaSerieImagens ... 80
IV.2.3 Pasta modDetectaBasico ... 80
IV.2.4 Pasta modDetecta ... 80
IV.2.5 Pasta vantvision ... 80
IV.3 Pasta sistema ... 81
IV.3.1 Pasta ardrone_autopylot ... 81
IV.3.2 Pasta ARDrone_SDK_Version_1_8_20110726 ... 81
IV.3.3 Pasta ARToolKit ... 81
IV.3.4 Pasta ARToolKit Install Files ... 81
IV.3.5 Pasta patternMaker ... 81
IV.4 Pasta videos ... 81

V Montagem experimental ... 83
V.1 Ensaios dos parâmetros da pose ... 83
V.2 Ensaios do parâmetro z com a régua ... 84

VI Artigo ... 86

LISTA DE FIGURAS

2.1 Modelo de câmera *pinhole*. Retirado de [14]. 5
2.2 Modelo de câmera *pinhole* rearranjado. Retirado de [14]. 5
2.3 Sequência de imagens para calibração da câmera do VANT usada no trabalho. 7
2.4 Pose do VANT em relação à pista de pouso. 8
2.5 Rotações yaw, pitch e roll. Retirado de [17]. 9

3.1 Marca de *ARToolKit* 15
3.2 Marcador de *ARToolKit* 15
3.3 Imagem mostrando um cubo virtual preso no marcador de *ARToolKit* 16
3.4 Cubo virtual e referencial desenhados sobre o marcador. 17
3.5 Cubo virtual e referencial desenhados com o eixo z invertido. 18

4.1 VANT *Parrot AR.Drone* escolhido para testar o sistema. 19
4.2 Altímetro ultrassônico do *Parrot AR.Drone*. Retirado de [22]. 20
4.3 Câmera frontal do *Parrot AR.Drone*. Retirado de [22]. 21
4.4 Câmera vertical do *Parrot AR.Drone*. Retirado de [22]. 21

5.1 Pista de pouso 23
5.2 Índices atribuídas a cada marca de ARToolKit que compõe a pista de pouso. 24
5.3 *Frame* na pista de pouso. 25
5.4 *Frame* na marca central M0. 25
5.5 *Frame* na marca central M0 (outro ângulo de vista). 26
5.6 *Frame* na câmera. Retirado de [19] 26
5.7 *Frame* na câmera vertical do *AR.Drone* 26
5.8 *Frame* no VANT. 27
5.9 Matrizes de transformação *transPistaToMarca*, *transMarcaToBelly* e *transBellyToQuad*. 28
5.10 Matriz de transformação *transPistaToQuad*. 28

6.1 Ensaio estático do parâmetro z - Medidas absolutas 36
6.2 Ensaio estático do parâmetro z - Erros absolutos. 36
6.3 Ensaio estático do parâmetro z - Erros percentuais. 37
6.4 Ensaio estático do parâmetro z - Marcas detectadas. 37
6.5 Ensaio estático do parâmetro z - Cubo. 38
6.6 Ensaio 1 contínuo do parâmetro z. 38
6.7 Ensaio 2 contínuo do parâmetro z 39

6.8 Ensaio 3 contínuo do parâmetro z 40
6.9 Ensaio durante voo - altura 40
6.10 Ensaio contínuo dos ângulos de orientação com o *AR.Drone* em repouso..... 41
6.11 Ensaio durante o voo - guinada 42
6.12 Ensaio estático - arfagem 43
6.13 Ensaio durante o voo - arfagem 43
6.14 Ensaio estático e durante o voo - rolagem..... 44

I.1 Projeto da pista de pouso. Medidas em milímetros..... 51
I.2 Ângulos entre os centros das marcas. Medidas em graus..... 52
I.3 Distâncias entre os centros das marcas. Medidas em milímetros. 53
I.4 Índices atribuídas a cada marca de ARToolKit que compõe a pista de pouso. 54
I.5 Marca de *ARToolKit* de índice 0 também chamada de marca central..... 54
I.6 Marca de *ARToolKit* de índice 2 também chamada de marca interna esquerda..... 55
I.7 Marca de *ARToolKit* de índice 4 também chamada de marca interna traseira. 55
I.8 Marca de *ARToolKit* de índice 6 também chamada de marca interna direita..... 55
I.9 Marca de *ARToolKit* de índice 1 também chamada de marca externa frontal. 56
I.10 Marca de *ARToolKit* de índice 3 também chamada de marca externa esquerda. 56
I.11 Marca de *ARToolKit* de índice 5 também chamada de marca externa direita. 56
I.12 Pista de pouso final 58

II.1 Fluxograma geral 59
II.2 Fluxograma do processo vantvision 61
II.3 Fluxograma do processo *modDetectaM* 62
II.4 Fluxograma do processo modProcessa 63
II.5 Fluxograma do processo *modGraph* 63
II.6 Janelas mostrando parâmetros x, y, z do VANT encontrados. 64
II.7 Janelas mostrando parâmetros x, y, z do VANT perdidos. 64
II.8 Janelas mostrando as medidas angulares de parâmetros da pose. 65
II.9 Janela mostrando os índices das marcas detectadas num determinado momento. 65
II.10 Fluxograma do processo *modShow* 66
II.11 Janela do processo *modShow* mostrando o vídeo e linhas guia..... 66
II.12 Fluxograma do processo *modSpeak* 67
II.13 Fluxograma do processo *modShell* 68

III.1 Controle *DUALSHOCK* 3 utilizado para controlar a aeronave. 77
III.2 Controle *DUALSHOCK* 3 - vista dos botões R1,R2,L1 e L2..... 77

V.1 Montagem para experimentos 83
V.2 Elementos da montagem 84
V.3 Régua 85
V.4 Montagem experimental com régua. 85

LISTA DE TABELAS

3.1 Alcances em relação ao lado das envoltórias quadradas dos marcadores (cf. [19]). 17

5.1 Exemplo das distâncias reais entre os centros das marcas detectadas da pista de pouso. 32

I.1 Tabela mostrando os índices das marcas, posição, orientação e o número do *pattern* correspondente. 57

I.2 Tabela mostrando o tamanho e posição relativa das marcas na pista de pouso. 57

II.1 Eventos do processo *modSpeak* 67

II.2 Comandos implementados no processo *modShell* 68

III.1 Tabela mostrando os botões do controle e as funcionalidades correspondentes implementadas. 78

III.2 Tabela mostrando os eixos do controle e as funcionalidades correspondentes implementadas no modo normal. 78

III.3 Tabela mostrando os eixos do controle e as funcionalidades correspondentes implementadas no modo invertido. 78

LISTA DE SÍMBOLOS

Siglas

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
GNU	GNU's Not Unix
LARA	Laboratório de Automação e Robótica
SDK	Software Development Kit
VANT	Veículo Aéreo Não-Tripulado
UAV	Unmaned Aerial Vehicle

Capítulo 1

Introdução

1.1 Contextualização

Neste trabalho, propomos um sistema de medição da pose (localização e orientação) baseado em visão computacional que pode ser empregado como uma alternativa mais barata ou como subsistema auxiliar para os sistemas de medição de veículos aéreos não tripulados encontrados comercialmente. O subsistema proposto servirá para auxiliar os diversos sistemas encontrados nessas aeronaves.

Veículos aéreos não tripulados (VANTs) ou, em Inglês, *Unmanned Aerial Vehicles* (UAVs) são sistemas robóticos que despertam o interesse de instituições civis e militares face as inúmeras aplicações dessa tecnologia. Dentre elas podemos citar: Reconhecimento tático, apoio operacional e execução de missões em atividades militares [1]; patrulhamento e monitoração de regiões remotas, fronteiras e áreas costeiras [2]; segurança e apoio policial [3]; busca e salvamento; operação em zonas de perigo ou de desastres; mapeamento; prospecção mineral e arqueológica; levantamento de uso da terra em regiões rurais e urbanas para diagnóstico e planejamento; estudos agriculturais com predição de rendimento de colheitas; observação e contagem de rebanhos; detecção de incêndios e suporte à sua extinção [4, 5]; pesquisa ambiental e do clima, incluindo composição atmosférica e qualidade do ar, monitoração e avaliação de reservas florestais, manguezais, áreas marítimas e limnológicas; a inspeção de estruturas tais como os oleodutos, linhas da transmissão, represas e estradas [6]; monitoração e controle de transportes; telecomunicações e serviços de transmissão; além de outras.

Se por um lado já existem no mercado vários VANTs que funcionam predominantemente por controle remoto ou com grau limitado de autonomia, por outro lado é intenso o esforço em Pesquisa, Desenvolvimento e Inovação (PD&I), na academia e no setor industrial, visando o estabelecimento de VANTs substancialmente autônomos. Para se alcançar o nível desejado de autonomia para realização de tarefas das mais diversas complexidades em variados cenários de aplicação, um VANT deve ser capaz de decolar, navegar e pousar sem nenhuma interferência humana [7]. Dessa forma, um VANT autônomo obedece um sistema de controle que toma decisões em função de um conjunto de dados obtidos por diferentes tipos de sensores. Dentre os principais sensores utilizados em VANTs podemos destacar altímetros ultrassônicos, que utiliza pulsos de ultrassom para estimar

sua altura em relação ao chão, acelerômetros para determinar a velocidade da aeronave, girômetros para a inclinação, magnetômetros para a orientação, sensores GPS para determinar sua posição no espaço e uma ou mais câmeras para filmar o ambiente [8–11]. Contudo, em muitos casos as imagens provenientes dessas câmeras são subutilizadas: Enquanto a necessidade de um sensor diferente para cada variável eleva o preço final da aeronave, técnicas de visão computacional já permitem estimar essas variáveis usando apenas as imagens de uma câmera. Nesse contexto, essas técnicas se consagraram como uma fonte passiva, barata e rica em informações complementando esse conjunto de sensores presentes em VANTs de forma a tornar possível seu controle automático.

Dentre os principais sistemas de controle de VANTs auxiliados por visão, pode-se definir dois grandes grupos em que basicamente diferem entre si na utilização ou não de pistas de pouso pré-determinadas para a realização do pouso. Dentre os trabalhos que não utilizam pistas de pouso pré-definidas, têm-se em [12] o desenvolvimento de uma metodologia na qual faz-se o uso de imagens provenientes de um sistema de câmeras a partir da exploração das relações bidimensionais da homografia da imagem e os planos no domínio tridimensional. Nesta abordagem, a ausência de uma pista é compensada pelo uso de um sistema de duas ou mais câmeras, que inviabiliza a integração a VANTs de pequeno porte. No caso de pistas pré-definidas, podemos encontrar uma extensa gama de trabalhos como [13] que utiliza uma marca que consiste em um quadrado na cor preta dentro do qual há outros pequenos quadrados dispostos de forma a facilitar o processamento da imagem e a identificação das coordenadas do alvo. Mais recentemente, [8] utilizou um sistema de duas câmeras e uma marca que consiste em vários anéis concêntricos nas cores preto e branco que são identificados por uma metodologia que envolve técnicas de *template matching*. O trabalho encontrado em [9] utiliza a conhecida pista de pouso em formato "H" e um sistema de visão que estima a posição do VANT em relação a marca com exceção da altura que é obtida por um sensor GPS embarcado.

Esses trabalhos fazem uso de etapas semelhantes dentre os quais podemos citar: a conversão das imagens coloridas para imagens em escala de cinza, o *thresholding* na qual a imagem em escala de cinza é convertida para uma imagem monocromática, *contour extraction* e *corner detection* onde os contornos das imagens são detectados e ainda processos de *template matching* para procurar padrões previamente definidos. Contudo, em vez de seguirmos essa mesma linha de trabalho fizemos uso de uma ferramenta que faz esse processamento inicial e nos focamos em trabalhar essa tecnologia para a sua aplicação em estimação de pose para VANTs.

1.2 Objetivos

Neste trabalho, propomos um sistema de medição de pose baseado em visão monocular, combinando recursos da biblioteca *ARToolKit* com multimarcas para auxiliar no processo de pouso de VANTs em geral. O desafio é estimar a pose de um VANT utilizando somente imagens obtidas de uma única câmera acoplada ao VANT quando ela captura as imagens de uma pista de pouso especialmente construída para esse propósito.

1.3 Resultados alcançados

Como resultado desse trabalho, o artigo "ESTIMAÇÃO DE ALTURA POR VISÃO MONOCULAR UTILIZANDO *ARTOOLKIT* COM MULTIMARCAS PARA ATERRISSAGEM DE VANTS" foi submetido à XIX edição do Congresso Brasileiro de Automática - CBA 2012, promovido pela Sociedade Brasileira de Automática (SBA). Uma cópia desse artigo é apresentado no anexo VI. Nesse artigo nos limitamos a estimar a altura, mas nesse manuscrito a estimação se extendeu para todos os demais parâmetros da pose.

1.4 Apresentação do manuscrito

Iniciamos o manuscrito introduzindo no capítulo 2 os conceitos e fundamentos teóricos necessários para entender o que foi realizado. Em seguida, o capítulo 3 descreve os softwares utilizados para desenvolver o projeto ressaltando as bibliotecas utilizadas, suas características e limitações. No capítulo 4 apresentamos o hardware utilizado com destaque especial para descrever o VANT comercial *Parrot AR.Drone* utilizado para testar o sistema. Apresentamos no capítulo 5 o processo de obtenção da pose utilizando visão. No capítulo 6 abordamos os experimentos realizados para comprovar a validade do sensor e, finalmente, no capítulo 7 discorremos a respeito das conclusões finais desse trabalho. No final do manuscrito apresentamos uma série de anexos documentando o sistema. O anexo I detalha a pista de pouso. Os anexos II e III apresentam os programas que integram o sistema de visão e o programa *autopilot* usado para testar o sistema junto ao VANT. O conteúdo do CD é descrito no anexo IV. No anexo V descrevemos como os experimentos foram realizados e uma cópia do artigo científico resultante deste trabalho é apresentado no anexo VI.

Capítulo 2

Fundamentação Teórica

2.1 Câmeras e calibração

Como o sistema desenvolvido depende de uma câmera que capture as imagens, nada mais natural que estudar as características de uma câmera. Assim inicialmente apresentamos os conceitos básicos de modelos de câmeras abordando conceitos como a geometria de projeção e de distorções de lentes. Em seguida apresentamos o processo de calibração indispensável para poder relacionar o mundo real com as informações que são capturadas pela câmera. As informações a seguir foram retiradas de [14].

2.1.1 Modelo de Câmeras

O modelo de câmera mais simples existente é conhecido como modelo de câmera *pinhole*. Nesse modelo os raios de luz de qualquer ponto particular de um objeto distante são bloqueados por uma parede imaginária. Essa parede imaginária possui um pequeno furo pelo qual passa apenas um único raio de luz a partir de cada ponto do objeto. Cada raio de luz que passa pelo furo é então projetado numa superfície formando uma imagem. Como resultado, a imagem nesse plano (também conhecido como plano projetivo) está sempre em foco e o tamanho da imagem relativo à distância do objeto é dado por um único parâmetro da câmera: a distância focal. Nesse modelo idealizado de câmera, a distância entre o furo *pinhole* e o plano projetivo corresponde precisamente à distância focal. Isso é mostrado na figura 2.1 onde f é a distância focal da câmera, Z é a distância entre a câmera e o objeto, X é a altura do objeto e x é a altura da imagem no plano projetivo. Na figura 2.1 podemos verificar por semelhança de triângulos que $-x/f = X/Z$ ou seja:

$$-x = f\frac{X}{Z} \tag{2.1}$$

Em seguida, rearranjamos as posições do *pinhole* e do plano de projeção como é ilustrado na figura 2.2.

Nessa nova configuração da figura 2.2 o *pinhole* agora se torna o *centro de projeção* e com isso a imagem agora é a interseção entre os raios de luz e esse novo plano de projeção. O resultado disso

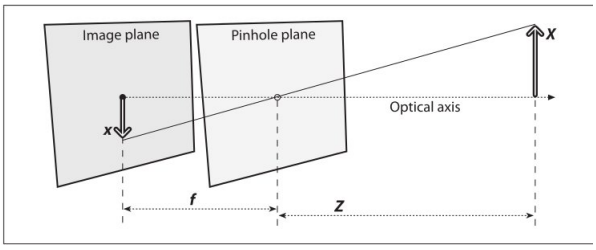


Figura 2.1: Modelo de câmera *pinhole*. Retirado de [14].

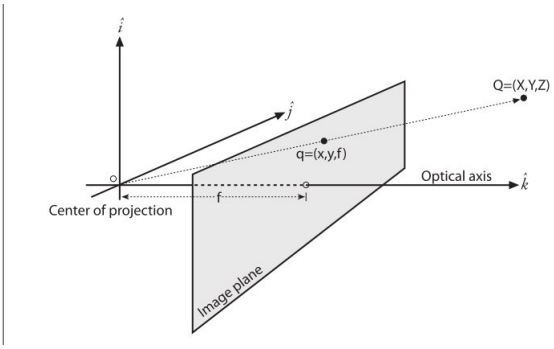


Figura 2.2: Modelo de câmera *pinhole* rearranjado. Retirado de [14].

é que a imagem aparece de cabeça para cima e por semelhança de triângulos a relação matemática passa a ser.

$$x = f\frac{X}{Z} \tag{2.2}$$

Uma observação importante é que o centro ótico não é equivalente ao centro da imagem podendo haver um diferença c_x entre suas abcissas e uma diferença c_y entre suas ordenadas.

O resultado dessas observações é que um modelo relativamente simples em que um ponto Q no mundo real que possui as coordenadas retangulares (X, Y, Z) é projetada no plano de projeção em alguma localização dada pelas relações:

$$x_{screen} = f_x\frac{X}{Z} + c_x \tag{2.3}$$

$$y_{screen} = f_y\frac{Y}{Z} + c_y \tag{2.4}$$

Verifique então que foram introduzidas distâncias focais nas direções x e y. A razão para isso é que os pixels individuais de câmeras típicas de baixo custo são retangulares em vez de quadradas.

A distância focal f_x, por exemplo, é o produto entre a distância focal física da lente pelo tamanho s_x do elemento gerador da imagem. Isso faz sentido quando verificamos que a unidade de s_x é de pixels por milímetro enquanto que F tem unidade de milímetros e daí f_x possui a unidade de pixels. O mesmo vale para f_y e s_y.

É importante constatar também que s_x e s_y não podem ser medidos diretamente por nenhum processo de calibração de câmera. O mesmo se aplica à distância focal física F que não pode ser

medido diretamente. Somente as combinações $f_x = Fs_x$ e $f_y = Fs_y$ podem ser encontradas pelo processo de calibração ao contrário da medida física de F que dependeria de desmontar a câmera e realizar uma medição direta.

2.1.2 Geometria de Projeção Básica

A relação que mapeia os pontos Q_i no mundo físico de coordenadas (X_i, Y_i, Z_i) aos pontos na imagem de projeção com coordenadas (x_i, y_i) é chamado de transformação projetiva ou projeção perspectiva. Quando trabalhamos com tais transformações fica mais conveniente utilizar as conhecidas coordenadas homogeneas [14].

As coordenadas homogeneas associadas com um ponto num espaço de projeção de dimensão n é expressa como um vetor de dimensão $n + 1$ (por exemplo, x, y, z se torna x, y, z, w com a restrição adicional de que cada dois pontos cujos valores são proporcionais são equivalentes.

Como no nosso caso o plano da imagem é o nosso espaço projetivo nós representaremos nesse plano com um vetor tridimensional $q = (q1, q2, q3)$. Relembrando que todos os pontos que possuem valores proporcionais no espaço projetivo são equivalentes, nós podemos recuperar as coordenas do pixel dividindo cada coordenada por $q3$. Isso nos permite encontrar os parâmetros que definem nossa câmera (no caso f_x, f_y,c_x e c_y) em uma única matriz três por três que é chamada de *matriz de parâmetros intrínsecos da câmera* ou simplesmente *matriz intrínseca da câmera*.

2.1.3 Distorção de Lentes

Em teoria é possível definir uma lente que não introduz distorções. Na prática nenhuma lente é perfeita. Assim, precisamos considerar as distorções da lente que ocorrem em dois tipos principais: Distorções radiais e tangenciais. A primeira ocorre em razão do formato das lentes enquanto que o segundo corre devido ao processo de montagem da câmera como um todo.

As lentes de câmeras reais distorcem os locais dos pixels perto das extremidades da lente. Esse fenômeno é o responsável pelo efeito conhecido como efeito olho de peixe. Dessa forma, à medida que nos afastamos radialmente do centro da lente, maior é a distorção e portanto ela é chamada distorção radial. A distorção tangencial ocorre devido a defeitos de montagem da câmera quando as lentes não estão perfeitamente paralelas ao plano de projeção.

2.1.4 Calibração

Como afirma [14] o processo de calibração de uma câmera nos dá tanto um modelo da geometria da câmera como um modelo de distorção de sua lente. Esses dois modelos definem os parâmetros intrínsecos da câmera e são usados para corrigir matematicamente as distorções de lente ao interpretar um cenário físico.

O processo de calibração faz uso de várias imagens de tabuleiros de xadrez (*chessboards*). A figura 2.3 mostra a sequência de imagens feitas desse tabuleiro com a câmera do VANT que foram usadas para calibrá-la.

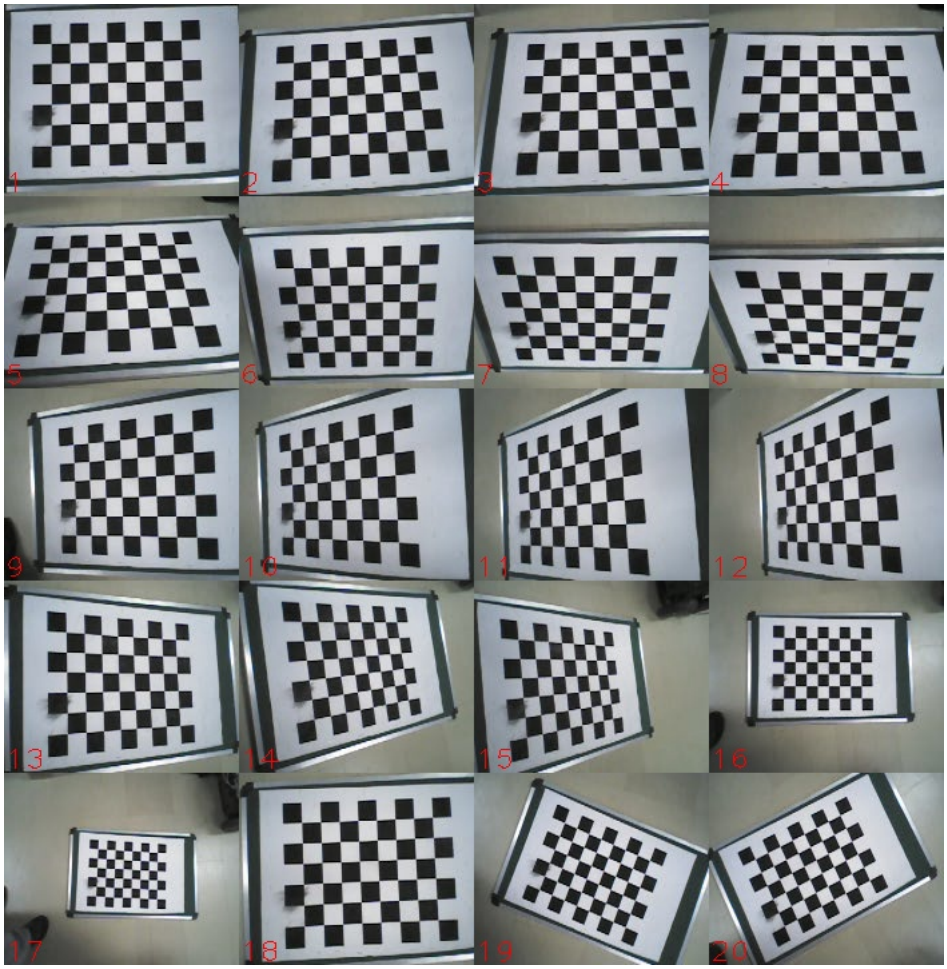


Figura 2.3: Sequência de imagens para calibração da câmera do VANT usada no trabalho.

Depois o algoritmo de calibração utiliza essas imagens para calcular os parâmetros intrínsecos da câmera. O algoritmo de calibração usado foi o *Camera Calibration Toolbox for MATLAB* que será descrito na seção 3.4.1 do capítulo 3, que descreve os programas utilizados nesse trabalho. Mais sobre esse algoritmo de calibração pode ser encontrado em [15].

2.2 Pose de VANT

Antes de definirmos o conceito de pose, precisamos primeiro definir o que é um *frame*. A palavra *frame* apareceu do termo em inglês *frame of reference* que significa quadro de referência. Ela é definida como uma uma base ortonormal e um ponto chamado de origem [16]. Colocando de forma simples, nesse trabalho o *frame* é um sistema de coordenadas cartesiano x, y, z cuja origem é colocado em algum ponto de algum objeto. Dessa forma, se o objeto se move ou sofre rotações o *frame* se move e rotaciona junto com ele. Geralmente para simplificar as análises (e adotamos isso nesse trabalho) um *frame* é colocado no centro geométrico do objeto e cada objeto possui seu *frame* associado. Nesse projeto há um *frame* fixo localizada no centro da pista de pouso, um *frame* em cada marca quadrada da pista, um *frame* na câmera vertical do VANT e outro no próprio VANT.

Dessa forma, podemos seguir e discutir brevemente o conceito de pose de um VANT. A pose de um VANT será definida como os parâmetros de localização e orientação do *frame* do VANT em relação ao *frame* da pista de pouso. Ou seja, a pose são os parâmetros de localização (x, y, z) e orientação (α, β, γ) do *frame* de um objeto em relação a outro. Os parâmetros de orientação correspondem respectivamente aos ângulos de guinada (YAW), arfagem ($PITCH$) e rolagem ($ROLL$) do VANT. Seus sentidos de crescimento obedecem à regra da mão direita. A figura 2.4 ilustra esses seis parâmetros da pose.

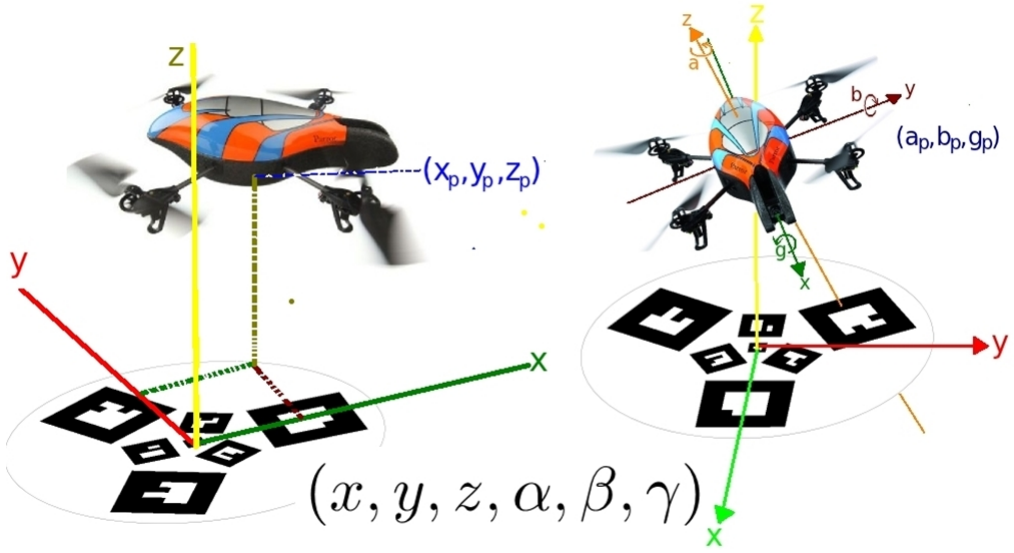


Figura 2.4: Pose do VANT em relação à pista de pouso.

Como veremos a seguir na seção 2.3 estamos interessados em relacionar um *frame* com o outro. Vale notar, finalmente, que muitas vezes a palavra "*frame*" fica subentendida no texto. Assim quando dizemos que queremos descobrir a pose do VANT em relação a pista, queremos na verdade a pose do *frame* fixo no VANT em relação ao *frame* fixo na pista.

2.3 Transformações de Coordenadas

Nos interessam a posição e orientação do VANT em relação a um referencial fixo na pista de pouso. Contudo, as informações fornecidas pelo *ARToolKit* é somente do *frame* da câmera em relação ao *frame* de uma determinada marca. Precisamos, portanto, realizar transformações de coordenadas para sair do *frame* fixo na pista de pouso, ir até a marca, da marca ir até a câmera e finalmente chegar no VANT. Nessa seção abordamos o estudo de transformações de coordenadas de um corpo rígido utilizados na determinação da pose do VANT. As informações a seguir foram retiradas de [17].

2.3.1 Translação 3D

Um corpo rígido A é transladado por algum deslocamento x_t, y_t, $z_t \in \Re$ usando:

$$(x, y, z) \mapsto (x + x_t, y + y_t, z + z_t). \tag{2.5}$$

Uma primitiva da forma

$$H_i = (x, y, z) \in W | f_i(x, y, z) \leq 0 \tag{2.6}$$

é transformada para

$$(x, y, z) \in W | f_i(x - x_t, y - y_t, z - z_t) \leq 0. \tag{2.7}$$

O corpo rígido transladado é chamado de $A(x_t, y_t, z_t)$.

2.3.2 Rotações *Yaw*, *Pitch* e *Roll*

Um corpo rígido pode ser rotacionado pelos três eixos ortogonais como já foi discutido na seção 2.2. A figura 2.5 recorda essas rotações. Essas rotações são conhecidas na literatura utilizando a notação, em inglês, *yaw*, *pitch* e *roll* que significam guinada, arfagem e rolagem, respectivamente.

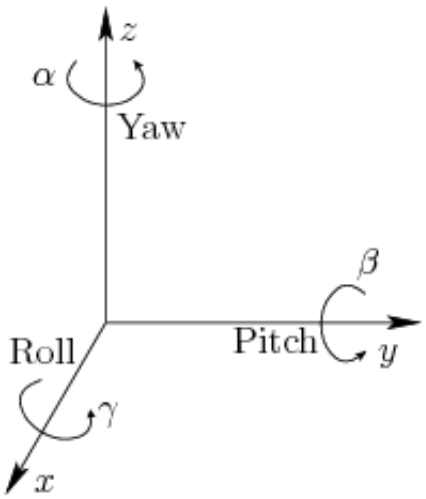


Figura 2.5: Rotações yaw, pitch e roll. Retirado de [17].

- Um *yaw* é um giro no sentido anti-horário de α em torno do eixo z. A matriz de rotação é dada por:

$$R_z(\alpha) = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.8}$$

- *pitch* é um giro no sentido anti-horário de β em torno do eixo y. A matriz de rotação é dada por:

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix}. \tag{2.9}$$

- *Roll* é um giro no sentido anti-horário de γ em torno do eixo x. A matriz de rotação é dada por:

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) \\ 0 & sin(\gamma) & cos(\gamma) \end{bmatrix}. \tag{2.10}$$

As rotações *yaw*,*pitch* e *roll* podem ser usadas para colocar um corpo 3D em qualquer orientação. Uma única rotação pode ser formada ao multiplicar as matrizes de rotação de *yaw*,*pitch* e *roll* obtendo:

$$R(\alpha, \beta, \gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma). \tag{2.11}$$

Daí, $R(\alpha, \beta, \gamma)$ pode ser escrita como:

$$\begin{bmatrix} \cos(\beta)\cos(\alpha) & -\cos(\gamma)\text{sen}(\alpha) + \text{sen}(\gamma)\text{sen}(\beta)\cos(\alpha) & \text{sen}(\gamma)\text{sen}(\alpha) + \cos(\gamma)\text{sen}(\beta)\cos(\alpha) \\ \cos(\beta)\text{sen}(\alpha) & \cos(\gamma)\cos(\alpha) + \text{sen}(\gamma)\text{sen}(\beta)\text{sen}(\alpha) & -\text{sen}(\gamma)\cos(\alpha) + \cos(\gamma)\text{sen}(\beta)\text{sen}(\alpha) \\ -\text{sen}(\beta) & \text{sen}(\gamma)\cos(\beta) & \cos(\gamma)\cos(\beta) \end{bmatrix}. \tag{2.12}$$

É importante destacar que $R(\alpha, \beta, \gamma)$ executa primeiro uma rotação *roll* para depois executar um *pitch* e finalmente um *yaw*. Se a ordem dessas operações for trocada, uma matriz de rotação diferente será obtida (cf. [17]).

2.3.3 Determinando *Yaw*, *Pitch* e *Roll* a partir de uma matriz de rotação

Muitas vezes é conveniente determinar os parâmetros α, β e γ diretamente de uma dada matriz de rotação. Suponha uma matriz de rotação arbitrária:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{2.13}$$

Comparando a matriz 2.13 com a matriz 2.12 encontramos equações que devem ser resolvidas para encontrar α, β e γ. Verifique que $r_{21}/r_{11} = tan(\alpha)$ e $r_{32}/r_{33} = tan(\gamma)$. Ainda, $r_{31} = -sin(\beta)$ e $\sqrt{r_{32}^2 + r_{33}^2} = cos(\beta)$. Resolvendo para cada ângulo, temos:

$$\alpha = tan^{-1}(r_{21}/r_{11}), \tag{2.14}$$

$$\beta = tan^{-1}(-r_{31}/\sqrt{r_{32}^2 + r_{33}^2}) \tag{2.15}$$

e

$$\gamma = tan^{-1}(r_{32}/r_{33}). \tag{2.16}$$

Existe a possibilidade de escolher dentre quatro quadrantes para as funções inversas da tangente. Cada quadrante deve ser escolhido olhando os sinais do número e denominador do argumento. O sinal do numerador define se a direção será acima ou abaixo do eixo x e o denominador define se a direção será à esquerda ou à direita do eixo y. Isso corresponde à função *atan2* da linguagem de programação C que expande o resultados da função tangente inversa no domínio de $[0, 2\pi)$ [17].

Assumindo $r_{11} \neq 0$ e $r_{33} \neq 0$ a função *atan2* pode ser usada para encontrar esses ângulos da seguinte forma:

$$\alpha = atan2(r_{21}, r_{11}), \tag{2.17}$$

$$\beta = atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \tag{2.18}$$

e

$$\gamma = atan2(r_{32}, r_{33}). \tag{2.19}$$

2.3.4 Matriz de transformação homogenia para corpos rígidos

Podemos reunir tudo o que foi visto até agora numa matriz chamada de matriz de transformação homogenia que possui informações de translação e rotação para corpos 3D e é utilizada na biblioteca *ARToolKit*.

Novamente, a ordem de operações são críticas. A matriz T representa a seguinte sequência de transformações:

1. *Roll* por γ
2. *Pitch* por β
3. *Yaw* por α
4. Translações por (x_t, y_t, z_t)

As primitivas do VANT podem ser transformadas para ter $T(x_t, y_t, z_t, \alpha, \beta, \gamma)$. Um corpo rígido capaz de translação e rotação tem, portanto, 6 graus de liberdade [17].

2.3.5 Transformações entre *frames*

Finalmente, podemos definir matrizes de transformações homogeneas que relacionam um referencial a um segundo e este a um terceiro e assim por diante de forma a podemos aplicar essas transformações em sequência para obter a matriz de transformação que relaciona o primeiro referencial com o último.

A mudança de um sistema de coordenadas m para outra c é apresentada na equação 2.20.

$$\begin{bmatrix} x_m \\ y_m \\ z_m \\ k_m \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{2.20}$$

e assim,

$$\mathbf{v_m} = T_c^m \mathbf{v_c}. \tag{2.21}$$

Podemos resumir a equação 2.20 no formato apresentado na equação 2.26 onde T_c^m é a matriz de transformação homogenia que transforma o sistema de coordenadas c no sistema m. Se por outro lado quisermos a transformação inversa, isto é, sair do sistema de coordenadas c e chegar ao sistema de coordenadas m, queremos a matriz T_m^c tal que:

$$\mathbf{v_c} = T_m^c \mathbf{v_m}. \tag{2.22}$$

É fácil demonstrar que:

$$T_M^B = (T_B^M)^{-1}. \tag{2.23}$$

Agora, se definimos um terceiro sistema de coordenadas, digamos, p e soubermos a transformação T_p^m que transforma de p para m, como obter a transformação que relaciona os sistemas c e p? A resposta é lógica:

$$\mathbf{v_c} = T_m^c \times T_p^m \mathbf{v_p}. \tag{2.24}$$

Assim, como:

$$\mathbf{v_c} = T_p^c \mathbf{v_p}, \tag{2.25}$$

temos:

$$T_p^c = T_m^c \times T_p^m. \tag{2.26}$$

Em suma, podemos obter transformações sequenciais multiplicando as matrizes de transformação de cada sistema em sequência. Isso será importante nesse projeto pois o *ARToolKit* fornece a matriz de transformação homogenia entre a câmera e a marca. Sabemos as matrizes entre a marca e a pista e entre a câmera e o VANT, logo para se obter a transformação entre o VANT e a pista faremos essas operações de transformação entre as matrizes disponíveis para encontrar a transformação procurada.

Capítulo 3

Softwares utilizados

3.1 Introdução

Nesse capítulo pretende-se introduzir as bibliotecas computacionais utilizadas para desenvolver o sistema. Inicialmente serão apresentadas funcionalidades da biblioteca *openCV* sobre a qual o sistema se apoiou. Em seguida, descrevemos a biblioteca *ARToolKit* utilizada para identificar e extrair informações das marcas que compõe a pista de pouso e concluindo o capítulo descrevendo, também, outros programas auxiliares utilizados no projeto.

3.2 A biblioteca *openCV*

O *OpenCV* é uma biblioteca *open source* de algoritmos de processamento de imagens desenvolvida inicialmente pela *Intel*. Esta biblioteca é composta por algoritmos otimizados de forma a utilizar os recursos do sistema para atingir menor custo computacional. Isso justifica escolher o *openCV* para aplicações que almejam alcançar resultados em tempo real. Essa biblioteca atualmente possui mais de 500 funções de processamento com aplicações dentre as quais se destacam: filtros, operações entre imagens, transformações morfológicas, calibração de câmeras, *tracking*, estimação de pose, reconhecimento e identificação de faces, gestos e objetos [14].

Escrita em C e C++ para os sistemas operacionais Linux, Windows e Mac OS X escolhemos utilizar o *openCV* como a plataforma principal para o desenvolvimento dos algoritmos do sistema. Assim, até mesmo as funções disponibilizadas por outras bibliotecas, como por exemplo o *ARToolKit*, foram adaptadas para funcionar juntamente com o *openCV*.

3.3 A biblioteca *ARToolKit*

A detecção das marcas da pista e a extração das informações de pose é feita usando recursos da biblioteca denominada *ARToolKit*.

Desenvolvida originalmente pelo Dr. Hirokazu Kato e mantida pelo *Human Interface Technology Laboratory (HIT Lab)* na *University of Washington* e também por uma série de outras

universidades [18], o *ARToolKit* é uma biblioteca originalmente utilizada para a criação de aplicações de realidade aumentada, nas quais envolvem a sobreposição de imagens virtuais no mundo real. Ela é distribuída para o público para uso não comercial sob a licença *GNU General Public License*. No entanto, ela deixou de ser mantida e atualizada.

De forma resumida, o funcionamento do *ARToolKit* ocorre a partir das seguintes etapas descritas em sua documentação [19]:

1. Captura-se o vídeo do mundo real e envia para o computador;
2. O software no computador busca, em cada quadro do vídeo, por marcadores pré-definidos;
3. Se um marcador é encontrado, o software calcula a posição da câmera em relação ao marcador;
4. Um modelo gráfico computacional é construído a partir dessa posição;
5. Este modelo é imerso no vídeo sobreposto ao mundo real.

O marcador consiste em uma imagem impressa em padrão pré-definido monocromático o qual é um padrão bidimensional. Para simplificar a notação vamos nos referir a este padrão bidimensional como **marca de *ARToolKit***. A figura 3.1 mostra um exemplo de marca de *ARToolKit* e a figura 3.2 mostra um marcador que utiliza essa marca.

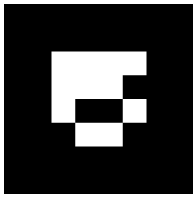


Figura 3.1: Marca de *ARToolKit*

Figura 3.2: Marcador de *ARToolKit*

No processo de execução do *ARToolKit* estamos interessados nos primeiros três passos que envolvem os processamentos de visão computacional. Neste caso são as rotinas de identificação e rastreamento dos marcadores.

Para ilustrar as caraterísticas imersivas da realidade aumentada mostramos na figura 3.3 um cubo sobre a imagem do marcador a partir da combinação das rotinas em conjunto com o *openCV* mencionadas anteriormente.

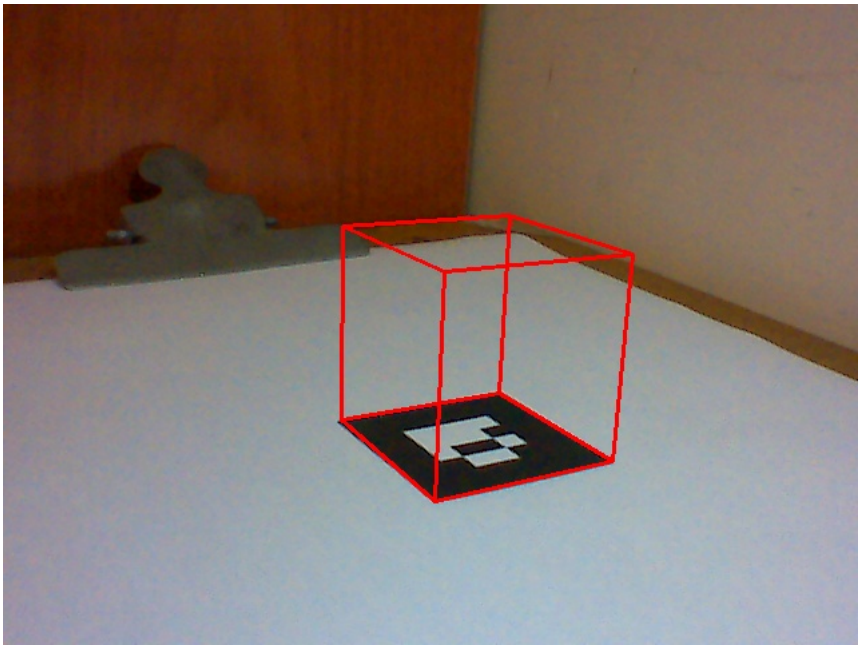


Figura 3.3: Imagem mostrando um cubo virtual preso no marcador de *ARToolKit*

3.3.1 Limitações

Embora aplicações menos exigentes do *ARToolKit* alcançem resultados satisfatórios com uma calibração da câmera com distorção simples. Para aplicações que necessitam de maior acurácia de medidas tridimensionais deve-se fazer uso de um processo de calibração mais refinado e complexo. Porém, mesmo calibrações de boa qualidade que buscam fazer uma estimativa robusta dos parâmetros de interesse são sujeitas a erros dada a interferência direta do usuário no posicionamento da câmera e do padrão de calibração [20].

Essas observações apresentadas em [21], que definem a relação entre o erro nas estimações dos parâmetros de posição e orientação da câmera em relação a uma única marca de *ARToolKit*, levam a crer que utilizar uma única marca não é aconselhável, tornando-se necessária a utilização de várias marcas para se alcançar resultados mais precisos.

Uma limitação direta é a que o marcador sempre deve estar no campo de visão da câmera. Se parte desse marcador for coberto, este não será identificado. Também existem faixas de distâncias em que o marcador não é detectado. Essa distância varia com o tamanho do lado do quadrado da marca de *ARToolKit*. A título de exemplo, a tabela 3.1 mostra a relação entre alguns valores do lado do quadrado do marcadores e os alcances máximos para os quais as rotinas de identificação deixam de detectar o marcador [19, 20].

Por fim, foi observado que para determinadas posições entre a câmera e o marcador ocorrem momentos em que as rotinas de estimação de pose do *ARToolKit* inverte a posição do eixo z. O efeito disso na imagem pode ser observado nas figuras 3.4 e 3.5 onde as informações do *ARToolKit* são usadas para desenhar um cubo dentro do qual há três segmentos que representam os eixos x, y e z do *frame* do marcador no mundo real.

Note que o cubo e o o eixo z do referencial da figura 3.5 encontram-se invertidos em relação aos

Lado [mm]	Alcance [mm]
69.85	406
99.90	635
107.95	834
187.20	1270

Tabela 3.1: Alcances em relação ao lado das envoltórias quadradas dos marcadores (cf. [19]).

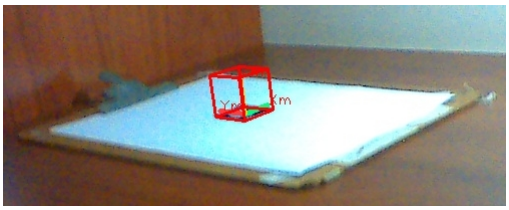


Figura 3.4: Cubo virtual e referencial desenhados sobre o marcador.

da figura 3.4. Esse erro ocorre a uma proporção pequena e apenas em determinadas posições da câmera em relação ao marcador e assim pode ser desprezado em aplicações de realidade aumentada. No entanto, para aplicações em sistemas de localização e orientação, este erro deve ser detectado e corrigido.

3.4 Programas Auxiliares

Foram utilizados programas auxiliares no desenvolvimento do projeto, listados a seguir:

3.4.1 *Câmera Calibration Toolbox for MATLAB*

O processo de calibração fornece tanto o modelo de geometria da câmera, bem como o modelo de distorção da lente da câmera. Esses dois modelos definem os parâmetros intrínsecos da câmera [14]. Essas informações são usadas para corrigir matematicamente as distorções de lente e auxiliar a interpretar um cenário físico.

Existem diversas ferramentas e metodologias usadas para identificar esses parâmetros de calibração. Tanto a biblioteca *ARToolKit* quanto a biblioteca *openCV* fornecem suas próprias rotinas e metodologias para calibração de câmeras.

Dessa forma, a calibração da câmera do *Parrot AR.Drone* foi realizada utilizando o *Camera Calibration Toolbox for MatLab* que é um conjunto de rotinas desenvolvidos pelo professor *Jean-Yves Bouguet* (consultar [15]) do Instituto de Tecnologia da Califórnia para calibração de câmeras.

3.4.2 Pattern Maker

As marcas de *ARToolKit* são originalmente desenhadas à mão e depois há um procedimento em que o software cria um arquivo chamado de *arquivo de treinamento* contendo as informações

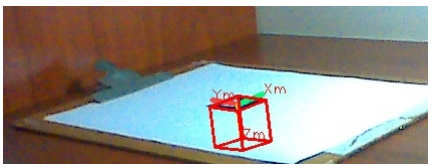


Figura 3.5: Cubo virtual e referencial desenhados com o eixo z invertido.

necessárias para posteriormente ser capaz de detectar essas marcas. No entanto, esse procedimento não é prático e é custoso de ser realizado.

Dessa forma, foi desenvolvido um algoritmo capaz de gerar e treinar marcas de *ARToolKit* em larga escala. Além disso, as marcas de *ARToolKit* geradas são claras o suficiente de forma a atingirem o melhor desempenho de reconhecimento pelos algoritmo de detecção do *ARToolKit*.

Essas marcas seguem restrições de formato e assim elas não são rotacionalmente simétricas. Do contrário, o *ARToolKit* não seria capaz de extrair informações de orientação do marcador. O *Pattern Maker* apresenta as vantagens descritas e justificam o interesse em usa-lo para gerar as marcas que foram utilizadas na pista de pouso desenvolvida.

Capítulo 4

Hardware utilizado

Nesse capítulo apresentamos o *hardware* utilizado nesse trabalho e serão descritos os requisitos que a arquitetura computacional deve possuir para executar o sistema desenvolvido, incluindo o VANT que será utilizado.

4.1 Requisitos Computacionais

O sistema de visão foi desenvolvido para ser executado em ambiente Linux. Ele foi testado em um computador com processador AMD Turion 64 X2 rodando sistema operacional Linux Ubuntu 11.10 32 bits versão *desktop*.

4.2 VANT *AR.Drone*

O sistema foi elaborado de forma a ser idenpendente de sistema operacional e arquitetura computacional, e assim capaz funcionar com qualquer VANT que utilize no seu controle qualquer tipo de sistema embarcado. No entanto, para comprovar sua validade foi preciso escolher uma plataforma para realizar os ensaios experimentais. A plataforma escolhida foi o quadricoptero comercial *PARROT AR.Drone* apresentado na figura 4.1

Figura 4.1: VANT *Parrot AR.Drone* escolhido para testar o sistema.

O VANT *AR.Drone* é um *quadrotor* comercial de dimensões $52,5$ x $51,5cm$ fabricado pela empresa *Parrot*. Podemos citar as seguintes vantagens de usar o *AR.Drone* como plataforma para testes do sistema de visão:

- O *AR.Drone* é um VANT de baixo custo e de fácil acessibilidade;
- Conjunto de sensores embarcados;
- O *AR.Drone* possui uma biblioteca aberta para comunicação e controle do mesmo. Dessa forma, o processamento das imagens de suas câmeras podem ser realizadas *stand-alone* por um computador.

4.2.1 Descrição do hardware do *AR.Drone*

Vamos descrever de forma resumida o hardware utilizado pelo *AR.Drone* enfatizando no conjunto de sensores embarcados.

4.2.1.1 Altímetro ultrassônico

O altímetro ultrassônico possui frequência de emissão de 40 KHz e frequência de recepção de 22,22 a 25 Hz [22]. A placa do altímetro ultrassônico é apresentado na figura 4.2.

Figura 4.2: Altímetro ultrassônico do *Parrot AR.Drone*. Retirado de [22].

4.2.1.2 Câmera Frontal (*Front Cam*)

A câmera frontal (figura 4.3) é uma câmera de resolução espacial modo *VGA* com 640 x 480 *pixels* a $30fps$ com ângulo de abertura de 93°. Não utilizamos essa câmera. As imagens da pista de pouso são obtidas com a câmera vertical descrita em seguida.

4.2.1.3 Câmera Vertical (*Belly Cam*)

A câmera vertical (ver figura 4.4), chamada nesse trabalho de *Belly Cam* por estar localizada na na parte inferior do *AR.Drone* é uma câmera de resolução QCIF (176 x 144 *pixels*), $60fps$ e ângulo de abertura de 64°.

Acrescentamos que embora essa câmera vertical do *AR.Drone* seja de elevada taxa de aquisição, ela possui baixa resolução se comparada, por exemplo, com a resolução de uma *WebCam*

Figura 4.3: Câmera frontal do *Parrot AR.Drone*. Retirado de [22].

Figura 4.4: Câmera vertical do *Parrot AR.Drone*. Retirado de [22].

típica. Desta forma, toda a implementação realizada com esta câmera, é de se esperar resultados superiores para câmeras de resolução espacial superior.

4.2.1.4 Sensores de orientação

A orientação é definida pelos ângulos α, β e γ já descritos na seção 2.2 que introduziu o conceito de pose. O ângulo α (ângulo de guinada) é dado por um girômetro de precisão de um eixo, embarcado no *AR.Drone*. Esse é um sensor magnetômetro que mede o ângulo de guinada em relação ao norte magnético (sul geográfico). Dessa forma, se a câmera frontal aponta para o norte magnético, teremos um ângulo α igual a $0°$. Ao girarmos o VANT para o leste, esse ângulo aumenta e diminui se girado para o oeste. No sul magnético há uma descontinuidade: quando o quadrotor gira mais de $180°$ a saída do mesmo altera para $-180°$. Assim a faixa de valores varia de $0°$ até $180°$ num sentido e de $0°$ até $-180°$ no outro [22, 23].

Os ângulos de rolagem γ e arfagem β são fornecidos por um girômetro de 2 eixos. Eles são nulos se o plano formado pelos eixos x e y do VANT é paralelo ao plano horizontal.

4.2.2 Descrição do SDK do *AR.Drone*

O SDK oficial do *AR.Drone* (cf. [24]) contém as bibliotecas *ARDroneLib* e *ARDroneTool* que compõe um *framework* que permitem escrever aplicações para comunicação e controle do *AR.Drone*.

Nesse trabalho nosso interesse principal são nos dados de navegação que o *AR.Drone* envia para o cliente usando uma estrutura de dados chamada *NAVDATA* além obviamente das imagens capturadas pela *belly cam*. O *NAVDATA* contém as informações dos sensores de altitude obtidas pelo altímetro ultrassônico e orientação (ângulos de arfagem, rolamento e guinada) da aeronave e é atualizada periodicamente a intervalos de tempo menores que $5ms$. Para maiores informações

recomendamos ler o manual do SDK em [24].

4.2.3 Descrição do programa *AutoPylot*

O programa *AutoPylot* foi desenvolvido originalmente pelo professor *Simon D. Levy* como um ponto de partida para que os programadores consigam desenvolver um piloto automático para o *Parrot AR.Drone*. Ele é um *framework* onde o programador possui as opções de implementar o controlador escolhendo dentre 3 diferente linguagens: linguagem C, linguagem *python* e até mesmo em *MatLab*.

Nesse projeto, modificamos o programa *AutoPylot* e criamos um programa chamado *autopilot* para poder fazer os testes necessários do sistema de visão desenvolvido. Essas alterações estão descritas no anexo III.

Capítulo 5

Sistema de visão computacional desenvolvido

Nesse capítulo será apresentado o sistema de visão que realiza a medição da pose de VANTs usando visão monocular e a abordagem de pista de pouso pré-definida. Inicialmente serão descrits as informações referentes à pista de pouso, em seguida a metodologia utilizada para estimar a pose e terminando com a proposta para o processo de fusão dos dados que fornecem a pose final.

5.1 Pista de Pouso

Com certeza um componente muito importante do sistema de visão desenvolvido é a pista de pouso para o VANT. Esta foi especialmente desenvolvida, de maneira tal, que seja possível utilizar os recursos de detecção e rastreamento provenientes da biblioteca *ARToolKit* em que cada marca específica fornece uma característica distinta da estimativa referentes à localização e orientação entre o VANT e a pista de pouso.

Para esta proposta de pista de pouso, foram utilizadas na construção sete marcas de *ARToolKit* com três tamanhos diferentes, conforme mostrado na figura I.12. No Anexo I podem ser encontrados maiores detalhes sobre a construção da pista de pouso.

A partir da figura I.12, as informações referentes às disposições geométricas da pista são:

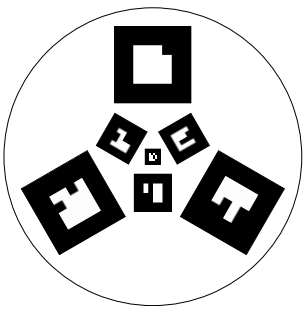


Figura 5.1: Pista de pouso

- Uma marca central (*M0*);
- Três marcas maiores externas (*M1, M3 e M5*);
- Três marcas de tamanhos intermediários (*M2, M4 e M6*).

Todos os índices de identificação foram atribuídos percorrendo o sentido anti-horário como apresentado na figura 5.2.

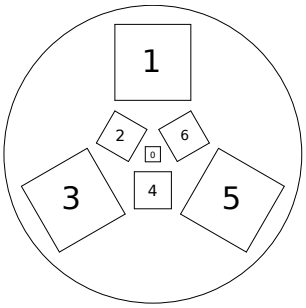


Figura 5.2: Índices atribuídas a cada marca de ARToolKit que compõe a pista de pouso.

Esta disposição foi concebida a partir das seguintes atribuições, a saber:

- **Detecção da marca e estimação da altura a partir de diferentes alturas** - A utilização de pelo menos três possíveis tamanhos de marcas faz com que o sistema de visão embarcada no VANT utilizado seja capaz de detectar ao menos uma dessas marcas, a partir de diferentes alturas. Os valores mínimos e máximos de altura dependem diretamente da resolução da câmera que é utilizada para adquirir as imagens.
- **Robustez à oclusão parcial** - Se ocorrer a oclusão parcial da pista de pouso, com pelo menos algumas marcas visíveis, o sistema ainda pode detectar as marcas remanescentes e estimar a altura a partir delas.
- **Fusão de dados para estimação da altura** - Cada marca detectada pelo sistema de visão informa seu próprio conjunto de variáveis de interesse referentes à orientação e localização do VANT. Essas informações podem ser utilizadas para obter a estimativa final da pose procurada.
- **Disposição das marcas** - As marcas externas e internas estão dispostas segundo uma geometria conhecida. Como todas as distâncias entre uma marca e outra são conhecidas na pista real e na imagem virtual, podemos usar essas informações juntamente com as informações dos parâmetros intrínsecos da calibração da câmera para calcular a altura da aeronave.

5.2 Metodologia

O objetivo desse trabalho é estimar a pose $(x, y, z, \alpha, \beta, \gamma)$ de um *frame* fixo no VANT em relação a um referencial fixo na pista de pouso. Essa seção apresenta a metodologia usada para atingir esse objetivo.

5.2.1 Sistemas de Coordenadas

Um assunto importante a ser abordado neste trabalho é com relação aos *frames* adotados. Estamos interessados em estimar a pose do VANT em relação à pista de pouso, para que desta forma seja possível utilizar esta informação extraída afim de realizar um procedimento de pouso sobre a pista pré-definida.

A figura 5.3 mostra a pista de pouso e seu respectivo *frame* desenhados na imagem da pista usando os algoritmos desenvolvidos de realidade aumentada. O eixo x (na cor verde) tem origem no centro da pista (que coincide com o centro da marca M0) e segue na direção do centro da marca M1. O eixo z (na cor amarela) é ortogonal ao plano da pista e a direção apontada para cima (ortogonal ao plano xy). Por fim o eixo y (na cor vermelha) é ortogonal aos dois primeiros eixos citados e seu sentido obedece à regra da mão direita.

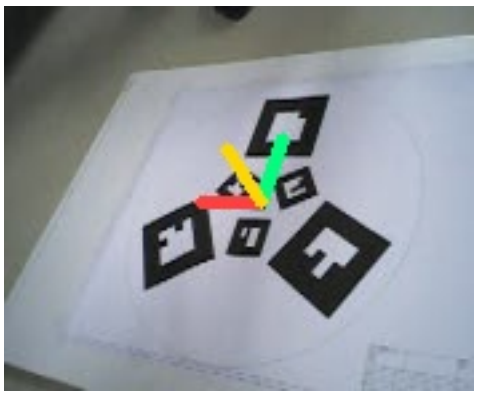


Figura 5.3: *Frame* na pista de pouso.

Cada marca apresentada possui seu próprio *frame* que pode ser entendido como um sistema de coordenadas local. Como exemplo tomamos como referência a marca central M0 apresentada nas figuras 5.4 e 5.5 que mostram o marcador central e os três eixos ortogonais desenhados em cima do mesmo.

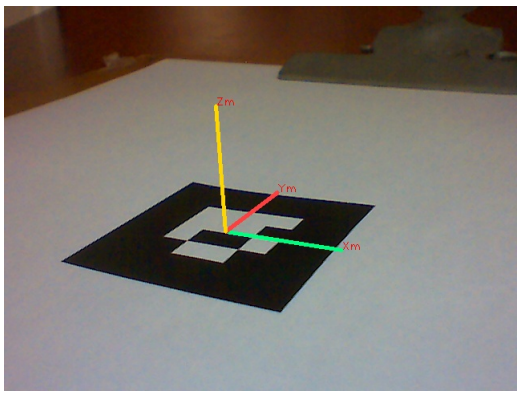


Figura 5.4: *Frame* na marca central M0.

As figuras 5.6 e 5.7 mostra o sistema de coordenadas utilizado pela câmera segundo a documentação do *ARToolKit*, ou seja, a origem fica no centro focal da imagem. Neste é considerado que o centro focal da imagem coincide com o centro do plano da imagem. O eixo z é ortogonal

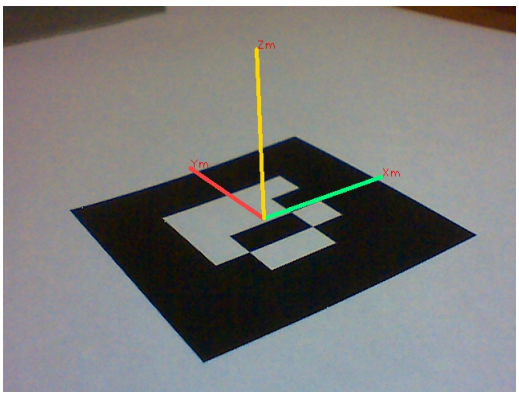


Figura 5.5: *Frame* na marca central M0 (outro ângulo de vista).

ao plano da imagem e cresce de forma a se afastar da câmera na mesma direção do eixo focal. O eixo x cresce a partir do centro para a direita em relação ao alinhamento do centro da imagem. O eixo y cresce a partir do centro para baixo, também em relação ao centro da imagem.

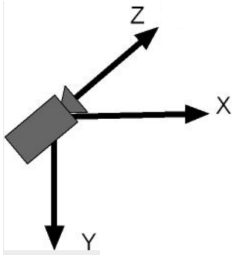


Figura 5.6: *Frame* na câmera. Retirado de [19]

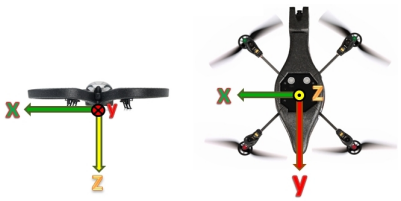


Figura 5.7: *Frame* na câmera vertical do *AR.Drone*

Quanto ao VANT, optou-se por atribuir o *frame* de acordo com a convenção clássica em que a origem fica no seu centro de gravidade com o eixo x a partir da origem e na mesma direção que o VANT percorre se voar para frente. O eixo z (ortogonal a x) sai do seu centro de gravidade e cresce na mesma direção em que o VANT ganha altitude e o eixo y é também ortogonal a esses eixos e segue o sentido dado pela regra da mão direita. A figura 5.8 ilustra essa descrição.

Finalmente, consideramos que a localização do centro de gravidade do VANT e da câmera vertical são coincidentes, ou seja, não há translação entre eles.

5.2.2 Estimando a pose

A pose que o *ARToolKit* fornece para uma determinada marca é fornecida a partir da marca em relação à câmera. São necessárias algumas transformações de coordenadas para obter a pose

Figura 5.8: *Frame* no VANT.

do VANT em relação ao frame da pista de pouso. Assim precisamos determinar as matrizes de transformação entre cada marca e a pista, entre a câmera vertical e o VANT, para em seguida realizar a partir do referencial na pista e, partir de transformações de coordenadas, chegar na posição do VANT. As matrizes de transformação são descritas a seguir:

1. ***transPistaToMarca*** - Para cada marca da pista de pouso há um matriz de transformação que relaciona o *frame* dessa marca ao *frame* da pista. Chamamos de *transPistaToMarca* essa matriz de transformação que fornece a pose da marca em relação à pista. Como são 7 marcas teremos ao todo sete matrizes deste tipo.

2. ***transMarcaToBelly*** - O *ARToolKit* fornece a pose da marca em relação à câmera vertical que aqui chamamos de *belly cam* por essa câmera estar localizada no dorso do quadrotor. Logo, de acordo com nossa notação o *ARToolKit* fornece a matriz de transformação *transBellyToMarca* que deve ser invertida para fornecer a matriz *transMarcaToBelly* procurada. Novamente cada marca fornece uma matriz *transMarcaToBelly* diferente e portanto o total de matrizes são 7 cada qual obtida por uma marca diferente.

3. ***transBellyToQuad*** - A matriz que relaciona a posição do *frame* do quadrotor, em relação ao *frame* da câmera vertical, é a matriz chamada de *transBellyToQuad*. Como a câmera vertical (*belly cam*) fica presa ao quadrotor, essa matriz será a mesma independente da marca identificada na pista.

4. ***transPistaToQuad*** - Finalmente a matriz de interesse que vai fornecer a pose do quadrotor em relação ao referencial da pista é a matriz de transformação *transPistaToQuad*. É obtida segundo uma transformação sequencial na qual saímos do referencial da pista e chegamos no referencial do quadrotor passando pela marca e pela câmera. Assim, primeiro multiplicamos *transPistaToMarca* por *transMarcaToBelly* obtendo *transPistaToBelly*. Em seguida multiplicamos *transPistaToBelly* por *transBellyToQuad* e chegamos na matriz *transPistaToQuad* procurada.

5. ***pose*** - A matriz *transPistaToQuad* é uma forma de representar a pose do quadrotor em relação a pista de pouso. A partir dela calculamos os seis parâmetros procurados $(x, y, z, \alpha, \beta, \gamma)$ conforme as equações da seção 2.3.3 do capítulo de fundamentação teórica.

A figura 5.9 mostra um exemplo onde relacionamos o *frame* da pista de pouso com o *frame* do VANT por intermédio de transformações de coordenadas sequenciais. Na figura 5.10 ilustramos a

matriz de transformação *transPistaToQuad* obtida pela multiplicação das matrizes de transformação *transPistaToMarca*, *transMarcaToBelly* e *transBellyToQuad*. Vale destacar que essas figuras exemplificam a situação em que a marca M5 é detectada. Na prática, esse procedimento se repete para todas as marcas detectadas em um dado momento. Assim, cada uma delas vai fornecer uma estimativa da pose do VANT em relação à pista de pouso.

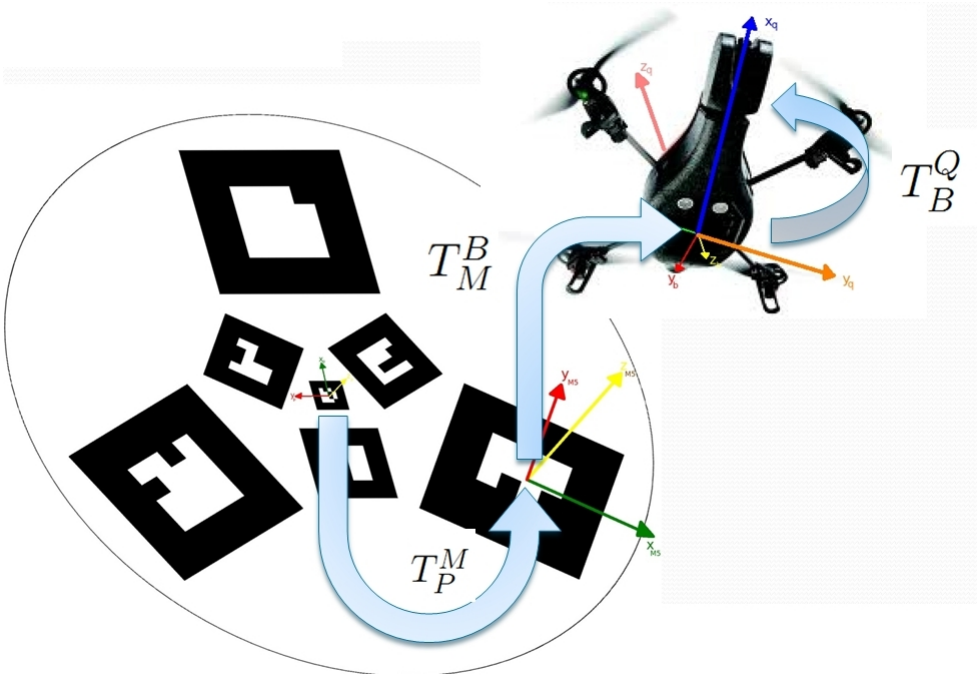


Figura 5.9: Matrizes de transformação *transPistaToMarca*, *transMarcaToBelly* e *transBellyToQuad*.

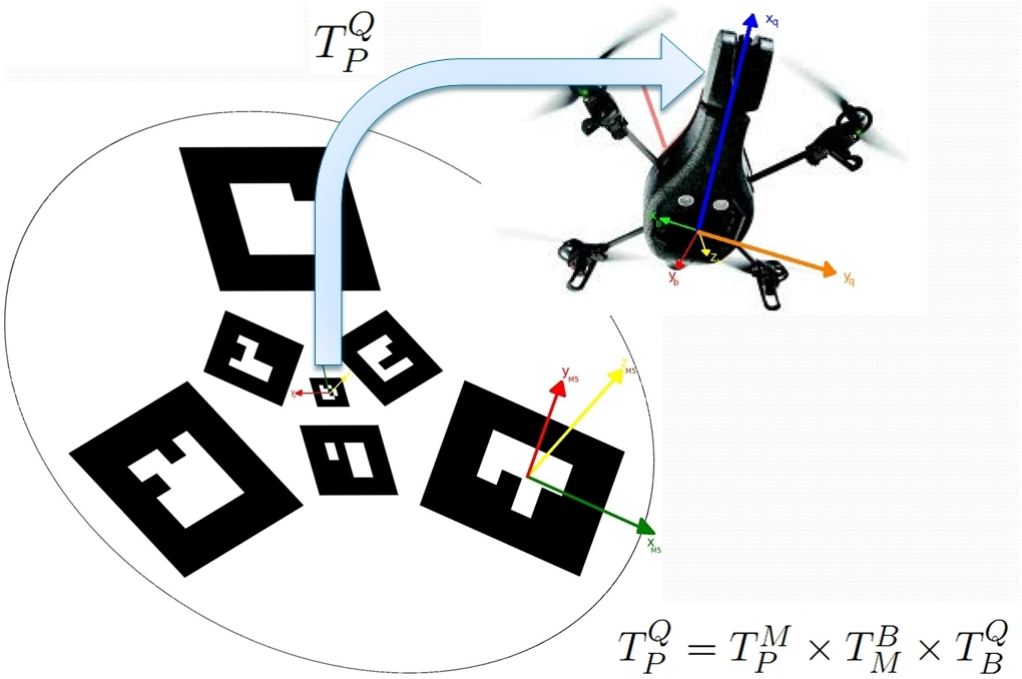


Figura 5.10: Matriz de transformação *transPistaToQuad*.

5.3 Fusão dos dados

A cada imagem capturada da câmera de dorso, os sete processos *modDetectaM* são lançados para identificar cada qual uma marca distinta da pista de pouso, podendo identificar ou não as respectivas marcas. Este processo pode ser influenciado devido a uma série de razões dentre as quais podemos citar:

- **Condições de luz** - Uma marca pode estar melhor iluminada que outra e assim ser identificada enquanto a outra não, mesmo que as duas estejam no campo de visão da câmera.
- **Oclusão parcial** - Uma marca pode estar escondida ou parcialmente escondida por algum objeto, ou mesmo, uma sombra pode prejudicar a detecção dessa marca.
- **Marca fora do campo de visão da câmera** - A câmera pode estar visualizando somente uma parte da pista onde pode-se ver completamente apenas algumas marcas.
- **Identificação errônea da marca** - É possível ainda que a detecção de uma das marcas do *ARToolKit* identifique erroneamente uma marca em relação com a outra.

Assim, conforme o VANT sobrevoa a pista de pouso, a cada imagem capturada podemos ter um conjunto de marcas identificadas, cujo total varia de 0 a 7 marcas identificadas. Sabe-se que cada marca identificada já fornece uma estimativa da pose do VANT em relação ao referencial fixo no centro da pista de pouso.

Dessa forma, supondo que em um dado momento M marcas do total possível tenham sido identificadas teremos um total de M eixos coordenadas de localização x, y, z e orientação α, β, γ. No caso ideal, era de se esperar que esses valores de pose entre duas marcas diferentes fossem exatamente os mesmos. Contudo, devido a aspectos físicos de construção da pista, calibração da câmera e mesmo as limitações do *ARToolKit* discutidos na seção 3.3.1 esses valores não serão exatamente iguais.

Portanto, se faz necessária a fusão de dados para descobrir uma forma de combinar os valores dos parâmetros fornecidos por cada uma das marcas de forma a chegar na melhor estimativa possível da pose procurada. Além disso, é necessário também comparar os dados entre si para detectar e descartar discrepâncias que podem ocorrer em função, por exemplo, da identificação errônea de marcas.

Assim, para alcançar uma melhoria significativa das estimativas, o procedimento de fusão dos dados obtidos foi realizado utilizando três metodologias diferentes: fusão multimarcas para todos os parâmetros da pose, fusão geométrica e fusão final específica para o parâmetro z da pose, já que para o VANT esse parâmetro equivale a altura de voo que é a informação mais importante nesse conjunto. As seções a seguir tratam cada uma desses processos de fusão.

5.3.1 Fusão multimarcas para todos os parâmetros da pose

A consideração que se faz nesse processo de fusão multimarcas é que das M marcas detectadas, o número de informações verdadeiras (isto é, que não são fruto de uma identificação errada de

marcas ou de algum provável ruído) são sempre maiores do que o número de informações erradas.

Sendo assim, do universo de informações recebidas, sabemos que dado uma tolerância, o número de informações corretas deve se repetir mais do que o número de informações erradas. Partindo desse pressuposto calculamos quais dessas informações se repetem mais e tomamos a média aritmética simples delas como a informação final.

A partir da hipótese acima, podemos definir que: Seja $\eta \in \Re$ uma estimativa para um dos parâmetros da pose. E também Γ o conjunto finito de estimativas η para um mesmo parâmetro da pose. Procuramos encontrar o valor η_f que fornece a melhor estimativa desse parâmetro dentre as diversas η estimativas de Γ.

Definimos $\delta \in \Re$ um valor de tolerância especificado.

Definimos o segmento ω (Equação 5.1) como um conjunto infinito de valores $\eta \in \Re$ de forma que:

$$\omega_i = [\eta_i - \delta, \eta_i + \delta] \tag{5.1}$$

Definimos o *operador sobreposição* denotado pelo símbolo $\leftrightharpoons$. A sobreposição entre dois segmentos ω_a e ω_b quaisquer é definida por: ω_a sobrepõe ω_b, se e somente se, $\exists$ algum valor η tal que $\eta \in \omega_a$ e $\eta \in \omega_b$. Podendo ser apresentada na forma matemática na equação 5.2.

$$\omega_a \leftrightharpoons \omega_b \leftrightarrow \exists \eta / \eta \in \omega_a, \eta \in \omega_b. \tag{5.2}$$

Definimos, conjunto Υ o conjunto formado por segmentos ω e também como *tamanho* de um conjunto como o número de elementos desse conjunto.

Então, pode-se agora apresentar a metodologia da fusão multimarcas:

1. Temos o conjunto Γ de valores η_i em que i é o índice da marca que forneceu o dado η_i, ou seja, i varia de 0 a 6.
2. A partir de um tolerância δ, calculamos para cada η_i de Γ o segmento ω_i correspondente formando assim o conjunto Υ formado pelos ω_i calculados.
3. Selecionamos do conjunto Υ quais segmentos ω_i que se sobrepõe entre si formando conjuntos Ψ_n de elementos η que se sobrepõe.
4. Dos n conjuntos Ψ_n formamos na etapa anterior, escolhemos aquele que possuir o maior número de elementos η.
5. Finalmente, a estimativa η_f procurada será dada pela média aritmética dos elementos do conjunto Ψ escolhido na etapa anterior.

Por exemplo, se num dado momento somente as marcas 0,1,3 e 4 forem detectadas, então o conjunto Γ de parâmetros η a partir do qual queremos descobrir η_f será:

$$\Gamma = \{\eta_0, \eta_1, \eta_3, \eta_4\} \tag{5.3}$$

Cada η_i gera um ω_i correspondente que formam o conjunto Υ:

$$\Upsilon = \{\omega_0, \omega_1, \omega_3, \omega_4\} \tag{5.4}$$

Suponha que os segmentos ω_0, ω_1 e ω_3 se sobreponham. E que ω_3 e ω_4 também se sobreponham. Então teremos dois subconjuntos Ψ:

$$\Psi_1 = \{\eta_0, \eta_1, \eta_3\} \tag{5.5}$$

e

$$\Psi_2 = \{\eta_3, \eta_4\}. \tag{5.6}$$

Então como o conjunto Ψ_1 possui o maior número de elementos η (3) a estimativa η_f será a média aritmética dos elementos de Ψ_1.

Essa metodologia pode ser vista como uma média que descarta elementos muito distantes e computado na média somente elementos próximos. No exemplo dado a estimativa η_4 foi descartada. A medida do quão distante ou próximo um valor é do outro é obtido usando a tolerância δ. Se δ for muito grande todas as estimativas vão se sobrepor e assim a estimativa final será a média aritmética simples delas. Se δ for muito pequeno, nenhuma estimativa vai se sobrepor e assim teremos tantos conjuntos Ψ de tamanho unitário, quantas estimativas η fizerem parte do conjunto Γ inicial. Nesses casos, e também em casos em que existirem mais de um conjunto Ψ com o maior tamanho, escolhemos o primeiro conjunto computado e descartamos os demais.

Em suma, aplicamos essa metodologia para cada conjunto de parâmetros e encontramos a pose de fusão multimarcas $(x_m, y_m, z_m, \alpha_m, \beta_m, \gamma_m)$. Vale ressaltar que os valores δ utilizados como tolerância para fazer a fusão multimarcas de cada parâmetro foram ajustados por tentativa e erro empiricamente chegando aos valores de 10 mm para os parâmetros x, y, z e $2,5°$ para os parâmetros α, β e γ.

5.3.2 Fusão geométrica para o parâmetro z

A fusão geométrica desenvolvida é uma forma de determinar a coordenada z do VANT em relação à pista de pouso sem recorrer as informações de pose fornecidas pelo *ARToolKit*, mas avaliando somente as informações relativas à geometria da posição do VANT e a pista. Além da pose, o *ARToolKit* fornece as coordenadas do centro de cada marca no referencial da imagem capturada pela câmera. Como sabemos a distância entre dois pontos na pista real, se soubermos a distância entre dois pontos na imagem, podemos usar essas duas informações juntamente com os os parâmetros intrínsecos de calibração da câmera para calcular de forma independente a distância da câmera até a pista de pouso. Essa distância é a altura de voo que é equivalente a coordenada z do VANT. Contudo vale salientar que essa metodologia de fusão geométrica só é válida quando o plano (x, y) do VANT é paralelo ao plano (x, y) da pista de pouso, ou seja, só pode ser usada para estimar a altura quando os ângulos de arfagem e rolagem do VANT estão próximos de zero.

Para entender a utilização da fusão geométrica suponha que em um dado instante de tempo o sistema de captura de imagens detecte as seguintes marcas da pista de pouso: M0, M1 e M2, e

que a distância entre os os centros de todas as marcas entre si são conhecidas. Nesse exemplo em particular, possuímos as seguintes informações apresentadas na tabela 5.1.

Marca	**Marca**	**Dist. real**
M0	M1	121.50 mm
M0	M2	47.64 mm
M1	M2	106.04 mm

Tabela 5.1: Exemplo das distâncias reais entre os centros das marcas detectadas da pista de pouso.

Como o *ARToolKit* fornece as coordenadas do centro de cada marca detectada na imagem, podemos estimar a altura através de uma regra de três simples descrita na Equação 5.7.

$$H_{real} = \frac{D_{real}}{D_{img}}.Foco_{img} \tag{5.7}$$

onde, H_{real} é a altura real a estimar (mm), D_{real} é a distância entre marcas da tabela 5.1 (mm), D_{img} é a distância entre marcas na imagem capturada $(pixels)$ e, finalmente, o $Foco_{img}$ é a distância focal da câmera $(pixels)$.

Neste caso, existem três distâncias medidas e estimamos três alturas que na prática deveriam ser as mesmas. No pior caso teremos somente duas marcas detectadas, que vai fornecer somente uma única distância entre marcas e portanto somente uma única altura. No melhor caso teremos as sete marcas detectadas, que vai fornecer a combinação das sete marcas tomadas duas a duas que é igual a 21(vinte e uma) estimações de alturas cuja média aritmética simples vai fornecer a altura final da fusão geométrica. Quando há uma ou nenhuma marca detectada a estratégia utilizada é fornecer o valor da última altura calculada e no caso em que ainda não foi realizada nenhuma estimação, esse valor é dado como zero e não é utilizado na fusão final.

5.3.3 Fusão final para o parâmetro z

A fusão final é feita a partir de um procedimento simples: realiza uma tentativa de escolher entre o parâmetro z_m calculado pela fusão multimarcas e a altura H_{real} calculada pela fusão geométrica. Sabemos que se o número de marcas detectadas for igual ou superior a duas marcas, então tomamos o parâmetro z_f da pose como a média aritmética simples do parâmetro obtida na fusão multimarcas e da altura calculada pela fusão geométrica.

Vale lembrar, no entanto, que a metodologia da fusão geométrica e final só servem para melhorar as estimações da altura do VANT quando esse encontra-se paralelo à pista de pouso.

Capítulo 6

Resultados Experimentais

Neste capítulo apresentamos os ensaios experimentais realizados com o objetivo de testar se os valores das medições feitas pelo sistema de visão computacional desenvolvido correspondem à realidade. Todos os experimentos foram feitos utilizando a câmera vertical do *AR.Drone* onde as medidas dos parâmetros obtidas pelo sensor de visão são comparadas às respectivas medidas dos sensores embarcados no VANT, que são fornecidas através da estrutura *NAVDATA*, como anteriormente descrito no capítulo 4.

É importante lembrar que foram realizados diferentes processos de fusão dos dados para o parâmetro z. Assim, os resultados de visão para esse parâmetro vem discriminados em separado nos gráficos com as legendas *Fusão Multimarcas*, *Fusão Geométrica* e *Fusão Final* para os resultados das medições por visão e *Altímetro* para as medições realizadas pelo sensor de ultrassom do *AR.Drone*, respectivamente. Em contrapartida, como os demais parâmetros da pose utilizam-se somente da fusão multimarcas, usamos a legenda *Visão* para os resultados de visão e a legenda *NAVDATA* para as medições dos sensores embarcados no VANT.

A seção a seguir classifica os ensaios realizados e em seguida, nas demais seções, apresentamos para cada parâmetro da pose $(x, y, z, \alpha, \beta, \gamma)$ o procedimento experimental e os resultados encontrados.

6.1 Classificação dos ensaios

Os ensaios podem ser classificados de três formas: A primeira forma é quanto à periodicidade em que os registros são gravados em arquivo. A segunda forma é quanto à utilização ou não de filtragem no valor do dado gravado. E a terceira forma é quanto ao estado do *AR.Drone* voando ou não.

Em relação à periodicidade, as coletas de cada registro dos ensaios podiam ocorrer de forma automática ou manual. No formato automático, a cada intervalo de tempo um novo registro era gravado e por isso chamamos esses ensaios de **ensaios contínuos**. No formato manual, os registros eram gravados segundo um comando dado pelo usuário não importando o tempo. Como nos ensaios manuais o VANT sempre se encontrava parado, chamamos esses ensaios de **ensaios estáticos**.

Já em relação à utilização ou não de filtragem, cada registro podia ser do dado filtrado ou da leitura instantânea do sensor naquele momento. O filtro implementado consistiu num filtro de média em que calculava-se a média dos vinte e cinco últimos dados.

Uma outra observação a ser feita é que não é necessária dar a partida nos motores do *AR.Drone* para que seja possível visualizar as imagens da câmera vertical, e também para que seus sensores façam as medições dos valores de cada parâmetro. Assim, os ensaios experimentais podiam ser feitos com o VANT parado posicionado acima da pista de pouso segurando-o com a mão ou içando-o através de cabos. Na prática a maioria dos ensaios foram feitos a partir do içamento do quadrotor com cabos, pois desta forma é mais fácil controlar as variáveis de interesse. Contudo, o sistema proposto tem como objetivo o seu uso durante o voo, logo ensaios com a aeronave voando também foram realizados. Nesses ensaios, foi registrado continuamente a cada segundo todos os dados fornecidos pela estrutura de dados *NAVDATA* do *AR.Drone* e todos os dados de visão para posterior análise. Desta feita, nesta situação do VANT voando foi denominada de **ensaio durante voo**.

6.2 Parâmetros x e y

Ao contrário dos parâmetros z, α, β e γ, não há sensores no *AR.Drone* capazes de fornecer dados que possam ser comparados as distâncias x e y do *frame* do VANT em relação ao centro da pista de pouso. A única solução seria fazer medições físicas diretas nessas direções entre o eixo óptico da câmera vertical do *AR.Drone* e um eixo imaginário perpendicular ao plano da pista de pouso que passa pela sua origem. Embora consideramos realizar tais ensaios, eles se mostraram impraticáveis em razão da dificuldade inerente em se obter estas medidas. Dessa feita, recorremos a ensaios qualitativos que podem ser encontrados nos vídeos que acompanham o CD (anexo IV).

Nesses vídeos, posicionamos o *AR.Drone* em cada um dos quatro quadrantes delimitados pelos eixos x e y no plano da pista de pouso e mostramos que pelo menos os sinais das medições são corretamente aferidos pelo sistema, ou seja, é possível saber se o *AR.Drone* se encontra à direita ou à esquerda de cada um dos eixos.

6.3 Parâmetro z (Altura)

O parâmetro z corresponde à altura do VANT em relação à pista de pouso. Podemos compará-lo com a altitude medida pelo altímetro ultrassônico do *AR.Drone* desde que o plano do VANT esteja paralelo ao plano da pista de pouso. Além disso, não é difícil medir essas distâncias diretamente com o auxílio de uma régua com precisão relativa. Com certeza, o parâmetro z foi o mais experimentado nesse trabalho e o que atingiu os melhores resultados.

Procedimentos experimentais

Realizamos ensaios estáticos (coleta manual, dados filtrados, VANT parado), ensaios contínuos (coleta automática, dados filtrados, VANT parado) e ensaios com o VANT voando (coleta auto-

mática e dados instantâneos). Os procedimentos do ensaio estático seguem os seguintes passos:

- **Procedimento ensaios estáticos**

  1. A pista de pouso é colocada paralela ao chão.
  2. Içar o VANT de forma que seu plano (x, y) fique paralelo ao plano (x, y) da pista.
  3. Incrementar a altura entre o VANT e o chão.
  4. Estabilizar o VANT de modo que seu plano seja paralelo ao plano da pista. Ou seja, fazer $\beta = \gamma = 0$.
  5. Medir a altura na régua.
  6. Medir a altura utilizando o altímetro ultrassônico do *Parrot AR.Drone*.
  7. Medir o parâmetro z dado pelo sensor de visão.
  8. Registrar também outras informações como o número de marcas detectadas naquela medição.
  9. Voltar ao passo 3.

  A primeira medida é feita com o quadrotor no chão. A segunda medida é feita numa altura em que o sistema de visão identifica pelo menos 1 marca (provavelmente a marca central). Os passos são repetidos até que o sistema de visão não identifique mais nenhuma marca. O Δz utilizado, isto é, o intervalo entre uma altura e outra é pequeno de forma que possamos fazer de 10 a 30 medições.

  Os procedimentos realizados para o ensaio contínuo utilizou os seguintes passos:

- **Procedimento ensaios contínuos**

  1. A pista de pouso é colocada paralela ao chão.
  2. O VANT é içado sobre a pista de pouso inicialmente com altura nula.
  3. A coleta de dados é iniciada e o parâmetro z é medido a cada segundo tanto pelo sistema de visão quanto pelos sensores do *AR.Drone*.
  4. A altura é incrementada continuamente até alcançar a altura máxima.

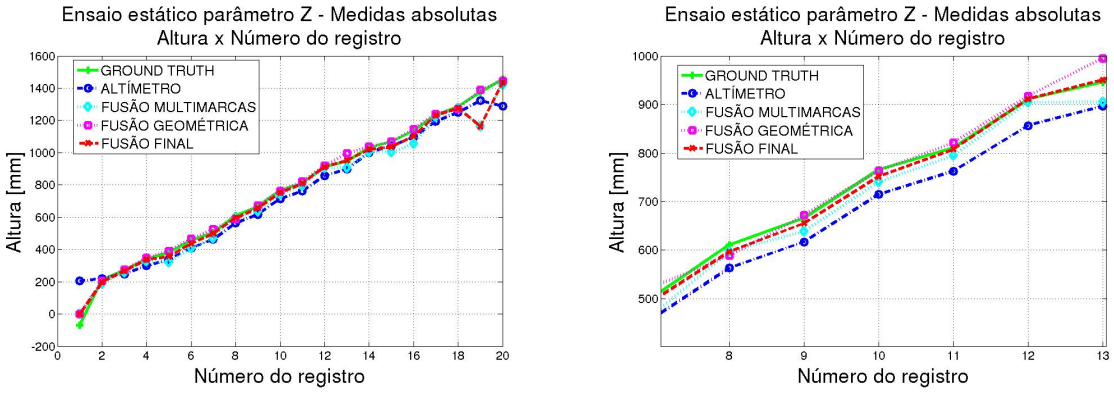
  O intervalo de tempo entre cada medição é de aproximadamente 1 segundo. A altura máxima que conseguimos medir foi de até 2 metros e meio.

- **Procedimento ensaios durante voo**

  O VANT é colocado para voar e tentamos mantê-lo voando sobre a pista de voo enquanto o programa grava a cada segundo as alturas lidas pelo altímetro ultrassônico e pelo sistema de visão. Vale destacar que os dados nesses ensaios não são filtrados e assim aparecerão muito mais ruídos nos gráficos. Além disso, pela natureza do VANT não é fácil manter o VANT voando sobre a pista de pouso sem que tenhamos que variar os ângulos de rolagem e arfagem do mesmo já que para que ele se desloque de um lado para o outro, precisamos variar esses ângulos. E ao variar esses ângulos, as medidas dadas pelos sensores de ultrassom do mesmo ficam comprometidas.

6.3.1 Resultados para ensaios estáticos

Nesses ensaios além dos dados de visão e aqueles fornecidos pelo *NAVDATA*, também registramos a altura com uma régua. Chamamos os dados fornecidos pela régua de *ground-truth* já que ele é a referencia para saber quais dos sensores se aproxima mais da verdade. Dessa forma, plotamos os gráficos da figura 6.1(a) que mostram, respectivamente, o *ground-truth*, a medição realizada pelo altímetro, a medição da fusão multimarcas, da fusão geométrica e da fusão final.



(a) Completo

(b) Detalhe

Figura 6.1: Ensaio estático do parâmetro z - Medidas absolutas

Na figura 6.1(b) mostramos um detalhe onde ampliamos a região central do gráfico para analisar melhor as curvas. A primeira observação é que a curva que mais se afasta das medidas do *ground-truth* são aquelas realizadas pelo altímetro ultrassônico enquanto que aquelas realizadas usando os dados obtidos por visão são mais próximas. Isso pode ser visualizado melhor ao plotarmos os erros absolutos em relação ao *ground-truth* calculado para cada registro do gráfico. Isso foi feito na figura 6.2.

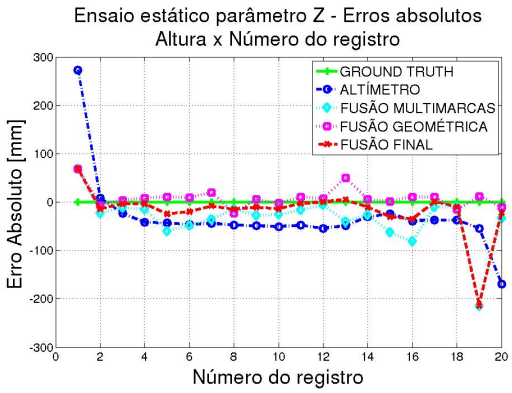
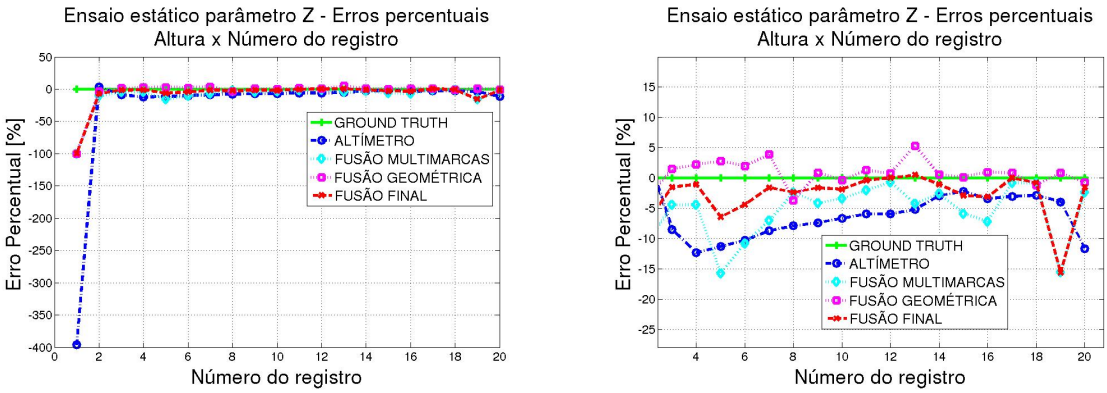


Figura 6.2: Ensaio estático do parâmetro z - Erros absolutos.

Analisando a figura 6.2 confirma-se que na maioria das medições as estimativas de visão possuem erro menor que as medidas do altímetro ultrassônico. Podemos visualizar isso também se plotamos os erros percentuais relativos. Isso é feito nas figuras 6.3(a) e 6.3(b) onde a última apresenta uma ampliação da primeira numa região em que podemos ver melhor as diferenças entre as curvas.



(a) Completo (b) Detalhe

Figura 6.3: Ensaio estático do parâmetro z - Erros percentuais.

Contudo, uma variável importante para se analisar e entender melhor os gráficos é saber em cada medição qual ou quais marcas da pista de pouso estavam sendo detectadas pelo algoritmo de visão. Essa informação é mostrada no gráfico da figura 6.4 que relaciona para cada registro quais marcas estavam sendo detectadas naquele momento.

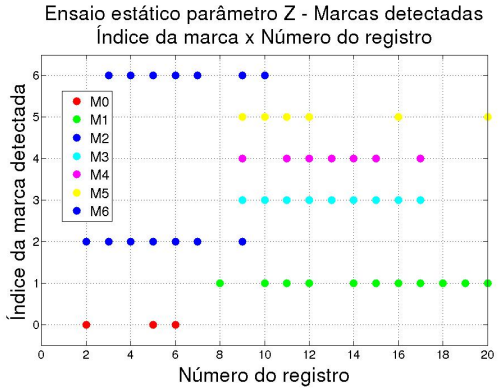


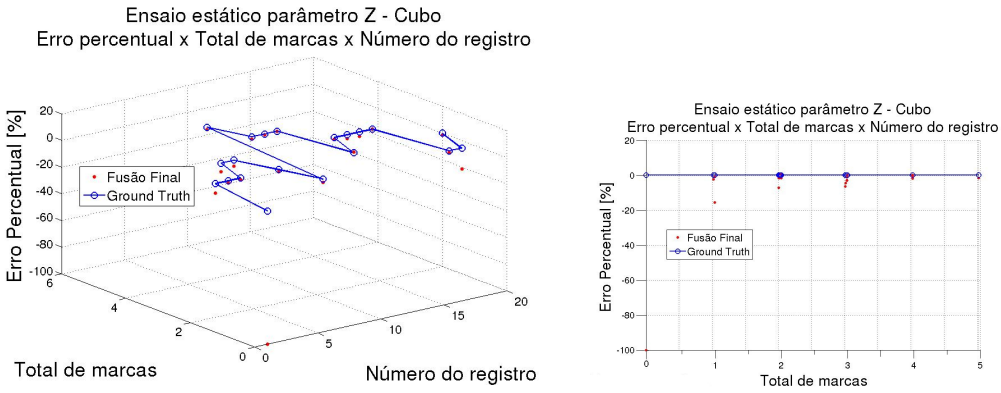
Figura 6.4: Ensaio estático do parâmetro z - Marcas detectadas.

Finalmente é interessante também plotar um gráfico tridimensional que relaciona o número do registro, o total de marcas detectadas naquele registro e o erro percentual da fusão final. Isso foi feito na figura 6.5(a).

Se olhamos esse cubo de forma a visualizar o total de marcas em relação ao erro percentual em todos os registros realizados, como ilustra a figura 6.5(b), uma conclusão interessante é que o erro percentual diminui à medida em que se aumenta o número de marcas detectadas. Assim quanto mais marcas são detectadas em determinado momento, melhor é a estimativa do valor.

6.3.2 Resultados para ensaios contínuos

Ensaios com coleta contínua são muito mais fáceis de serem realizados já que não temos que ficar lendo na régua o valor da medida a cada registro. Tudo o que que temos que fazer é pedir ao programa que registre a cada intervalo de tempo um novo registro com os dados lidos pelos



(a) Número do Registro x Total de Marcas x Erro Percentual

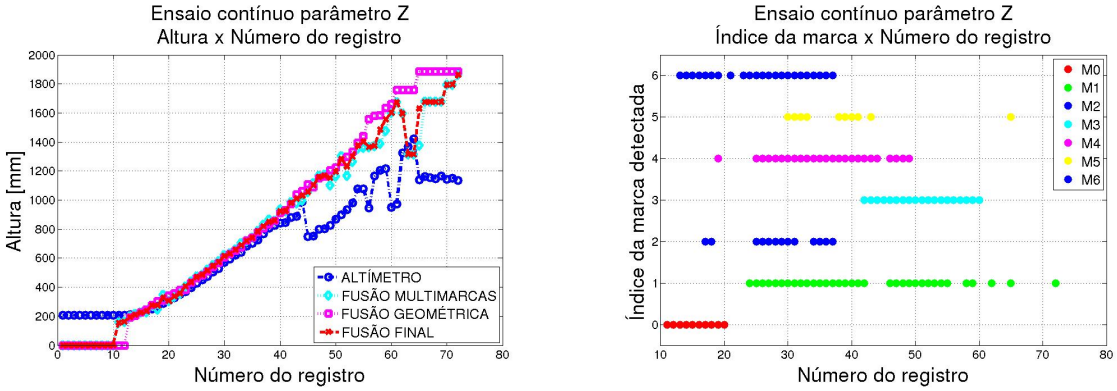
(b) Total de Marcas x Erro Percentual

Figura 6.5: Ensaio estático do parâmetro z - Cubo.

sensores. A desvantagem é que não temos um *ground-truth* confiável para tomar como verdade e as leituras devem ser analisadas umas com as outras. Dessa forma, decidiu-se ser mais breve nessa seção selecionando alguns gráficos de ensaios realizados comentando cada um.

6.3.2.1 Ensaio 1

- Nesse ensaio, registramos a cada segundo os dados do altímetro, da fusão multimarcas, da fusão geométrica e da fusão final quando o VANT é içado do chão até uma altura limite gerando os gráficos mostrados nas figuras 6.6(a) e 6.6(b).



(a) Alturas

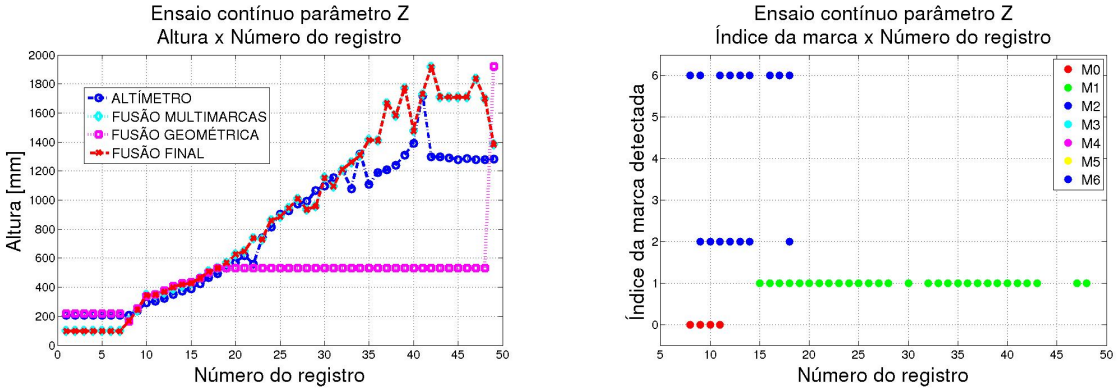
(b) Marcas detectadas

Figura 6.6: Ensaio 1 contínuo do parâmetro z.

Esse ensaio foi feito elevando-se a altura continuamente e colhendo os dados de interesse em intervalos de tempo regulares de 1 segundo. A partir da observação do gráfico da figura 6.6(a), verifica-se que a uma dada altura (aproximadamente 1 metro) o altímetro ultrassônico começa a errar acusando alturas menores do que a real enquanto que os dados de altura provenientes da fusão final continuam subindo linearmente.

6.3.2.2 Ensaio 2

- Esse ensaio foi conduzido de forma semelhante ao ensaio anterior. Contudo, a pista de pouso estava mal iluminada o que fez com que o sistema de visão passasse a identificar somente uma marca nos últimos registros. A consequência disto é que a fusão geométrica ,que depende de pelo menos 2 marcas identificadas para funcionar, fica "parada" no último valor válido. No entanto, isso não afeta a fusão final que verifica que isso está ocorrendo e descarta esse os dados da fusão geométrica até que um número maior de marcas seja identificada. Esse comportamento pode ser visto nas figuras 6.7(a) e 6.7(b).



(a) Alturas

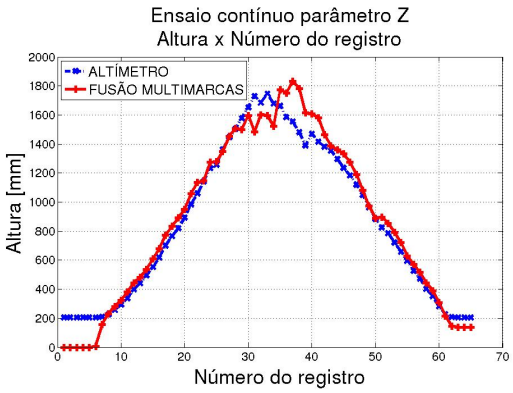
(b) Marcas detectadas

Figura 6.7: Ensaio 2 contínuo do parâmetro z

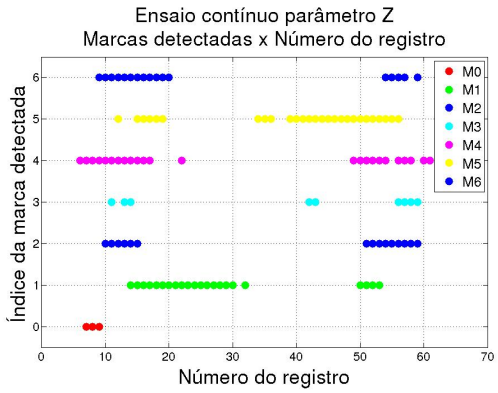
6.3.2.3 Ensaio 3

- Finalmente, é importante lembrar que embora os dados de altura fornecidos pela fusão final apresentaram resultados muito bons. Eles só são confiáveis caso o plano de voo do VANT esteja paralelo ao plano da pista de pouso. Em casos em que não podemos afirmar que o VANT esteja paralelo ao plano da pista de pouso, só podemos confiar nas estimativas do parâmetro z fornecidas pela fusão multimarcas onde a orientação do VANT não afeta essa estimativa. Sabendo disso, fizemos um ensaio em que simulamos a decolagem e aterrissagem do VANT e coletamos somente os dados de altura do altímetro e da fusão multimarcas. As figuras 6.8(a) e 6.8(b) mostram esse ensaio contínuo no qual o VANT foi içado a partir do chão (altura nula) subindo gradualmente até uma dada altura máxima e depois foi abaixado também gradualmente até novamente tocar o chão.

As observações que podem ser feitas a partir do gráfico da figura 6.8(a) auxiliado pelas informações da figura 6.8(b) são que as informações de altura da fusão multimarcas são por si só confiáveis o bastante já que conseguem acompanhar o sensor altímetro do *Parrot AR.Drone*. Mais tarde mostraremos gráficos obtidos em experimentos com voos reais.



(a) Altímetro e Fusão Multimarcas

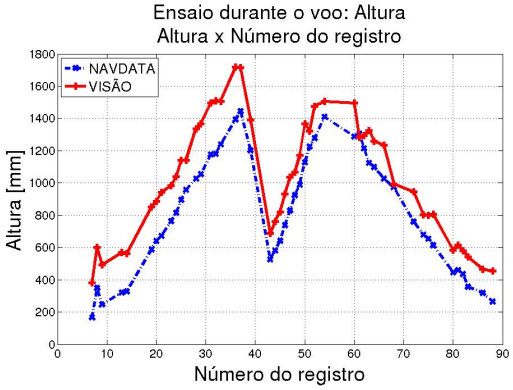


(b) Marcas detectadas

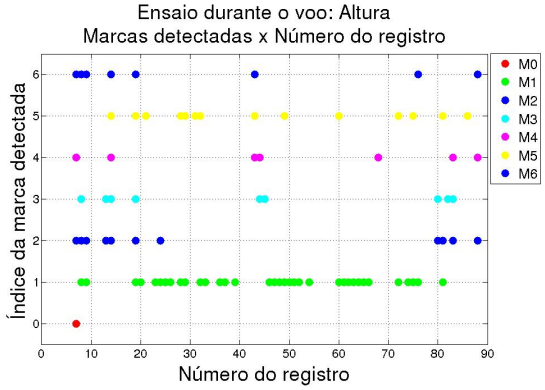
Figura 6.8: Ensaio 3 contínuo do parâmetro z

6.3.3 Resultados para ensaios durante voo

A figura 6.9(a) mostra o gráfico das alturas durante o ensaio experimental na qual o VANT voava. A figura 6.9(b) mostra as marcas detectadas nesse voo. Verifique que os gráficos apresentam comportamento semelhante aos realizados estaticamente já que os dados do altímetro ultrassônico (dados esses do *NAVDATA*) erram para menos enquanto os dados de visão ficavam mais próximos do que deve ser o *ground-truth*.



(a) Altura



(b) Marcas detectadas

Figura 6.9: Ensaio durante voo - altura

6.4 Parâmetro α (Guinada ou *YAW*)

O parâmetro α é o responsável pela guinada do VANT. O sensor do *Parrot AR.Drone* é um girômetro magnético, ou seja, ele fornece a guinada em relação ao sul magnético (norte geográfico). Dessa forma, antes de medir deveríamos tomar o cuidado de alinhar o norte (eixo x) da pista de pouso com o norte geográfico. Contudo, enfrentamos um problema ao tentar fazer essas medições porque o sensor do *AR.Drone* apresentou um comportamento no mínimo estranho durante os experimentos. Para mostrar esse comportamento atípico, plotamos na figura 6.10 os gráficos que

mostram o resultado de um ensaio experimental onde deixamos o *AR.Drone* parado no chão e coletamos a cada segundo os valores dos sensores de orientação *YAW*, *PITCH* e *ROLL*.

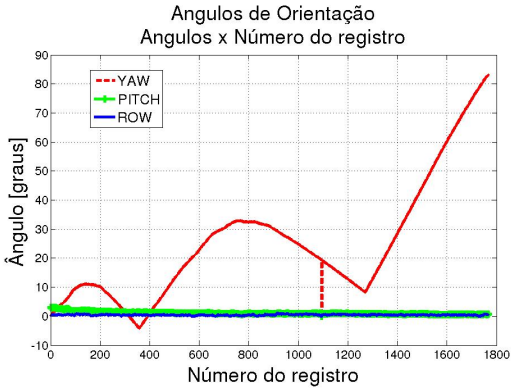


Figura 6.10: Ensaio contínuo dos ângulos de orientação com o *AR.Drone* em repouso.

Era esperado que todos os valores se mantivessem constantes flutuando apenas numa pequena faixa correspondente ao erro do sensor. Isso ocorre com os dados de *PITCH* e *ROLL*. Contudo, as leituras do sensor responsável pelo ângulo de guinada não se mantiveram constantes. Assim, como não sabemos a origem desse erro, não haveria como comparar esses dados com os dados de guinada do sistema de visão.

Dessa forma, a solução foi a seguinte: não importa qual é o referencial nulo do sensor de visão e do sensor do VANT, o que importa é que ao girarmos o VANT, as variações angulares lidas por esses dois sensores deveriam ser as mesmas.

Procedimentos experimentais

Somente fizemos ensaios desse parâmetro com o VANT voando. Logo o procedimento é o mesmo adotado em todos os ensaios desse tipo, ou seja, os dados são gravados a cada segundo.

6.4.1 Resultados para ensaios durante voo

A figura 6.11 mostra o gráfico do ângulos de guinada (*YAW*) do VANT obtida por visão e pelo *NAVDATA*. Veja que embora os dois gráficos comecem em medidas angulares distintas (o que é esperado já que os dois possuem referenciais inicias distintos) ambas curvas seguem o mesmo comportamento o que indica variações angulares iguais. Convém lembrar que as marcas detectadas nesse ensaios são as mesmas da figura 6.9(b) anteriormente mostrada no gráfico da altura (o experimento que gerou esses gráficos foi o mesmo).

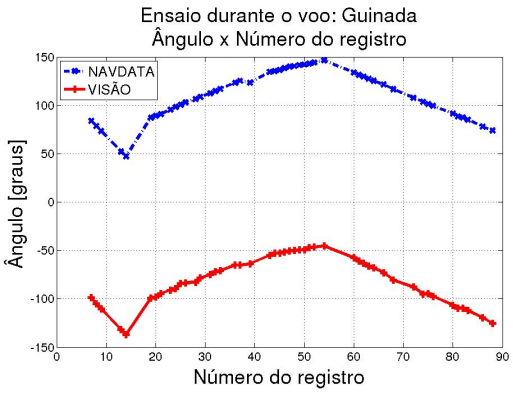


Figura 6.11: Ensaio durante o voo - guinada

6.5 Parâmetro β (Arfagem ou *PITCH*)

Procedimentos experimentais

O parâmetro de arfagem (β) foi ensaiado de forma estática segundo o seguinte procedimento experimental:

1. A aeronave é manualmente disposta em cima da pista de pouso e lentamente inclinada de forma a fazer variar o ângulo β.
2. Uma pessoa pressiona um botão no controle responsável por registrar naquele momento os dados.

Esse procedimento é repetido para gravar registros em diversos ângulos. Além disso, também foram feitos ensaios com o VANT voando.

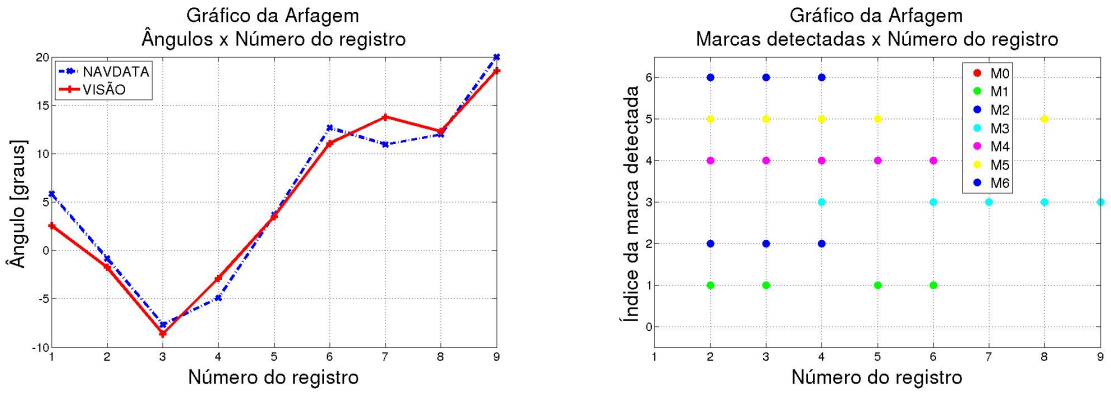
6.5.1 Resultados para ensaios estáticos

Nesse manuscrito selecionamos um dos ensaios realizados para comparar o ângulo de arfagem fornecido pelo *NAVDATA* e aquele fornecido pelo sistema de visão desenvolvido. Esse ensaio é manual, ou seja, cada registro é feito com o VANT parado. Além disso, aplicamos um filtro de média nos dados, isto é, cada registro dos ângulos dos gráficos é na realidade a média dos últimos 25 registros coletados. Esse filtro foi necessário já que os ângulos do sistema de visão apresentaram ruídos em suas saídas como mostraremos nos gráficos dos ensaios durante o voo.

A figura 6.12(a) mostra o ensaio e a figura 6.12(b) as marcas detectadas em cada registro. Verifique que os ângulos do sistema de visão se aproximam dos ângulos fornecidos pelo *NAVDATA*.

6.5.2 Resultados para ensaios durante voo

A figura 6.13 mostra os gráficos do ângulo de arfagem (*PITCH*) do VANT obtida por visão e pelo *NAVDATA*. Veja que os valores ficaram muito próximos conforme era esperado. No entanto,



(a) Ângulos

(b) Marcas detectadas

Figura 6.12: Ensaio estático - arfagem

ocorreram picos no gráfico que são ruídos que devem ser tratados por algum filtro. Esses picos não ocorrem nos ensaios estáticos devido ao filtro de média utilizado. As marcas detectadas são as mesmas que na figura 6.9(b).

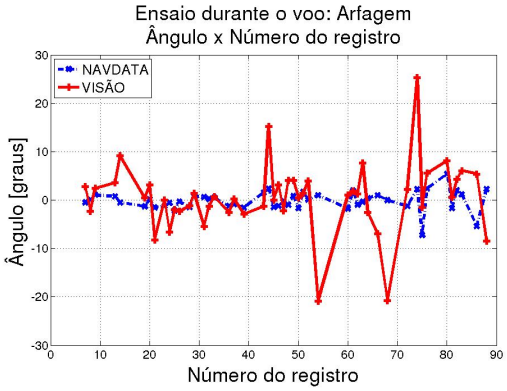


Figura 6.13: Ensaio durante o voo - arfagem

6.6 Parâmetro γ (Rolagem ou *ROLL*)

Procedimentos experimentais

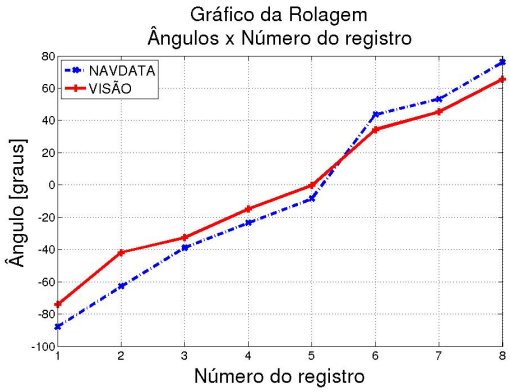
O parâmetro de rolagem (γ) foi ensaiado de maneira semelhante ao parâmetro β realizando-se ensaios estáticos.

1. A aeronave é manualmente disposta em cima da pista de pouso e lentamente inclinada de forma a fazer variar o ângulo γ.
2. Uma pessoa pressiona o botão no controle responsável por registrar naquele momento os dados.

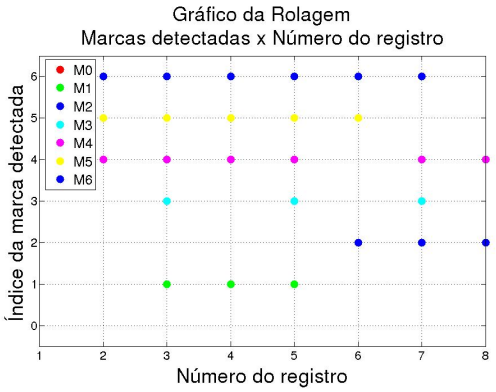
Novamente, esses passos são repetidos para vários ângulos de inclinação. Também realizamos ensaios durante o voo.

6.6.1 Resultados para ensaios estáticos

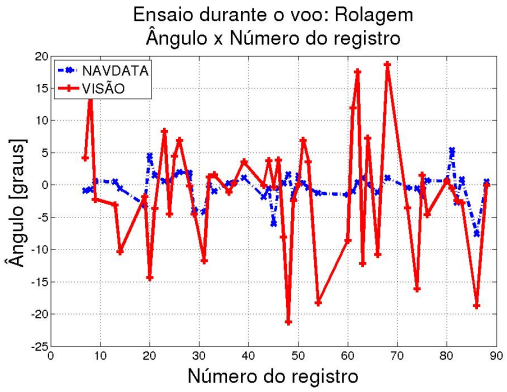
Novamente, apresentamos um ensaio realizado para comparar o ângulo de rolagem (ou rolagem) fornecido pelo *NAVDATA* e aquele fornecido pelo sistema de visão. As figuras 6.14(a) e 6.14(b) mostram o ensaio e as marcas detectadas respectivamente. Novamente, as medidas dos dois sensores são próximas.



(a) Ângulos



(b) Marcas detectadas



(c) Ensaio durante o voo

Figura 6.14: Ensaio estático e durante o voo - rolagem

6.6.2 Resultados para ensaios durante voo

A figura 6.14(c) mostra o gráfico do ângulos de rolagem (*ROLL*) do VANT obtida por visão e pelo *NAVDATA*. As marcas detectadas são as mesmas daquelas mostradas na figura 6.9(b). Veja que os valores também ficaram próximos, mas novamente ocorrem picos que devem ser tratados por algum filtro.

Os ensaios com o VANT parado foram obtidos filtrando os dados por um filtro de média. Os ensaios durante o voo do VANT foram obtidos sem filtrar os dados. A comparação entre os resultados mostra que as medidas instantâneas dos parâmetro z e α são bem estáveis e dispensam, portanto, a necessidade de usar um filtro. Os parâmetros β e γ, no entanto, se mostraram bem instáveis nos ensaios em que esse filtro não foi utilizado. Dessa feita, recomenda-se utilizar um filtro (sugerimos um filtro simples de média ou mediana) para melhorar a confiança nessas medições.

Capítulo 7

Conclusões

Neste trabalho foi proposta uma metodologia para a realização de estimação da pose a partir de sistemas baseados em visão monocular, combinando recursos da biblioteca *ARToolKit* com multimarcas para auxiliar no processo de pouso de VANTs em geral.

A principal vantagem do sistema apresentado é seu baixo custo já que o mesmo consegue estimar todos os parâmetros da pose usando somente uma câmera. A principal desvantagem é a necessidade de haver uma pista de pouso visível para identificar o referencial no solo em que se deseja pousar.

Com certeza o desempenho do sistema de visão ainda é fortemente dependente dos parâmetros de calibração da câmera, pelo menos utilizando os parâmetros intrínsecos, tanto quanto da estratégia de fusão dos dados que foi utilizada. Logo, como uma primeira sugestão para trabalhos futuros recomenda-se testar outras estratégias de fusão dos dados.

Reconhecemos também que embora os ensaios experimentais realizados mostram que as medições de visão se equipararam ou até mesmo sobrepujaram as medições dos sensores do *AR.Drone*, não foi possível realizar estudos estatísticos para confirmar isso com uma abordagem estatística formal. Esse estudo estatístico não foi realizado porque o mesmo dependeria da realização de um número bem maior de ensaios experimentais, impossibilitado devido à falta de tempo.

Uma segunda sugestão é utilizar uma metodologia de içamento com maior controle e precisão. Um exemplo seria a utilização do manipulador *shunck* do LARA para fazer esses ensaios de posicionamento do VANT com precisão milimétrica. Enquanto era possível realizar manualmente em torno de cinco ensaios por dia, esse manipulador poderia ser programado para realizar 50 ensaios diários coletando os dados de forma automática sem a necessidade de uma pessoa para realizar as medidas físicas a cada momento.

Finalmente, uma última sugestão para trabalhos futuros é utilizar esse sistema para fazer o controle de posição do VANT de forma que seja possível realizar sua decolagem e aterrissagem de forma automática usando somente as informações de pose provenientes do sistema de visão.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ABDESSAMEUD, A.; TAYEBI, A. Global trajectory tracking control of VTOL-UAVs without linear velocity measurements. *Automatica*, Elsevier, v. 46, n. 6, p. 1053–1059, 2010.

[2] HULL, L. Drone makes first UK 'arrest' as police catch car thief hiding under bushes. *MailOnline*, Fev. 2010. Disponível em: <http://www.dailymail.co.uk/news/article-1250177/Police-make-arrest-using-unmanned-drone.html>.

[3] NETO-CAP, A.; ALMEIDA, P. D.; PM, C. *A análise do emprego do veículo aéreo não tripulado (VANT) nas ações e operações PM*. Dissertação (Trabalho de conclusão de curso) — Universidade do Estado da Bahia, 2009.

[4] MERINO, L. et al. An unmanned aircraft system for automatic forest fire monitoring and measurement. *Journal of Intelligent & Robotic Systems*, Springer, p. 1–16, 2011.

[5] CAO, X. et al. Research of the automatic insertion system of a searching and rescuing robot. In: *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*. [S.l.: s.n.], 2010. p. 1121 –1123.

[6] OLIVEIRA F.A.R. E DE ARAUJO DIAS, A. de. *Verificação de Falhas em Linhas de Transmissão por Redes Neurais Artificiais a partir dos Espectros de Freqüência de Imagens*. Dissertação (Trabalho de graduação) — Universidade de Brasília, 2007.

[7] LANGE, S.; SUNDERHAUF, N.; PROTZEL, P. A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. In: *Advanced Robotics, 2009. ICAR 2009. International Conference on*. [S.l.: s.n.], 2009. p. 1 –6.

[8] JEON, D.; CHO, K.; KIM, D.-H. Vision-based autonomous landing for small-scale unmanned rotorcraft. In: *Object Component Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*. [S.l.: s.n.], 2011. p. 274–280.

[9] FAN, Y.; HAIQING, S.; HONG, W. A vision-based algorithm for landing unmanned aerial vehicles. In: *Computer Science and Software Engineering, 2008 International Conference on*. [S.l.: s.n.], 2008. v. 1, p. 993 –996.

[10] FUCEN SHI HAIQING, W. H. Z. The object recognition and adaptive threshold selection in the vision system for landing an unmanned aerial vehicle. In: *Proceedings of the 2009 IEEE International Conference on Information and Automation*. [S.l.: s.n.], 2009.

[11] SARIPALLI, S.; MONTGOMERY, J.; SUKHATME, G. Visually guided landing of an unmanned aerial vehicle. *Robotics and Automation, IEEE Transactions on*, v. 19, n. 3, p. 371 – 380, june 2003. ISSN 1042-296X.

[12] BOSCH, S.; LACROIX, S.; CABALLERO, F. Autonomous detection of safe landing areas for an uav from monocular images. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. [S.l.: s.n.], 2006. p. 5522 –5527.

[13] YUAN, Z. et al. A real-time vision-based guided method for autonomous landing of a rotorcraft unmanned aerial vehicle. In: *Mechatronics and Automation, 2005 IEEE International Conference*. [S.l.: s.n.], 2005. v. 4, p. 2212 – 2215 Vol. 4.

[14] BRADSKI, D. G. R.; KAEHLER, A. *Learning opencv, 1st edition*. First. [S.l.]: O'Reilly Media, Inc., 2008. ISBN 9780596516130.

[15] BOUGUET, J.-Y. *Camera Calibration Toolbox for Matlab*. July 2010. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[16] DORAN, C.; LASENBY, A. *Geometric Algebra for Physicists*. Cambridge: Cambridge Univ. Press, 2003.

[17] LAVALLE, S. M. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[18] KATO, H.; BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In: *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*. [S.l.: s.n.], 1999. p. 85 –94.

[19] KATO, H.; BILLINGHURST, M. *ARToolKit Documentation*. October 1999. `http://www.hitl.washington.edu/artoolkit/documentation/`.

[20] SOUZA, A. L. G.; OLIVEIRA, G. F. *SISTEMA DE LOCALIZAÇÃO PARA ROBÔ MÓVEL*. Universidade de Brasília, Brasília, DF. Brasil., 2011. TRABALHO DE GRADUAÇÃO.

[21] MALBEZIN, P.; PIEKARSKI, W.; THOMAS, B. Measuring artootkit accuracy in long distance tracking experiments. In: *Augmented Reality Toolkit, The First IEEE International Workshop*. [S.l.: s.n.], 2002. p. 2 pp.

[22] ARAÚJO, E. R. M. M.; KUSSABA, H. T. M. *SISTEMAS DE PERSEGUIÇÃO BASEADO EM QUADRIRROTORES*. Universidade de Brasília, Brasília, DF. Brasil., 2011. TRABALHO DE GRADUAÇÃO.

[23] SA, P. *Parrot AR.Drone A technological first*. 2012. `http://ardrone.parrot.com/parrot-ar-drone/en/technologies`.

[24] PARROT. *AR.Drone Developer Guide*. [S.l.], Fevereiro 2011. SDK 1.7.

[25] MITCHELL, M.; SAMUEL, A. *Advanced Linux Programming*. Thousand Oaks, CA, USA: New Riders Publishing, 2001. ISBN 0735710430.

[26] PARROT. *Parrot AR.Drone Quick Start Guide*. [S.l.].

[27] JOHNSON, D.; BERTHIAUME, C.; WITKOWSKI, B. *Augmented Reality - ARToolkit Patternmaker*. July 2002. `http://www.cs.utah.edu/gdc/projects/augmentedreality/`.

ANEXOS

I. PISTA DE POUSO

Este anexo tem como objetivo descrever a pista de pouso.

I.1 Construção

A concepção da pista de pouso foi um processo gradual. Inicialmente pensou-se em utilizar uma única marca de *ARToolKit* mas logo as limitações dessa biblioteca apresentadas na seção 3.3.1 motivou construir uma pista mais elaborada utilizando várias marcas de tamanhos diferentes de forma a alcançar uma faixa de alcance de altura maior.

Assim, a segunda ideia era uma pista de pouso com 4 marcas com uma marca no centro e as outras três em volta de forma que a camera pudesse capturar as 3 marcas de uma vez.

Em seguida, para melhorar ainda mais o alcance chegamos ao número de marcas utilizados atualmente que são o de 7 marcas: uma central, outras três de tamanho intermediário e mais três outras com o tamanho maior.

Finalmente, definiu-se que as marcas deveriam ser geometricamente dispostas na pista de forma que a pista final fosse circular. Nos preocupamos também com o tamanho desse círculo que não poderia ser muito pequeno de forma que sua área comportasse as sete marcas mas também não poderia ser muito maior que o VANT inviabilizando sua construção.

Dessa forma, a concepção final era de que cada uma dos dois conjuntos de três marcas deveriam ser tais que ao se ligar seus centros formaríamos triângulos equiláteros concêntricos e nesse centro estaria a marca central.

A pista foi, portanto, concebida de forma tanto científica como também artística onde outro aspecto considerado era também a aparência final, porque não? Assim, utilizando uma ferramenta de CAD, percorremos os seguintes passos que produziram o projeto da pista de pouso apresentada na figura I.1.

- Desenhamos um quadrado com lado igual a L onde seria colocado a marca central.
- Desenhamos em seguida a circunferência que circunscreve esse quadrado.
- Desenhamos o triângulo cuja circunferência anterior fica inscrita nele. O lado desse triângulo será o tamanho do lado dos quadrados das marcas de tamanho intermediárias.

  Contudo, ainda existe a preocupação de que haja um espaço em branco entre uma marca e outra para que o algoritmo de detecção do *ARToolKit* reconheça as bordas e encontre as marcas.

  Dessa forma, tomamos uma medida l_1 entre cada lado do triângulo desenhado nesse passo onde desenhamos os três quadrados que receberão as três marcas intermediárias.

- Agora medimos as distâncias entre as bordas de cada um desses quadrados e tomamos outro distanciamento l_2 de onde desenhamos os três demais quadrados que receberão as últimas três marcas de tamanhos maiores.

Note na figura I.1 que a pista de pouso é toda parametrizada e assim os três tamanhos do quadrado das marcas usadas relacionam-se entre si matematicamente. Quando escolhemos $L = 20mm$, $l_1 = 9mmel_2 = 14mm$ tivemos que os lados dos quadrados intermediários e maiores foram respectivamente $48.99mmme100,44mm$. Um é aproximadamente a metade do outro e por esse motivo escolhemos esses valores.

A figura I.1 mostra a pista de pouso parametrizada.

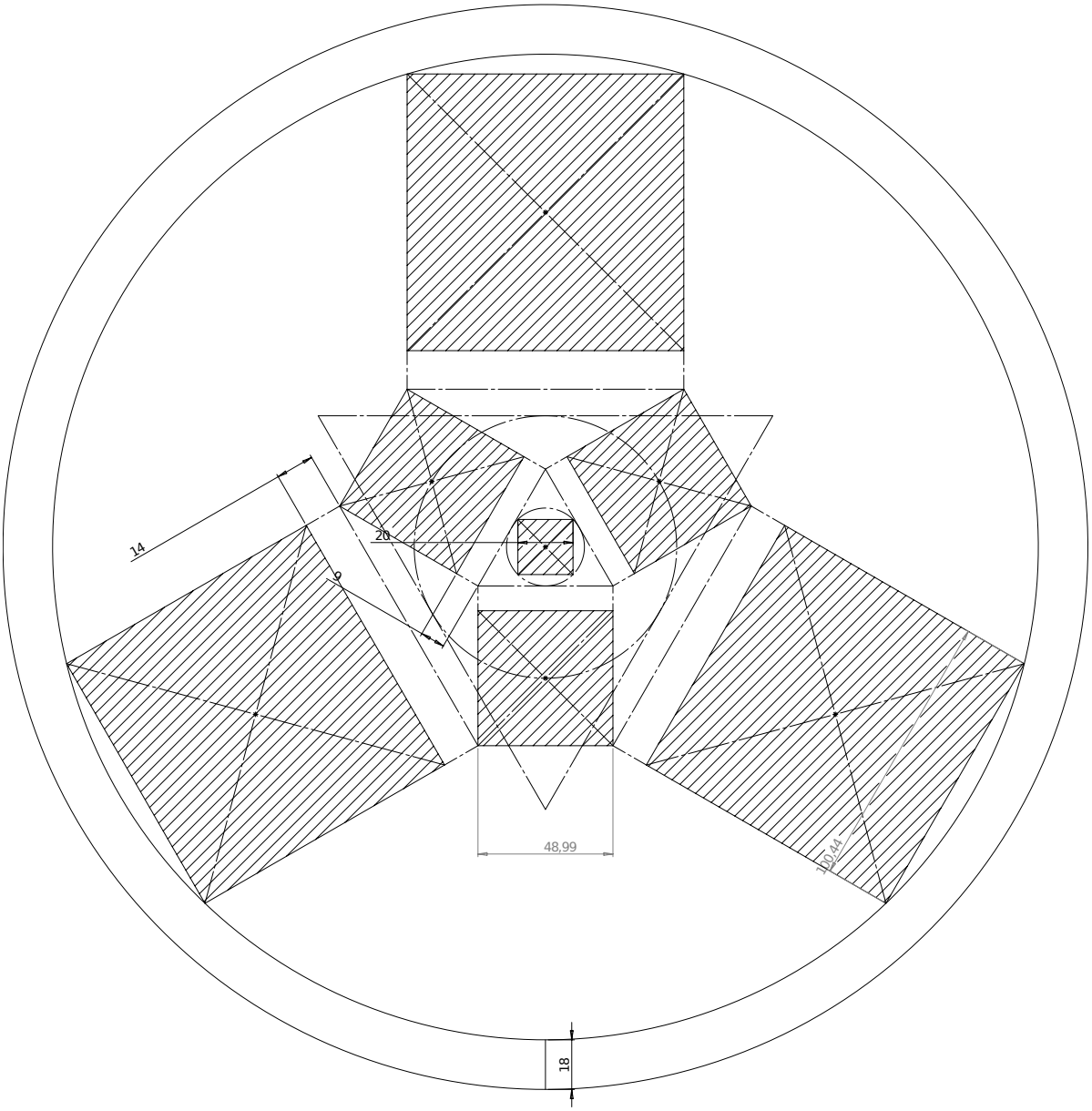


Figura I.1: Projeto da pista de pouso. Medidas em milímetros.

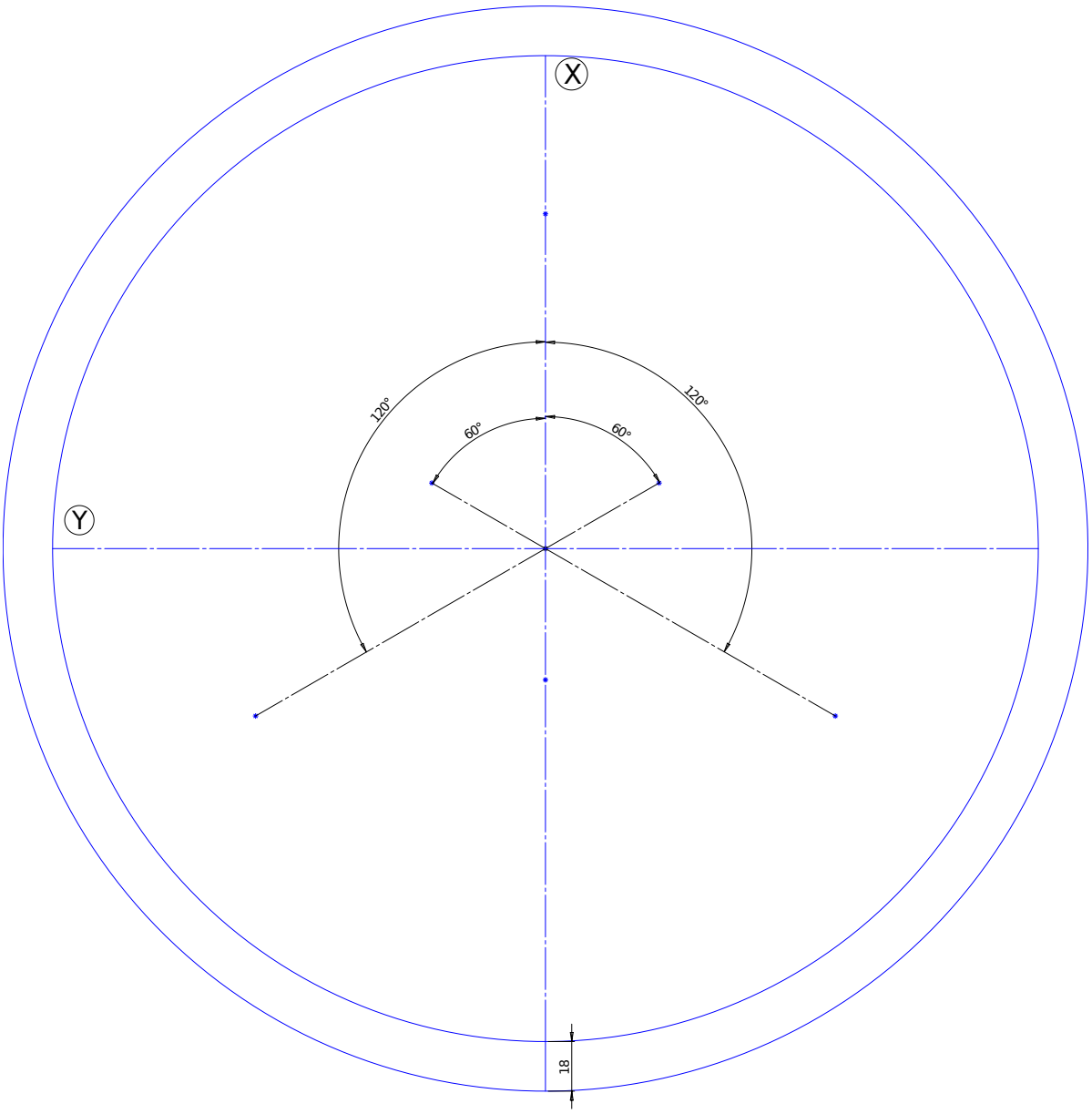


Figura I.2: Ângulos entre os centros das marcas. Medidas em graus.

A figura I.2 mostra os centros dos 7 quadrados que receberão as 7 marcas e as relações angulares entre elas.

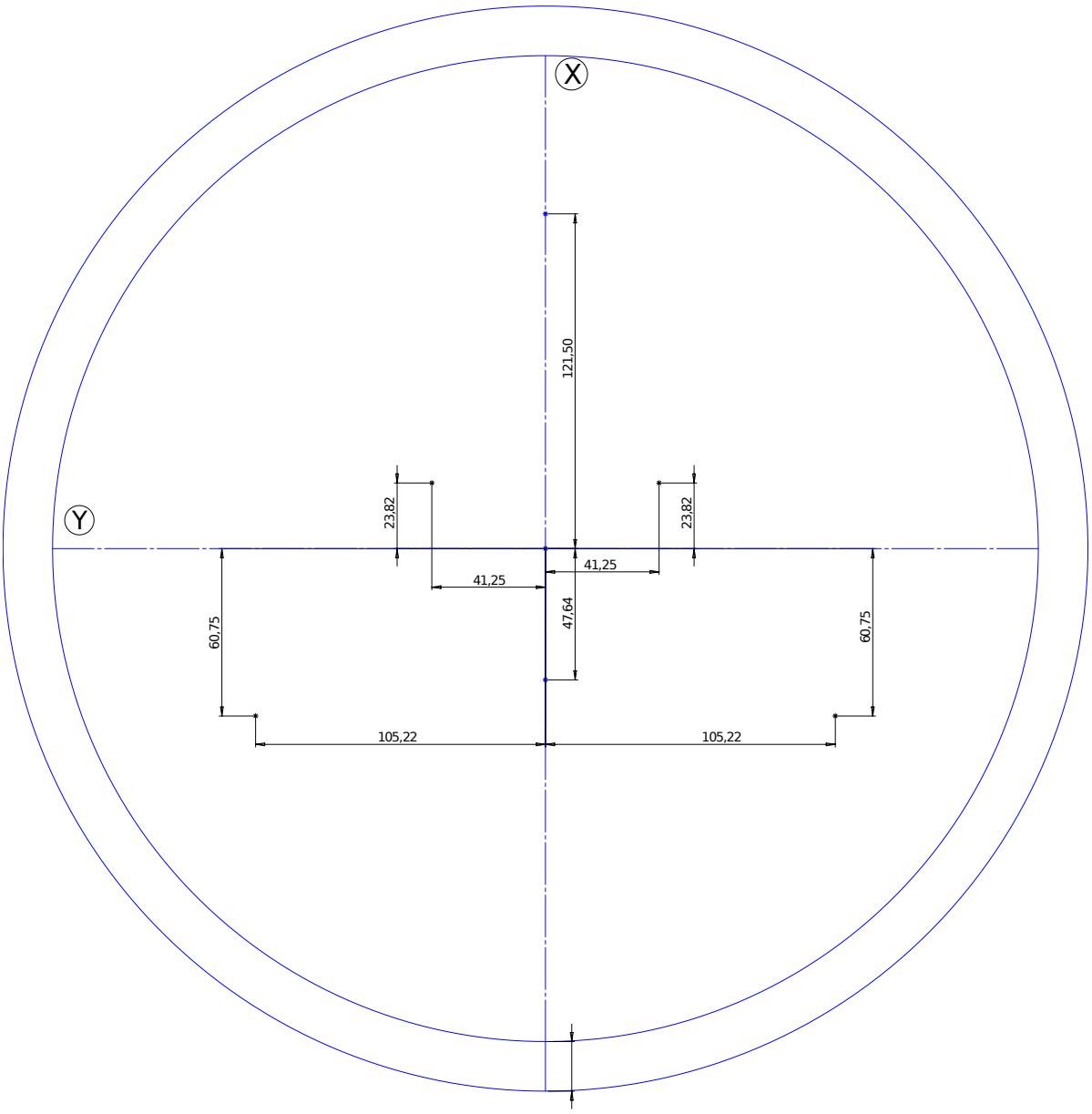


Figura I.3: Distâncias entre os centros das marcas. Medidas em milímetros.

A figura I.3 mostra os centros dos 7 quadrados e as distâncias entre elas e o centro da pista.

I.2 Marcas de *ARToolKit*

O próximo passo era escolher uma marca de *ARToolKit* para cada quadrado do projeto da pista de pouso e dar para cada uma dessas marcas um índice de identificação.

Como já adiantamos na seção 5.1 as 7 marcas de *ARToolKit* na pista de pouso receberam os índices de 0 a 6 sendo que a marca central recebeu o índice 0 e as demais marcas receberão os índices de 1 a 7 percorrendo a pista no sentido anti-horário. A figura que mostra isso foi repetida aqui para melhor visualização disso na figura I.4.

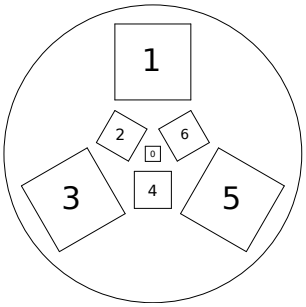


Figura I.4: Índices atribuídas a cada marca de ARToolKit que compõe a pista de pouso.

Mas porque isso feito dessa forma? Primeiro porque fica mais fácil lembrar qual é o índice de cada marca, basta contar as marcas no sentido anti-horário mencionado. E segundo porque isso faz com que todas as marcas internas possuam índice par enquanto que as marcas externas possuam índice ímpar e isso faz com que a pista fique organizada.

Em seguida, apresentamos agora as marcas geradas usando o programa *Pattern Maker* e escolhidas para fazer o papel de cada uma das marcas da pista nas figuras I.5 I.9 I.6 I.10 I.7 I.11 e I.8.

Figura I.5: Marca de *ARToolKit* de índice 0 também chamada de marca central.

Figura I.6: Marca de *ARToolKit* de índice 2 também chamada de marca interna esquerda.

Figura I.7: Marca de *ARToolKit* de índice 4 também chamada de marca interna traseira.

Figura I.8: Marca de *ARToolKit* de índice 6 também chamada de marca interna direita.

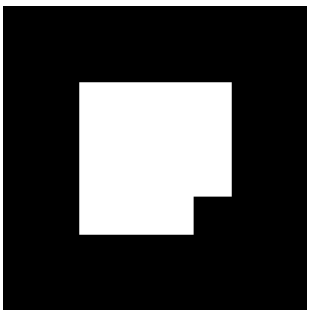


Figura I.9: Marca de *ARToolKit* de índice 1 também chamada de marca externa frontal.

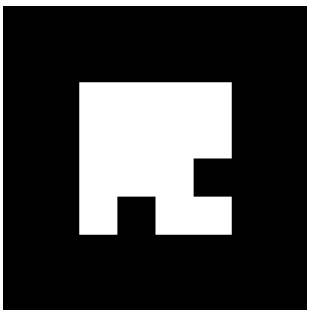


Figura I.10: Marca de *ARToolKit* de índice 3 também chamada de marca externa esquerda.

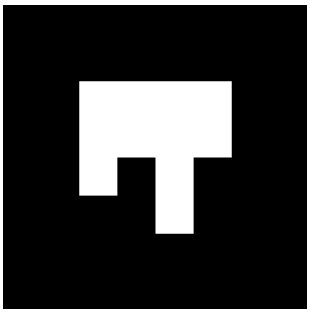


Figura I.11: Marca de *ARToolKit* de índice 5 também chamada de marca externa direita.

I.3 Disposição geométrica das marcas

Como dissemos na seção 3.4.2 as marcas de *ARToolKit* foram geradas pelo programa *Pattern Maker* que gera marcas em larga escala. Uma particularidade desse programa é que ele gera as marcas a partir de números, logo se pedimos para ele gerar a marca 1, por exemplo, ele sempre vai gerar a mesma marca. A tabela I.1 mostra quais foram esses números de *pattern* que geraram as marcas utilizadas na pista de pouso.

Índice	**Posição**	**Orientação**	**Numero do *Pattern***
0	Central	Centro	330
1	Externa	Frontal	1
3	Externa	Esquerda	17
5	Externa	Direita	86
2	Interna	Esquerda	444
4	Interna	Traseira	228
6	Interna	Direita	337

Tabela I.1: Tabela mostrando os índices das marcas, posição, orientação e o número do *pattern* correspondente.

Finalmente, as marcas foram dispostas na pista de pouso segundo o projeto já apresentado na figura I.1. Elas se encontram, em relação ao *frame* da marca central no centro da pista de pouso, transladadas e rotacionadas segundo mostra a tabela I.2.

Índice	**Posição**	**Lado[mm]**	c_x**[mm]**	c_y**[mm]**	yaw**[graus]**
0	Central	20	0	0	0
1	Externa	100,44	121,5	0	0
2	Interna	48,99	23,82	41,25	60
3	Externa	100,44	-60,75	105,22	120
4	Interna	48,99	-47,64	0	180
5	Externa	100,44	-60,75	-105,22	240
6	Interna	48,99	23,82	-41,25	300

Tabela I.2: Tabela mostrando o tamanho e posição relativa das marcas na pista de pouso.

Os ângulos de rotação e os centros c_x e c_y das marcas são tomados no referencial da pista de pouso com as rotações feitas no sentido da regra da mão direita aplicado ao eixo z, isto é, o sentido anti-horário. Note que a diferença entre as rotações de duas marcas não centrais de índices consecutivos é igual a sessenta graus. Os centros c_x e c_y foram medidos na pista final.

I.4 Configuração final

A figura I.12 mostra a pista de pouso final com todas as marcas nos seus devidos lugares.

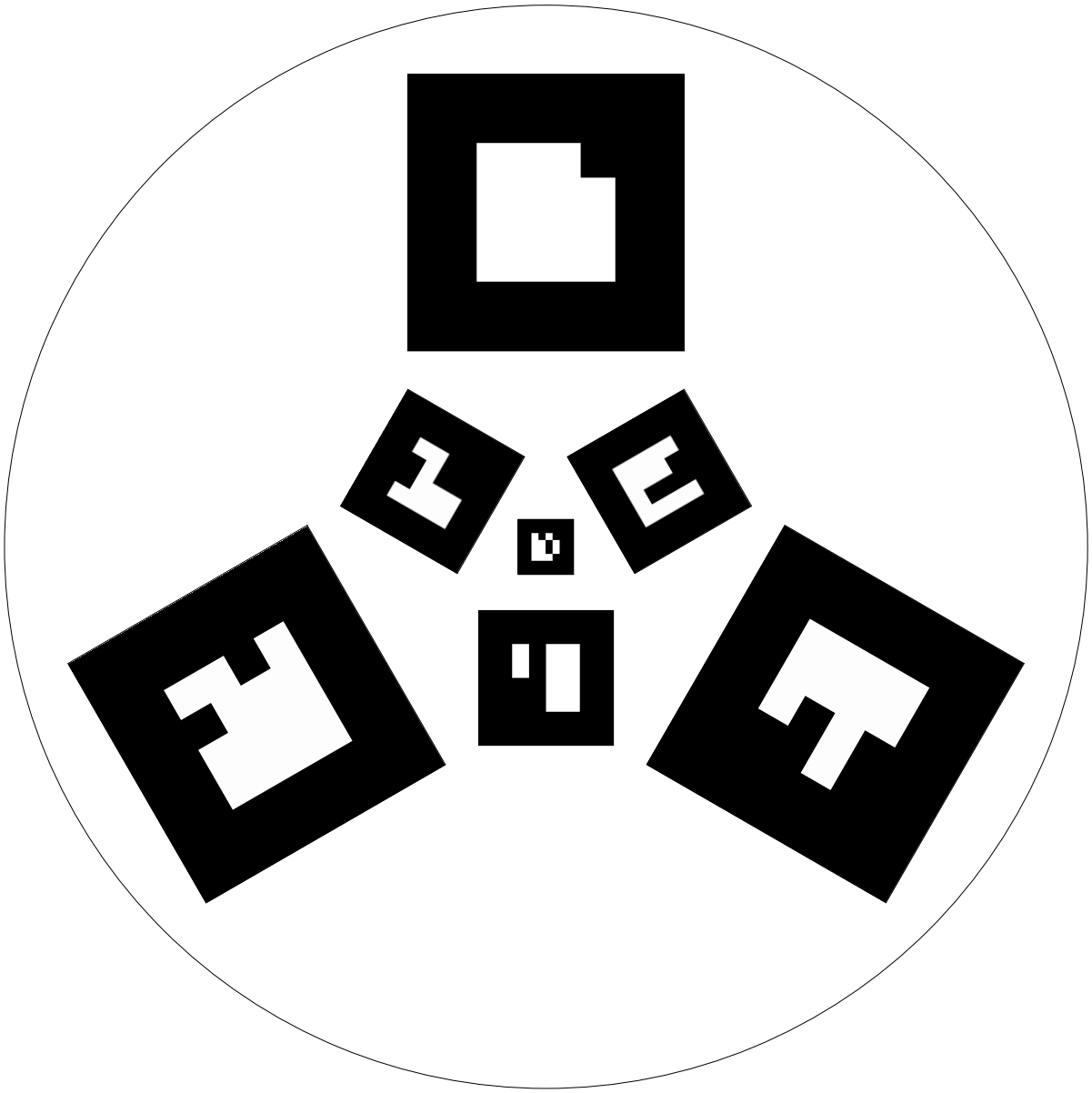


Figura I.12: Pista de pouso final

II. PROGRAMA VANTVISION

Este anexo tem como objetivo descrever o programa vantvision que é o sistema principal desenvolvido.

II.1 Implementação do Software

II.1.1 Introdução

Nessa seção apresentamos a metodologia utilizada para encontrar os parâmetros de interesse a partir da imagem e aspectos da implementação, isto é, a programação dos algoritmos responsáveis por atingir esse fim.

II.1.2 Metodologia

O sistema de visão foi desenvolvido segundo a metodologia apresentada no fluxograma da figura II.1.

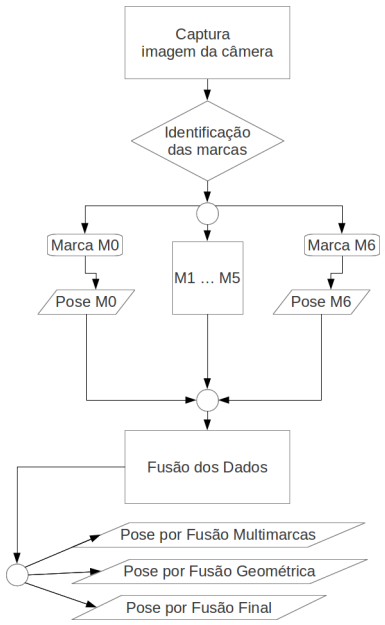


Figura II.1: Fluxograma geral

Inicialmente capturamos uma imagem da câmera. Em seguida procuramos identificar a presença de uma ou mais marcas nessa imagem. A biblioteca *ARToolKit* fornece para cada marca sua pose em relação à câmera. A partir desse conjunto de poses utilizamos três processos de fusão dos dados para estimar os parâmetros de interesse.

II.1.3 Estratégia de programação

Na seção 3.3.1 discutimos as limitações e problemas do *ARToolKit* que nos levou a pensar numa estratégia de programação para minimizar esses problemas. Dessa forma, optamos por uma estrutura de processamento multiprocesso de forma que os algoritmos responsáveis pela detecção de cada marca fossem independentes uns dos outros. Essa independência é positiva pois reduz a probabilidade de que esses problemas do *ARToolKit* ocorram em todas as marcas detectadas simultaneamente.

Há duas formas de se obter isso: a primeira é utilizar *threads* e a segunda é utilizar vários processos propriamente ditos separados com alguma forma de assegurar a comunicação entre eles e evitar situações de concorrência.

Como descreve [25] a decisão entre usar *threads* ou processos pode ser difícil. Nesse projeto optou-se por utilizar processos separados e memória compartilhada como forma de comunicação entre os processos por uma série de razões das quais podemos enfatizar:

- **Independência** - Cada processo executa independentemente dos demais. Isso é positivo em duas situações. A primeira é o já mencionado erro do *ARToolKit* quando ele inverte o eixo z. A segunda situação é a liberdade de instanciar ou não processos conforme a necessidade ou conforme a disponibilidade de recursos.
- **Modularização** - Como cada processo roda um executável diferente fica mais fácil modularizar o sistema criando um programa diferente para realizar cada função do sistema. Essa modularização facilita também a manutenção dos algoritmos visto que modificações são feitas no código fonte de um determinado processo deixando os demais inalterados.

Contudo, é importante apresentar também as desvantagens de se utilizar processos separados e descrever o que foi feito em relação a isso:

- **Concorrência** - Tanto sistemas que usam *threads* e vários processos podem ser prejudicados por situações de concorrência. Essas situações ocorrem quando dois ou mais processos tentam escrever em uma mesma área da memória ao mesmo tempo. Para evitar essas situações os processos desenvolvidos foram divididos em dois tipos: processos que somente escrevem na memória e processos que somente leem dados da memória compartilhada. Além disso, cada processo que escreve na memória possui sua área especial para escrever os seus dados. Ao utilizar essas regras evitamos situações de concorrência.
- **Sincronia** - Os processos instanciados passam a concorrer pelos recursos do sistema e seu funcionamento é assíncrono, ou seja, o escalonamento de que processo utilizará o processador a cada instante é tarefa do sistema operacional. Por exemplo, se o sistema operacional decidir que determinado aplicativo de vídeo que esteja sendo executado juntamente com os processos do sistema de visão possuem maior prioridade, ele pode alocar os recursos do sistema para o aplicativo de vídeo e deixar, por exemplo, a detecção de alguma das sete marcas da pista de pouso na fila. Contudo, mesmo com esse perigo, o usuário ainda possui alguns recursos

como pedir ao sistema para dar prioridade para alguns processos[1] e claro, usar o bom senso e não executar programas pesados juntamente com o algoritmo de visão.

II.2 Algoritmos desenvolvidos

Como já explicamos utilizamos uma abordagem multiprocesso para desenvolver o sistema de visão. Ao todo são 9 algoritmos a partir dos quais são instanciados processos filhos que trabalham em conjunto para produzir os resultados. A seguir vamos apresentar cada um desses algoritmos.

II.2.1 vantvision, *autopilot* e *modVision*

Os algoritmos *vantvision*, *autopilot* e *modVision* são versões diferentes do mesmo programa que apresentam a mesma funcionalidade. Eles são responsáveis por criar a memória compartilhada, capturar as imagens da fonte de imagens e compartilhar essas imagens com os demais processos. O fluxograma da figura II.2 mostra como isso é feito usando o processo vantvision.

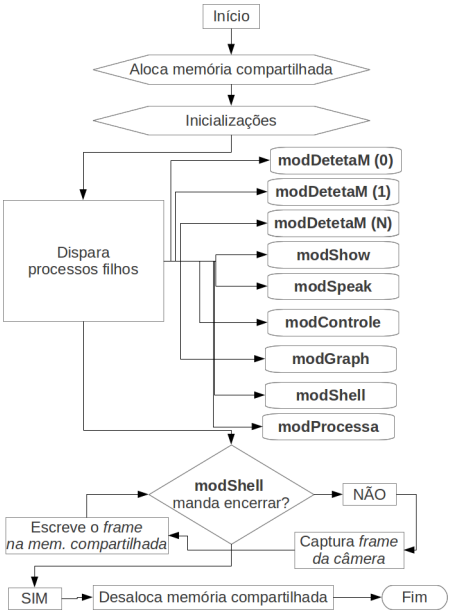


Figura II.2: Fluxograma do processo vantvision

O programa vantvision foi desenvolvido para testar o sistema sem precisar que a fonte de imagem seja de uma câmera acoplada no VANT. Ele fornece três opções de captura das imagens. A primeira é obter essas imagens de uma câmera que esteja conectado ao computador. A segunda é usar um arquivo de vídeo e a terceira é usar uma imagem. Ele também é responsável por compartilhar essas imagens na memória compartilhada e finalmente instanciar os demais processos filhos que farão a detecção das marcas, o processamento desses dados e a definição dos comandos de controle.

O programa *autopilot* foi adaptado do *AutoPylot* e tem a função de capturar a imagem da câmera do quadrotor e compartilhar. Além disso o *autopilot* também faz a interface com o qua-

[1] Na seção II.3.3 mostramos um comando criado para essa finalidade.

drotor para que seu controle automático possa ser feito ao ler os comandos de controle produzidos pelo *modControle*. Contudo, como não era nosso objetivo desenvolver um controlador, o programa *modControle* não é utilizado na versão atual deste sistema.

Finalmente, o programa *modVision* só é executado caso o *autopilot* o seja. Sua única função é instanciar os demais processos que compõe o sistema de visão.

II.2.2 *modDetectaM*

O programa *modDetectaM* é responsável por detectar alguma marca determinada na imagem que está sendo adquirida pelo vantvision ou pelo *autopilot*. É somente nele que utilizamos os códigos do *ARToolKit* para detectar a marca e extrair as variáveis de interesse.

Como a pista foi confeccionada com 7 marcas, o programa *modDetectaM* é instanciado em até 7 processos semelhantes que compartilham o mesmo código fonte mas que são chamados com parâmetros distintos de forma que cada processo seja responsável pela detecção de uma marca diferentes da pista. Cada processo procura pela sua marca e encontrando, escreve essas informações no seu local específico da memória compartilhada.

A figura II.3 mostra um fluxograma do processo.

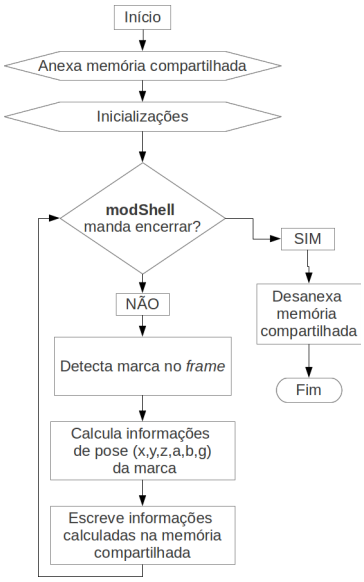


Figura II.3: Fluxograma do processo *modDetectaM*

II.2.3 modProcessa

O programa *modProcessa* é instanciado uma única vez assim como os demais programas com exceção do *modDetectaM*. O *modProcessa* é responsável por obter da memória compartilhada entre os processos todas as informações disponíveis e fazer os processos de fusão desses dados.

A figura II.4 mostra um fluxograma de seu algoritmo.

Até agora foi falado dos programas e consequentemente dos processos que são fundamentais

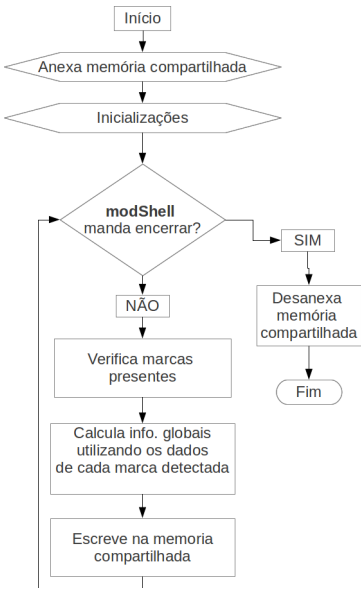


Figura II.4: Fluxograma do processo modProcessa

para o correto funcionamento do algoritmo de visão. Os demais programas apresentado a seguir não precisam estar rodando e só são usados caso o usuário queira monitorar os dados que o sistema de visão está produzindo.

II.2.4 *modGraph*

O *modGraph* é um processo que verifica na memória compartilhada os dados produzidos pelos 7 possíveis processos *modDetectaM* (um para cada marca da pista) e pelo processo *modProcessa* mostrando na tela um ou mais gráficos em tempo real com o histórico dos últimos dados produzidos. Seu fluxograma é apresentado na figura II.5.

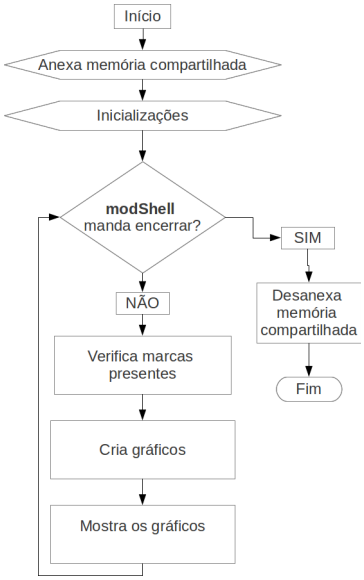


Figura II.5: Fluxograma do processo *modGraph*

O algoritmo *modGraph* cria em tempo de execução várias janelas onde mostramos gráficos dos valores monitorados. São até três tipos de janelas mostrando até três tipos diferentes de informações: janelas tipo gráfico de valores para monitorar os parâmetros x, y, z; janelas tipo medida angular para monitorar os ângulos α, β, γ e janelas tipo informação para apresentar informações relevantes.

Janela com gráficos

As janelas tipo gráficos (ver figuras II.6 e II.7) são constituídas de uma quadrado azul de fundo negro dentro do qual é renderizado um gráfico de barras. No canto superior esquerdo há dois valores na cor branca. O valor mais a esquerda é o valor instantâneo da variável. O valor mais a direita é o valor médio dos últimos 25 valores mostrados no gráfico. Esse valor médio funciona como um filtro de média para ajudar a ler o valor do parâmetro quando as medições encontram-se estáveis e sua posição no gráfico é marcado com uma linha vermelha horizontal. As barras são dispostas de forma que quanto mais à esquerda mais antiga no tempo é o dado e quanto mais à direita mais novo é o valor. Essas barras podem se dar nas cores vermelho claro, vermelho escuro, verde claro e verde escuro de acordo com o dado. A cor vermelha aparece caso o dado seja um valor negativo. A cor verde, caso positivo. As intensidades de cor claro/escuro é para indicar que o valor é ativo/inativo respectivamente.

As figuras II.6 e II.7 mostram 3 janelas do tipo gráfico monitorando os parâmetros x, y, z da pose do VANT em relação à pista de pouso. A primeira figura foi gerada quando a pista estava visível. A segunda, quando nenhuma das marcas da pista estavam sendo detectadas. Note as cores das barras.

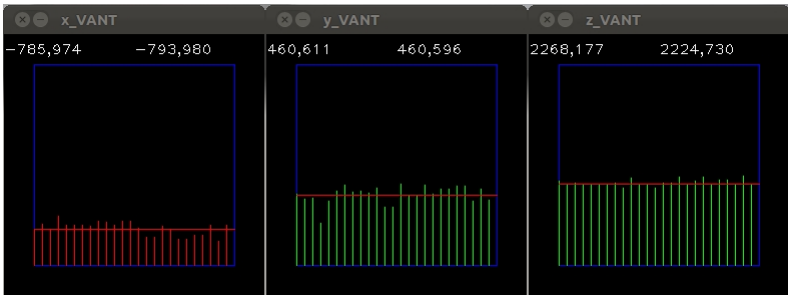


Figura II.6: Janelas mostrando parâmetros x, y, z do VANT encontrados.

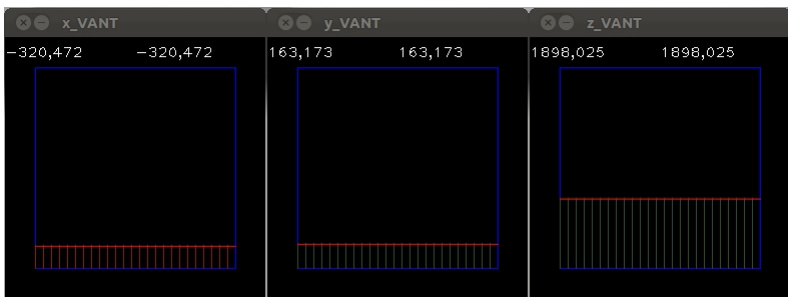


Figura II.7: Janelas mostrando parâmetros x, y, z do VANT perdidos.

Janela com medidas angulares

Quando o valor que queremos monitorar é um ângulo, fica mais fácil visualizar esse ângulo ao desenhar duas linhas que se cruzam formando o ângulo monitorado. Isso é feito nas janelas do processo *modGraph* do tipo medidas angulares. Um exemplo pode ser encontrado na figura II.8 onde monitoramos os parâmetros α, β, γ que correspondem às informações de *YAW*,*PITCH* e *ROLL* do VANT.

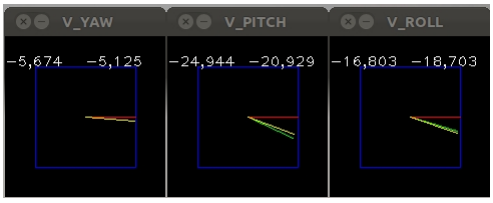


Figura II.8: Janelas mostrando as medidas angulares de parâmetros da pose.

Essas janelas de medidas angulares mostram um quadrado azul com fundo negro onde desenhamos um segmento vermelho indicanco a referência, ou seja, o ângulo zero.

Novamente, no canto superior esquerdo está o valor instantâneo da variável e mais à direita o valor médio dos últimos 25 valores assumidos. sentido de crescimento positivo dos valores é o senstido anti-horário. O segmento verde mostra o ângulo instantâneo daquela variável e o segmento amarelo o ângulo médio.

Janela com informações

Além de gráficos e de medidas angulares, o *modGraph* também pode mostrar na tela *strings* de texto com informações úteis. Na figura II.9 mostramos uma janela onde imprimi-se os índices das marcas detectadas num determinado momento.

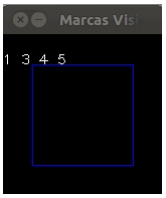


Figura II.9: Janela mostrando os índices das marcas detectadas num determinado momento.

Registros

Vale destacar, finalmente, que incorporamos ao *modGraph* a função de registrar os dados provenientes do *AR.Drone* e do sistema de visão. Esse recurso foi criado para criar os gráficos do capítulo 6. Para utilizá-lo, o usuário deve dar o comando no controle remoto ou no terminal descrito no anexo III.

II.2.5 *modShow*

O *modShow* apresenta na tela em tempo real as imagens que estão sendo capturadas pelo processo principal (*vantvision* ou *autopilot*) e desenha nessas imagens indicadores para facilitar o entendimento do que está acontecendo. Como exemplo, há um código padrão que mostra quais marcas foram detectadas desenhando uma linha vermelha que sai de um dos cantos da tela até o centro da marca. A figura II.10 mostra o fluxograma desse processo.

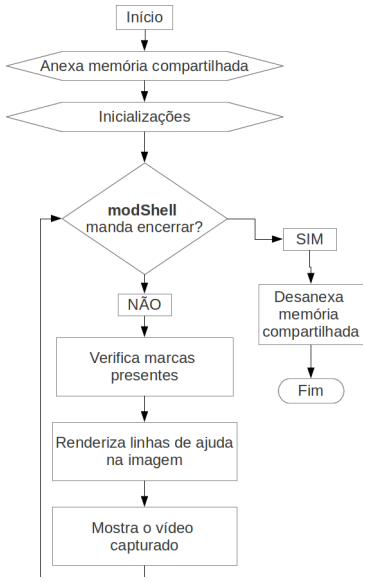


Figura II.10: Fluxograma do processo *modShow*

Além de mostrar as imagens o processo *modShow* também é responsável por capturar imagens e gravar vídeos através de comandos passados pelo processo *modShell* ou pelo processo *autopilot*. A figura II.11 mostra um exemplo da janela mostrando o vídeo obtido da *WebCam* na qual apenas algumas marcas da pista de pouso são detectadas e linhas vermelhas partindo do canto superior esquerdo e chegando no centro das marcas detectadas na imagem.

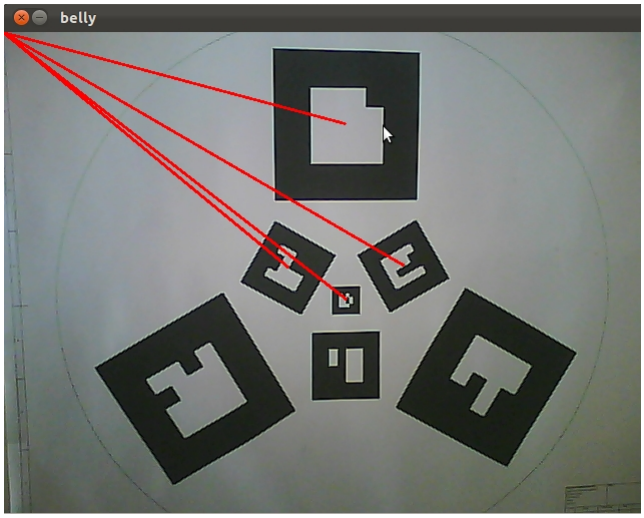


Figura II.11: Janela do processo *modShow* mostrando o vídeo e linhas guia

Tabela II.1: Eventos do processo *modSpeak*

Evento	**Frase**
Marca detectada	Marca N detectada
Marca perdida	Marca N perdida
SPEAK_GRAV_INIC	Gravação iniciada
SPEAK_GRAV_STOP	Gravação finalizada
SPEAK_BAT_LOW	Cuidado! Bateria baixa!
SPEAK_BUTT_PS	Cuidado! Cuidado! Encerrando Tudo!

II.2.6 *modSpeak*

O *modSpeak* como o próprio nome diz usa um sintetizador de voz nativo do sistema operacional *Linux Ubuntu* para avisar toda vez que uma marca foi detectada ou perdida. Além disso, ele também foi configurado para alertar determinados eventos como, por exemplo, situações em que a bateria do *AR.Drone* está baixa. O fluxograma é mostrado na figura II.12.

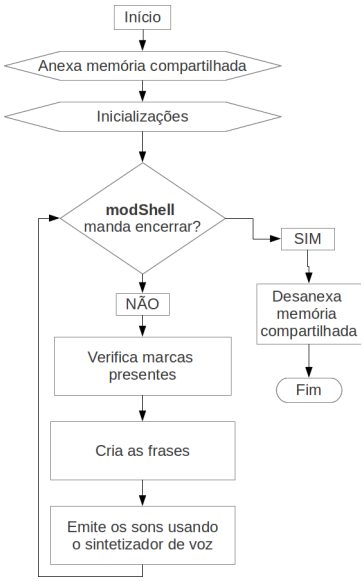


Figura II.12: Fluxograma do processo *modSpeak*

A tabela II.1 mostra algumas frases emitidas pelo sintetizador de voz quando determinados eventos ocorrem.

II.2.7 *modShell*

Finalmente o processo *modShell* é responsável por passar comandos e alterar parâmetros de configuração em tempo de execução além da função de dar o primeiro comando para encerrar cada um dos processos. O fluxograma desse processo pode ser visto na figura II.13.

O processo de encerramento do sistema de visão se dá em cascata com um processo sendo responsável por encerrar o outro. O processo *modShell* é responsável por encerrar o processo *modShow*, *modSpeak*, *modGraph* e *modProcessa* caso eles estejam rodando. Quando o processo

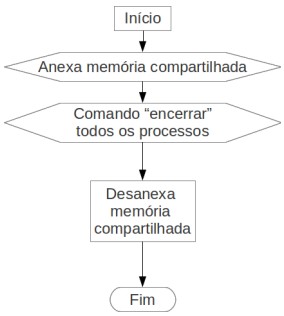


Figura II.13: Fluxograma do processo *modShell*

Tabela II.2: Comandos implementados no processo *modShell*

Argumento	**Função**
Sem argumentos	Encerra o sistema
p	Captura imagem
v	Inica/para gravação de vídeo
renice	Redefine prioridades dos processos
autoshoot	Grava imagens continuamente

modShow, por sua vez, recebe o comando de finalização ele é o responsável por mandar os processos *modDetectaM* finalizarem. Finalmente, o último processo *modDetectaM* que é finalizado é o encarregado de comandar o processo *vantvision* para deixar de capturar as imagens e encerrar (somente no caso dele ser o processo principal). Essa metodologia de encerramento é necessária para não deixar processos *zumbis* rodando na máquina indefinidamente e para que não ocorram problemas de acesso de memória pois como os processos *vantvision* e *autopilot* são os responsáveis pela criação e liberação da memória compartilhada, eles devem ser os últimos a finalizarem. Alguns dos demais recursos do processo *modShell* estão descritos na tabela II.2. O processo *autopilot* é o único que possui um pedido de encerramento independente já que o mesmo pode ser executado sem a necessidade de executar o sistema de visão.

II.3 Descrição do sistema de visão

Como o sistema é modularizado, desenvolvemos duas formas de utilizá-lo. A primeira forma é iniciando o sistema pelo processo vantvision. A segunda é pelo processo *autopilot* seguido do processo *modVision*. A seguir vamos mostrar essas duas formas de iniciar o sistema.

II.3.1 Iniciado pelo processo *vantvision*

Os comandos listados no trecho II.1 mostra o que acontece se executamos o executável *vantvision* com o argumento "-h".

Listing II.1: Verificando ajuda do executável vantvision

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
```

```
ronie@LapXRonie:~/prog/vantvision$ ./vantvision -h

>>> <vantvision 7247> iniciado <<<

 ./vantvision --> Captura da Camera 0
 ./vantvision numCamera --> Captura da Camera numCamera
 ./vantvision 1 video.avi --> Captura do Video
 ./vantvision 2 imagem.jpg --> Captura da imagem
ronie@LapXRonie:~/prog/vantvision$
```

O processo *vantvision* cria a memória compartilhada, chama os processos filhos e faz a aquisição das imagens ou da câmera especificada, ou do arquivo de vídeo especificado ou mesmo de uma única imagem estática. Isso foi bastante vantajoso na etapa de desenvolvimento do sistema pois não precisávamos ter um VANT preparado para gravar imagens e testar o sistema o tempo todo. os testes podiam ser feitos usando a própria *WebCam* do computador ou mesmo algum vídeo gravado anteriormente.

II.3.2 Iniciado pelo processo *autopilot* e pelo processo *modVision*

Na prática, queremos que a fonte de captura das imagens na qual o sistema de visão vai atuar seja proveniente do VANT. Dessa forma, alteramos os programas *autopilot* para criar a memória compartilhada e introduzimos um programa chamado *modVision* encarregado unicamente de instanciar os demais processos do sistema de visão.

Dessa forma, o processo de início do sistema de visão segue os seguinte passos:

1. Certificar-se que o AR.Drone está conectado ao computador. Uma dica é usar o comando mostrado na listagem II.2.

Listing II.2: Comando make ping

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ cd auto/
ronie@LapXRonie:~/prog/vantvision/auto$ make ping
ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=3.24 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=1.19 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=1.17 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=1.36 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=1.18 ms
64 bytes from 192.168.1.1: icmp_req=6 ttl=64 time=1.37 ms
64 bytes from 192.168.1.1: icmp_req=7 ttl=64 time=1.24 ms
^C
--- 192.168.1.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6004ms
rtt min/avg/max/mdev = 1.178/1.542/3.240/0.697 ms
```

O comando make ping verifica se o computador está conectado com o *AR.Drone*.

2. Conectar o controle de PS3 ao computador e apertar seu botão *PS* para certificar-se que o mesmo está ligado. Caso não tenha um controle, esse item pode ser desconsiderado.

3. Abrir um terminal e executar o executável *autopilot*. O controle deverá ser conectado, algumas inicializações deverão ocorrer e ele deverá começar a imprimir os dados do *NAVDATA* na tela.

Listing II.3: Executando o autopilot

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ cd auto/
ronie@LapXRonie:~/prog/vantvision/auto$ ./autopilot
Setting locale to en_GB.UTF-8
Wait authentification
Wait authentification
Wait authentification
=================> 192.168.1.1
Input device Sony PLAYSTATION(R)3 Controller found
Input device js0 added
Starting thread video_stage
Starting thread navdata_update
Starting thread ardrone_control
Video stage thread initialisation
Thread navdata_update in progress...
Timeout when reading navdatas - resending a navdata request on port 5554
[Navdata] Checksum failed : 742 (distant) / 678 (local)
VLIB new codec 32

 roots's memVV->imageData = 786448

 roots's memVV->imageData2 = 819217
ctrl state=     0 battery= 0% theta=+0.000000 phi=+0.000000 psi=+0.000000
vz= +0
ctrl state=     0 battery= 0% theta=+0.000000 phi=+0.000000 psi=+0.000000
vz= +0
```

4. Abrir um segundo terminal e executar o executável *modVision*.

Listing II.4: Executando o modVision

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ ./modVision
>>> <modVision 3174> iniciado <<<

numArgs = 7
0 - './modDetectaM'
1 - '0'
```

```
2 - '851986'
3 - '48'
4 - 'Data/4x4_48.patt'
5 - '21'
6 - 'Data/calibMatlab.dat'

numArgs = 7
0 - './modDetectaM'
1 - '1'
2 - '851986'
3 - '1'
4 - 'Data/4x4_1.patt'
5 - '100'
6 - 'Data/calibMatlab.dat'

numArgs = 7
0 - './modDetectaM'
1 - '2'
2 - '851986'
3 - '444'
4 - 'Data/4x4_444.patt'
5 - '49'
6 - 'Data/calibMatlab.dat'

numArgs = 7
0 - './modDetectaM'
1 - '3'
2 - '851986'
3 - '17'
4 - 'Data/4x4_17.patt'
5 - '100'
6 - 'Data/calibMatlab.dat'

numArgs = 7
0 - './modDetectaM'
1 - '4'
2 - '851986'
3 - '228'
4 - 'Data/4x4_228.patt'
5 - '49'
6 - 'Data/calibMatlab.dat'

numArgs = 7
0 - './modDetectaM'
1 - '5'
2 - '851986'
3 - '86'
4 - 'Data/4x4_86.patt'
5 - '100'
```

```
6 - 'Data/calibMatlab.dat'

numArgs = 7
0 - './modDetectaM'
1 - '6'
2 - '851986'
3 - '337'
4 - 'Data/4x4_337.patt'
5 - '49'
6 - 'Data/calibMatlab.dat'

numArgs = 2
0 - './modProcessa'
1 - '851986'

numArgs = 3
0 - './modShow'
1 - '851986'
2 - 'Data/calibMatlab.dat'

numArgs = 2
0 - './modGraph'
1 - '851986'

numArgs = 2
0 - './modSpeak'
1 - '851986'
```

O comando II.3 inicia o programa *autopilot* que é o programa *AutoPylot* modificado. Ele é encarregado de comunicar com o Parrot AR.Drone, criar a memória compartilhada e obter o *stream* de vídeo da câmera do *Drone* compartilhando essas imagens com os outros processos.

O comando II.4 por sua vez executa o processo *modVision* que por sua vez instancia os demais processos filhos responsáveis pelo restante das funções do sistema de visão.

II.3.3 Redefinindo a prioridade dos processos

Como discutido na seção II.1.3 criamos uma função que pede ao sistema operacional para priorizar os processos do sistema de visão em execução.

Com o programa *vantvision* ou com os programas *autopilot* e *modVision* executando. Abrir um novo terminal, ir na pasta vantvision do sistema e executar como *root*:

Listing II.5: Executando o renice

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ sudo su
[sudo] password for ronie:
```

```
root@LapXRonie:/home/ronie/prog/vantvision# ./modShell renice
>>> <modShell 3232> iniciado <<<

3195 (process ID) old priority 0, new priority -20
3201 (process ID) old priority 0, new priority -18
3202 (process ID) old priority 0, new priority -18
3203 (process ID) old priority 0, new priority -18
3204 (process ID) old priority 0, new priority -18
3205 (process ID) old priority 0, new priority -18
3206 (process ID) old priority 0, new priority -18
3207 (process ID) old priority 0, new priority -18
3208 (process ID) old priority 0, new priority -16
3209 (process ID) old priority 0, new priority -12
3210 (process ID) old priority 0, new priority -15
3211 (process ID) old priority 0, new priority -10

>>> <modShell 3232> finalizado <<<
```

O comando II.5 vai fazer com que o sistema operacional Ubuntu realoque as prioridades dos processos.

II.3.4 Finalização do sistema de visão

A finalização do sistema de visão consiste em executar o programa *modShell* sem argumentos a partir de qualquer terminal. Ele é responsável pelo primeiro comando de encerramento que desencadeia outras ordens de encerramento em cascata por parte dos outros processos.

Assim, basta abrir um terminal e executar o executável *modShell*.

Listing II.6: Executando o modShell

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ ./modShell

>>> <modShell 3325> iniciado <<<

<Shell> loopShow = 1

<Shell> loopSpeak = 1

<Shell> loopGraph = 1

<Shell> loopProcessa = 1

<Shell> loopCalibra = 0

<Shell> loopShow = 0
```

```
<Shell> loopSpeak = 0

<Shell> loopGraph = 0

<Shell> loopProcessa = 0

>>> <modShell 3325> finalizado <<<
```

O comando II.6 encerra todos os processos filhos rodando na máquina. Se o processo principal foi o *vantvision* ele deverá ser encerrado também. A listagem II.7 mostra o que ocorre no terminal que estava executando o processo *vantvision* e/ou demais processos filhos.

Listing II.7: Executando o modShell

```
>>> <modGraph 3210> finalizado <<<

>>> <modDetectaM 3204 3> finalizado <<<

<modDetectaM 3202 1> loopVV = 1

<modDetectaM 3202 1> loopVV = 0

>>> <modDetectaM 3202 1> finalizado <<<

>>> <modDetectaM 3206 5> finalizado <<<

>>> <modDetectaM 3205 4> finalizado <<<

>>> <modDetectaM 3203 2> finalizado <<<

>>> <modDetectaM 3201 0> finalizado <<<

>>> <modSpeak 3211> finalizado <<<

>>> <modDetectaM 3207 6> finalizado <<<
```

Contudo, se o processo principal for o *autopilot* esse deverá se encerrado em separado utilizando o botão *PS* do controle. Na ausência de um controle, o usuário ainda pode entrar na pasta do executável *autopilot* e dar o seguinte comando.

Listing II.8: Encerrando o processo autopilot

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ cd auto/
ronie@LapXRonie:~/prog/vantvision/auto$ ./make kill
```

Tanto o comando de encerrar dado pelo terminal ou pelo controle fará com que o processo *autopilot* encerre com a seguinte mensagem mostrada na listagem II.9.

Listing II.9: Mensagem final ao encerrar o processo autopilot

```
ctrl state=131072 battery=63% theta=+0.014100 phi=-0.006210 psi=-0.097028 alti
+0 vz= +0
ctrl state=     0 battery=63% theta=+0.014080 phi=-0.006170 psi=-0.097039 alti
+0 vz= +0
Video stage thread ended
```

II.4 Conclusão

Em todos os testes realizados o sistema apresentou resultados estáveis. Nos testes realizados verificou-se que ele consome pouca memória principal e a abordagem multiprocessos funciona bem no ambiente Linux Ubuntu. Além disso, essa abordagem possibilita que o sistema seja adaptado para atender a necessidades específicas de cada VANT já que os processos podem ser instanciados conforme a necessidade. Sua desvantagem está na dependência de um sistema operacional capaz de escalonar os processos de forma que eles executem em conjunto. E, apesar de alguns recursos como o comando "renice" apresentado, esse escalonamento é controlado pelo sistema operacional.

III. PROGRAMA AUTOPILOT

O programa *autopilot* foi criado alterando-se o código fonte original do programa *AutoPylot*. Este anexo tem como objetivo descrever a compilação, execução e os comandos de controle implementados.

III.1 Compilação e Execução

A compilação é feita executando o comando *make* dentro do diretório do autopilot como mostra a listagem III.1.

Listing III.1: Compilação

```
ronie@LapXRonie:~$ cd prog
ronie@LapXRonie:~/prog$ cd vantvision/
ronie@LapXRonie:~/prog/vantvision$ cd auto/
ronie@LapXRonie:~/prog/vantvision/auto$ make
```

A execução correta do programa segue os seguintes passos que já foram mostrados no anexo II.

1. Desconectar-se da conexão de internet wi-fi atual.
2. Ligar o *Parrot AR.Drone* conectando o cabo da bateria ao cabo da aeronave.
3. Esperar o computador descobrir a *hotspot* criado pelo *AR.Drone*.
4. Conectar o computador a esse *hotspot*.
5. Verificar a conexão com o *AR.Drone* rodando o comando *make ping*.
6. Se o comando enviar e receber pacotes com sucesso. Encerrar o processo de *ping* pressionando *Ctrl + C* no teclado.
7. Executar o programa autopilot.

O programa será executado e você deverá ver uma tela como a da figura que mostra os dados recebidos pelo NAVDATA.

Para encerrar o programa pressione o botão PS do controle PS3. Na ausência deste, entre em outro terminal, entre na pasta do programa autopilot e execute o comando *make kill* que irá encerrar o programa.

III.2 Controle

Nessa seção mostraremos os novos comandos programados no arquivo *gamepad.c* do projeto *autopilot*.

O controle do *Parrot AR.Drone* é feito usando um controle de *Playstation 3* o *DUALSHOCK 3 Wireless Controller* mostrado na figura III.2. É importante destacar a necessidade de conectar o cabo USB do controle ao computador e apertar o botão PS do controle para que o mesmo funcione e seja reconhecido pelo sistema operacional Linux. O cabo é necessário o tempo todo.

Figura III.1: Controle *DUALSHOCK* 3 utilizado para controlar a aeronave.

Figura III.2: Controle *DUALSHOCK* 3 - vista dos botões R1,R2,L1 e L2.

A tabela III.1 mostra as funcionalidades relacionadas para cada botão. A tabela III.2 mostra as funcionalidades relacionadas para os cada um dos eixos do controle no modo normal. Como veremos o botão *BUTTON_LA* muda o modo de controle dos eixos para aqueles mostrados na tabela III.3.

É importante tecer algumas observações:

- O comandos para capturar imagens e gravar vídeos são executados pelo processo *modShow*, logo só funcionarão quando o programa *modVision* também tiver sido executado.
- O comando do controle que encerra o programa autopilot envia por segurança o comando de emergência que vai desligar os motores da aeronave e assim no caso de ela estar voando ela não sairá por aí sem controle.
- Como pode ser visto pelas tabelas III.2 e III.3 o botão *BUTTON_LA* só dá ao usuário mais uma opção de controle de voo ao tornar possível a comutação das funcionalidades dos eixos 1 e 5 do controle durante o voo. No modo invertido o usuário consegue guiar a aeronave mantendo-se uma dada altitude.
- Os eixos 3 e 4 que não aparecem na tabela não são utilizados e portanto são ignorados.
- Toda vez que o comando de emergência for usado, o usuário deve apertar o botão de emergência uma segunda vez para desligar o modo de emergência.

Botão	Descrição	Função
BUTTON_RIGHT	Direita	Animação de LEDs
BUTTON_LEFT	Esquerda	Animação de LEDs
BUTTON_DOWN	Baixo	Altura máxima limitada a 5 metros
BUTTON_UP	Cima	Altura máxima ilimitada
BUTTON_LA	*Left Axis Click*	Muda o modo de controle axial
BUTTON_R1	*Right 1*	Captura tela
BUTTON_L1	*Left 1*	Inicia/Pausa gravação de vídeo
BUTTON_ZAP	*SQUARE*	Troca a camera
BUTTON_BOLA	*CIRCLE*	Comando *Flat Trim*
BUTTON_START	*START*	Decola/Pousa
BUTTON_SELECT	*SELECT*	Emergência
BUTTON_AUTO	*X*	Liga/Desliga registro automático dos dados
BUTTON_TRI	*TRIANGLE*	Coleta um único registro dos dados
BUTTON_PS	*PLAYSTATION*	Encerra o programa autopilot

Tabela III.1: Tabela mostrando os botões do controle e as funcionalidades correspondentes implementadas.

Eixo	Descrição	Função
0	esquerdo vertical	Controla o angulo de arfagem (frente/traz)
1	esquerdo horizontal	Controla o angulo de rolamento (esquerda/direita)
5	direito vertical	Controla a velocidade vertical (sobe/desce)
2	direito horizontal	Controla a angulo de guinada (rotação)

Tabela III.2: Tabela mostrando os eixos do controle e as funcionalidades correspondentes implementadas no modo normal.

- Finalmente, não custa avisar que essas alterações são experimentais. Use-as sobre o seu próprio risco. Recomenda-se testar com muito cuidado os comandos antes de realizar voos com elevadas altitudes.

Eixo	Observação	Função
0	esquerdo vertical	Controla o angulo de arfagem (frente/traz)
1	esquerdo horizontal	Controla a angulo de guinada (rotação)
5	direito vertical	Controla a velocidade vertical (sobe/desce)
2	direito horizontal	Controla o angulo de rolamento (esquerda/direita)

Tabela III.3: Tabela mostrando os eixos do controle e as funcionalidades correspondentes implementadas no modo invertido.

IV. CONTEÚDO DO CD

Este anexo tem como objetivo descrever o conteúdo do CD. No CD encontramos as seguintes pastas que serão explicadas a seguir.

```
/
├── doc
├── prog
│   ├── criaCalibraDat
│   ├── criaSerieImagens
│   ├── modDetecta
│   │   ├── Data
│   │   └── Videos
│   ├── modDetectaBasico
│   │   └── Data
│   └── vantvision
│       ├── auto
│       │   ├── bak
│       │   └── shoot
│       │       └── imagens
│       ├── coletas
│       ├── Data
│       ├── imagens
│       ├── registros
│       └── videos
├── sistema
│   ├── ardrone_autopylot
│   ├── ARDrone_SDK_Version_1_8_20110726
│   ├── ARToolKit
│   ├── ARToolKit Install Files
│   └── patternMaker
└── videos
    ├── validacao
    └── ensaios
```

IV.1 Pasta doc

Na pasta *doc* encontramos a documentação do sistema de visão desenvolvido. Ela é composta dos seguintes arquivos:

- **monografiaVANTVISION.pdf** - Uma cópia desta monografia em formato pdf.
- **artigoVANTVISION.pdf** - Uma cópia do artigo científico.
- **docVANTVISION.pdf** - Uma documentação mais detalhada abordando cada arquivo da pasta */prog/vantvision* e da pasta */prog/vantvision/auto* além de instruções de instalação do sistema de visão desse trabalho.
- **docAutoPylot.pdf** - Uma cópia da documentação do programa original *AutoPylot*.

- **pistaMarcasEscalaReal.pdf** - Pista de pouso em escala real a ser impressa em papel tamanho A2.

IV.2 Pasta prog

A pasta *prog* contém todos os programas desenvolvidos nesse trabalho. Recomenda-se copiá-la para a pasta *home* do usuário e usá-la dessa forma. A seguir descrevemos as subpastas.

IV.2.1 Pasta criaCalibraDat

A pasta *criaCalibraDat* contém um programa escrito para criar o arquivo de calibração *calibMatlab.dat* indispensável para que o sistema faça boas medições dos parâmetros da pose. Mais informações na documentação disponível em "/doc/docVANTVISION.pdf" do CD.

IV.2.2 Pasta criaSerieImagens

A pasta *criaSerieImagens* contém um programa escrito para criar a figura 2.3 do capítulo 2.

IV.2.3 Pasta modDetectaBasico

A pasta *modDetectaBasico* contém o programa escrito com as funções mais básicas necessárias para detectar uma marca de ARToolKit já utilizando *openCV* em vez de *openGL*.

IV.2.4 Pasta modDetecta

A pasta *modDetecta* contém uma alteração do programa *modDetectaBasico*. Nessa pasta está um programa utilizado para testar os algoritmos principais do sistema final *vantvision*. O programa *modDetectaM* final foi escrito a partir deste programa.

IV.2.5 Pasta vantvision

A pasta *vantvision* contém os arquivos dos programas que integram o sistema de visão *vantvision* desenvolvido nesse trabalho. A explicação de cada arquivo dessa pasta está no arquivo *doc/docVANTVISION.pdf*.

IV.2.5.1 Pasta auto

A subpasta *vantvision* contém os arquivos do programa *autopilot* que foi feito adaptando-se o código fonte do programa *AutoPylot* original. É importante destacar que para o correto funcionamento do sistema, essa pasta deve ficar dentro da pasta *vantvision* para que os dois programas possam compartilhar o arquivo *key* usado para transimitir entre os processos a chave da memória compatilhada. Mais informações no arquivo *doc/docVANTVISION.pdf*.
As subpastas *bak* contém *backups* originais do programa original *AutoPylot* e não foram modificadas.
A subpasta *shoot* não é mais utilizada, mas ela pode ser usada como destino de captura de tela realizadas pelo programa *autopilot* caso o usuário descomente alguns comentários do código fonte.

IV.2.5.2 Subpastas da pasta vantvision

Na subpasta *coletas* são armazenados os arquivos de coletas experimentais do programa *modGraph*.
Na subpasta *registros* são armazenados os arquivos de registros experimentais do programa *modGraph*.

Na subpasta *imagens* são armazenados as imagens de captura de telas do programa *modShow*.
Na subpasta *videos* são armazenados os videos capturados pelo programa *modShow*.
Na subpasta *Data* estão os arquivos de treinamento *.patt* de cada uma das marcas da pista de pouso utilizadas e o arquivo de calibração da câmera chamado "calibMatlab.dat" . Esses arquivos são lidos pelos processos *modDetectaM* do sistema. Mais informações consultar *doc/docVANTVISION.pdf*.

IV.3 Pasta sistema

Na pasta *sistema* estão as bibliotecas e arquivos de sistema utilizadas pelo sistema de visão. A descrição de instalação dessas bibliotecas está no arquivo *doc/docVANTVISION.pdf* ou em documentação própria de cada um [19, 24, 26, 27].

IV.3.1 Pasta ardrone_autopylot

A pasta *ardrone_autopylot* contém os arquivos do programa *AutoPylot* original a partir do qual foi criado o programa *autopilot* que faz a comunicação com o *AR.Drone*. Seus arquivos são explicados em documentação própria. Uma cópia dessa documentação está no arquivo *doc/docAutoPylot*.

IV.3.2 Pasta ARDrone_SDK_Version_1_8_20110726

A pasta *ARDrone_SDK_Version_1_8_20110726* contém os arquivos de sistema do SDK do *Parrot AR.Drone* utilizados nesse trabalho. Mais informações é encontrado em documentação própria. Consulte [24, 26].

IV.3.3 Pasta ARToolKit

A pasta *ARToolKit* contém os arquivos de sistema da bilioteca de realidade aumentada *ARToolKit* utilizadas. Para mais informações consultar documentação própria [19].

IV.3.4 Pasta ARToolKit Install Files

A pasta *ARToolKit Install Files* contém os arquivos de instalação do sistema da bilioteca de realidade aumentada *ARToolKit* utilizadas. Para mais informações consultar documentação própria [19] e também a documentação da pasta "doc" do CD.

IV.3.5 Pasta patternMaker

A pasta *patternMaker* contém os arquivos do programa *Pattern Maker* utilizados para gerar as marcas de *ARToolKit* usadas na pista de pouso. Consultar [27].

IV.4 Pasta videos

A pasta *videos* contém os vídeos ilustrativos criados para mostrar o funcionamento do sistema desenvolvido. São as seguintes pastas contendo os seguinte vídeos:

Pasta validacao

Contém vídeos de validação dos resultados mostrando os gráficos de cada um dos parâmetros enquanto a aeronave é manipulada em cima da pista de pouso e durante um voo *indoor*, isto é, realizada num espaço fechado.

- **vidIntro.wmv** - Introdução aos vídeos.
- **vidZ.wmv** - Vídeo mostrando a estimativa do parâmetro z.
- **vidXY.wmv** - Vídeo mostrando a estimativa dos parâmetros x e y.
- **vidALPHA.wmv** - Vídeo mostrando a estimativa do parâmetro α.
- **vidBETA.wmv** - Vídeo mostrando a estimativa do parâmetro β.
- **vidGAMMA.wmv** - Vídeo mostrando a estimativa do parâmetro γ.
- **vidINDOOR1.wmv** - Primeiro vídeo mostrando todos os parâmetros durante um voo *indoor*.
- **vidINDOOR2.wmv** - Segundo vídeo mostrando todos os parâmetros durante um voo *indoor*.

Pasta ensaios

Contém vídeos feitos durante ensaios experimentais em que a aeronave é colocada para voar de forma que possamos registrar a cada momento todos os dados do *NAVDATA* e do sistema de visão para posteriormente estuda-los comparativamente.

- **vidOUTDOOR1.wmv** - Voo outdoor.
- **vidOUTDOOR2.wmv** - Outro voo outdoor.

V. MONTAGEM EXPERIMENTAL

Este anexo tem como objetivo descrever a montagem experimental utilizada para realizar os experimentos.

V.1 Ensaios dos parâmetros da pose

Para realizar os experimentos foi necessário montar uma estrutura que permitisse controlar o valor do parâmetro que se desejava medir.
O sistema de suspensão do VANT é constituído pelos seguintes itens:

- Caixa na qual o VANT é colocado. As figuras V.1(a) e V.1(b) mostram a caixa do *Parrot AR.Drone* que foi cortada de forma que tanto a câmera vertical quando o altímetro ultrassônico pudessem ficar direcionados para baixo. Dessa forma, o VANT pode ser içado enquanto faz imagens da pista de pouso. O VANT dentro da caixa é mostrado nas figuras V.1(c) e V.1(d).

(a) Caixa suspensa

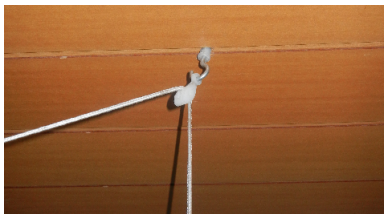
(b) Caixa suspensa sobre pista

(c) *AR.Drone* na caixa.

(d) *AR.Drone* suspenso.

Figura V.1: Montagem para experimentos

- Sistema gancho fixo no teto, polia e corda para içar a caixa mostrado nas figuras V.2(a) e V.2(b).



(a) Gancho fixo no teto e polia.

(b) Corda presa na caixa com um gancho auxiliar.

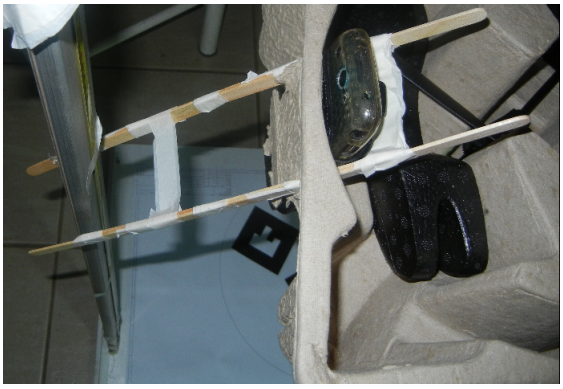
Figura V.2: Elementos da montagem

V.2 Ensaios do parâmetro z com a régua

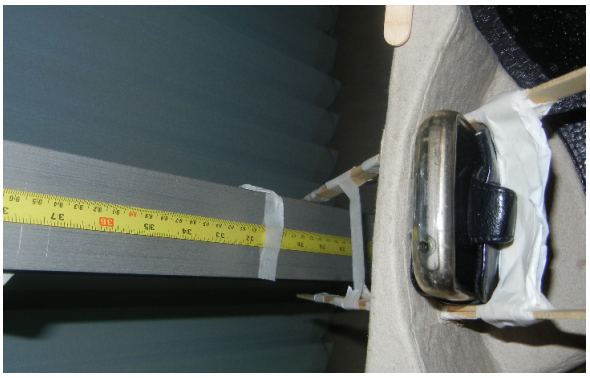
A montagem padrão apresentada na seção anterior serviu para fazer os ensaios tanto contínuos quanto estáticos do parâmetro z. No caso dos ensaios estáticos em que os valores foram lidos numa régua utilizamos como auxilio a seguinte montagem:

- Uma régua de até 2,4 metros foi colocada ao lado da caixa conforme mostra a figura V.3.
- Uma câmera de IP foi colocada dentro da caixa mirando a régua com auxilio de duas hastes de madeira para guiar a câmera e impedir que a caixa içada rodasse (figuras V.4(a) e V.4(b)).
- Os dados vistos pela câmera IP eram vistos por um outro computador (ver figura V.4(c)) para facilitar a visão da medida. Cabe destacar que todas as medições realizadas dessa forma possuíam uma distância entre o ponto em que era realizada a medição até o centro focal da câmera vertical que depois eram descontadas para a plotagem dos gráficos.

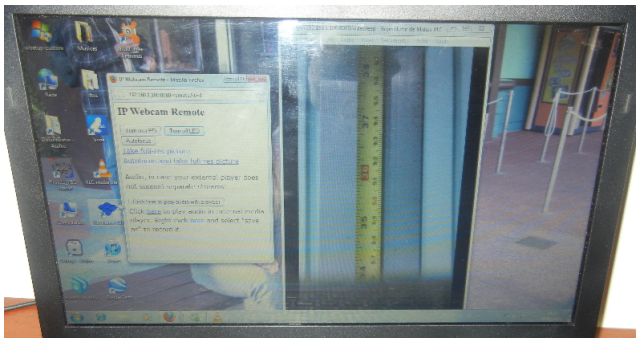
Figura V.3: Régua



(a) Câmera de IP mirando a régua.



(b) Outra foto da câmera de IP mirando a régua.



(c) Computador visualizando a imagem da câmera de IP

Figura V.4: Montagem experimental com régua.

VI. ARTIGO

Uma cópia do artigo “ESTIMAÇÃO DE ALTURA POR VISÃO MONOCULAR UTILIZANDO *ARTOOLKIT* COM MULTIMARCAS PARA ATERRISSAGEM DE VANTS” submetido à XIX edição do Congresso Brasileiro de Automática - CBA 2012, promovido pela Sociedade Brasileira de Automática (SBA), é apresentado nesse anexo.