Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Leandro Lima Gomes

Lucas Pires Leal

Controle de um Quadricóptero por Servovisão

Rio de Janeiro

2014

Leandro Lima Gomes

Lucas Pires Leal

Controle de um Quadricóptero por Servovisão

Projeto de graduação apresentado, como requisito parcial para obtenção do grau de Engenheiro Eletricista, à Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Tiago Roux de Oliveira

Coorientador: Prof. José Paulo Vilela Soares da Cunha

Rio de Janeiro

2014

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

G633 Gomes, Leandro Lima.
Controle de um Quadricóptero por Servovisão / Leandro Lima Gomes; Lucas Pires Leal. – 2014.
108f.

Orientador: Tiago Roux de Oliveira.
Coorientador: José Paulo Vilela Soares da Cunha.
Projeto Final (Graduação) - Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.
Bibliografia p.70-71.

1. Engenharia Elétrica. 2. Aeronave não tripulada. 3. Visão computacional. I. Oliveira, Tiago Roux de. II. Cunha, José Paulo Vilela Soares. III. Universidade do Estado do Rio. IV. Título.

CDU 621.3

Assinatura

Data

Leandro Lima Gomes

Lucas Pires Leal

Controle de um quadricóptero por servovisão

Projeto de graduação apresentado, como requisito parcial para obtenção do grau de Engenheiro Eletricista, à Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro.

Aprovado em:.

Banca Examinadora:

_____

Prof. Tiago Roux de Oliveira (Orientador)
Faculdade de Engenharia – UERJ

_____

Prof. José Paulo Vilela Soares da Cunha (Coorientador)
Faculdade de Engenharia – UERJ

_____

Prof. Alessandro Jacoud Peixoto
Escola Politécnica – UFRJ

_____

Prof. Andrei Giordano Holanda Battistel
COPPE – UFRJ

Rio de Janeiro

2014

DEDICATÓRIA

À toda minha família e companheira Juliana Amendola, pelo amor dedicado a mim, por acreditar em meu potencial e me dar forças para a conclusão deste curso, pois sem o incentivo e paciência deles não seria possível a realização deste projeto.

Lucas Pires Leal

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus. Pela vida maravilhosa a qual me foi ofertada, pelas grandes conquistas que possibilitou em minha vida e pelas pessoas incríveis a quem tive a grande honra de conhecer ao longo do meu caminho, e sem as quais nada seria.

Agradeço especialmente aos meus pais Aristides Gomes e Azenete Lima Gomes, principais colaboradores do meu sucesso, pelas palavras confortadoras nos momentos de dificuldade e pelo apoio incondicional à minha formação como cidadão.

Agradeço a minha querida amiga, companheira e esposa Jiulianne Pereira, que me ensina a cada dia a beleza de amar, me ajuda a superar os problemas e a ser feliz. Pela compreensão nos momentos de pressão durante o desenvolvimento deste projeto.

Agradeço ao laboratório PROSAICO, com especial carinho aos professores Lisandro Lovisolo e Michel Tcheou que muito me apoiaram e disponibilizaram um ambiente agradável durante todo esse período de trabalho.

Por fim, agradecimentos especiais aos meus orientadores, Professor Tiago Roux e Professor José Paulo Vilela Soares da Cunha. Um muito obrigado pelo estímulo constante, pelo conhecimento ofertado e principalmente por acreditarem em meu potencial.

Leandro Lima Gomes

AGRADECIMENTOS

Aos meus orientadores, Prof. Tiago Roux de Oliveira e Prof. José Paulo Vilela Soares da Cunha por toda a ajuda e orientação, apontando os melhores caminhos, dando estímulos para o desenvolvimento deste trabalho e pela amizade demonstrada nesses anos.

Aos professores do PROSAICO, pelas dicas e por fornecer a estrutura do laboratório para o estudo e desenvolvimento deste projeto.

Aos meus amigos de FMC Technologies pela paciência com meus estudos e pelos conselhos que foram essenciais para o bom andamento deste trabalho.

A UERJ, porque sem ela não poderia ter realizado este sonho de conquista.

A todos aqueles, que embora não citados nominalmente, contribuíram direta e indiretamente para a execução deste trabalho.

Lucas Pires Leal

RESUMO

GOMES, Leandro Lima, LEAL, Lucas Pires. *Controle de um Quadricóptero por Sservovisão.* 2014. 108 f. Projeto Final (Graduação em Engenharia Elétrica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2014.

Atualmente, o uso de veículos aéreos não tripulados (VANTs) como base em pesquisas acadêmicas vem aumentando gradativamente ao longo dos anos devido, principalmente ao seu baixo custo de implementação. Este projeto visa o desenvolvimento de um algoritmo de controle de um veículo aéreo não tripulado por servovisão, utilizando um sistema de captura de movimentos como sensor.Utilizando o quadricóptero AR.Drone como veículo e o sistema de captura Vicon, foi desenvolvido um algoritmo que integra essas duas plataformas a fim de gerar ações de controle para a resolução de problemas em servovisão robótica e controle. Baseando-se no problema de rastreamento de alvo, onde o objetivo é fazer o veículo seguir um alvo móvel, foi implementado um controle Proporcional-Derivativo (PD) em tempo discreto onde o sinal de controle é a diferença entre a posição do alvo e a do veículo, obtidas a partir do sistema Vicon.

Palavras-chave: Quadricóptero. Servovisão. Controle Proporcional-Derivativo (PD). AR.Drone. Sistema Vicon.

ABSTRACT

GOMES, Leandro Lima, LEAL, Lucas Pires. Control of a Quadrotor by visual servoing. 2014. 108 f. Projeto Final (Graduação em Engenharia Elétrica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2014.

Currently, the use of unmanned aerial vehicles (UAV's) on academic research has increased over the years due, mainly, to their low cost of implementation.This project aims to develop an algorithm to control an unmanned aerial vehicle by visual servoing using a motion capture system as a sensor. By using the quadcopter AR.Drone a mobile robot and the Vicon System as sensor, an algorithm has been develop, which integrates these platforms to generate the control actions for solving problems involving robotics visual servoing and control. Based on the target tracking problem, where the goal is to make the vehicle to follow a moving target, was implemented a Proportional Derivative control in discrete time where the control signal is the difference between the target position and the vehicle, measured from the Vicon System.

Keywords: Quadcopter. Visual servoing. PD Control. AR.Drone. Vicon Motion capture system.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
LED	Light Emitting Diode
PD	Proporcional Derivativo
RAM	Random Access Memory
ROV	Remotely Operated underwater Vehicle
SDK	Software Development Kit
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UERJ	Universidade do Estado do Rio de Janeiro
VANT	Veículos Aéreos Não Tripulados
WI-FI	Wireless Fidelity

SUMÁRIO

1 INTRODUÇÃO

A utilização de veículos aéreos não tripulados (VANT's ou UAV's – *Unmmaned aerial vehicles)* vem se tornando frequente em pesquisas acadêmicas, pelo baixo custo de implementação e a possibilidade de realização de tarefas onde a presença humana é dispensável ou até mesmo impossibilitada.

Os veículos aéreos apresentam inúmeras formas de configuração estrutural, podendo se apresentar como *Quadricópteros* (veículos aéreos com quatro rotores com pás rotativas) e do tipo "avião", que apresentam asas fixas, conforme Figura 1 e Figura 2.

Figura 1 - UAV de pás rotativas: Quadricóptero
Fonte: www.microdrones.com

Figura 2 - UAV de asas fixas: Avião
Fonte:http://commons.wikimedia.org/wiki/File:NASA_ALTUS_UAV.jpg

A principal característica dos UAV's é que estes podem ser autônomos, isto é, seu controle é realizado por uma programação previamente carregada em seu sistema, ou controlado remotamente. O último modo é o mais empregado hoje em dia, no qual uma pessoa fica responsável em controlar a maioria das ações remotamente, característica essa que fez com que essas aeronaves se popularizassem em atividades onde o risco ao ser humano é alto, como em ações militares e inspeções em áreas de risco.

Na área militar, os UAV's, comumente denominados de *Drones*, são de aplicação extensiva e bem consolidada, sendo utilizados desde 1991 em missões de reconhecimento (Haulman, 1991-2003) e mais recentemente na realização de operações de intervenção. Nos últimos anos, a expansão dessa tecnologia tem sido

muito rápida, principalmente devido à redução de custos de componentes eletrônicos, tais como microprocessadores, sensores e atuadores, inclusive para uso na área civil, destacando-se nos seguintes segmentos[1] :

- Vigilância de áreas de fronteira;
- Inspeção de oleodutos, gasodutos e linha de transmissão elétrica;
- Enlace de comunicações e cobertura de eventos para as redes de TV;
- Controle de animais indesejáveis em fazendas e quintais vizinhos às florestas;
- Controle de safras agrícolas;
- Controle de queimadas;
- Vigilância aérea das vias rodoviárias para dissuadir, detectar e identificar eventuais pontos críticos, bem como para apoiar a gestão e o planejamento de infraestrutura;

1.1 Quadricópteros

O quadricóptero, ou helicóptero quadrirrotor, é um veículo aéreo não tripulado que utiliza-se de quatro motores com hélices para se locomover no ar. Estes rotores são fixos à uma estrutura em formato de “X” e geralmente no centro fica localizada a central de processamento e alguns sensores, porém essas características podem ser alteradas de acordo com a necessidade de sua utilização.

O estudo dessa tecnologia vem ganhando notoriedade no mundo acadêmico, principalmente por ter uma mecânica simples, boa manobrabilidade e uma carga útil (capacidade de carga do veículo), o que lhe dá uma possibilidade de adicionar sensores e outros periféricos, como câmeras de vídeo, entre outros. Entretanto, o principal desafio para a utilização do quadricóptero é o seu controle, pois o quadricóptero possui uma instabilidade inerente, além da pouca autonomia das baterias que lhe fornecem energia, com duração média de 15 minutos de vôo.

---

[1] UAV Visions Aeronatics – http://www.uavision.com.

Uma característica que é bem explorada nesses veículos, é a questão da forma de operação, na qual pode se operar de forma autônoma, ou seja, um controle pré-determinado é gravado em sua memória, e o veículo atua sem controle ativo humano, ou da forma remotamente operada, onde há uma interface com o ser humano que participa ativamente no controle de suas ações à distância.

Os quadricópteros industriais, utilizados para aplicações específicas, em sua maioria militares, possuem componentes de tecnologias avançadas e por isso, são essencialmente caros, o que inviabiliza sua utilização em pesquisas acadêmicas. Entretanto, nos últimos anos, foram lançados veículos quadrirrotores com propósito recreativo, que dada sua sofisticação em termos de componentes eletrônicos e materiais, foram adotados como plataformas de pesquisa , como é o caso do AR.Drone (Figura 3) (Ahn, 2011) comercializado pela empresa francesa *Parrot* e o *Draganfly* (Figura 4) (Podhradsky, 2012), comercializado pela empresa Canadense *RC Toys*.

Figura 3 - AR.Drone
Fonte:http://www.mln.com.au/product/?itemID=3895

Figura 4 - DraganFly
Fonte: http://novus2.com/uav360/tag/aviation/

Em algumas universidades a pesquisa com essa tecnologia já está bem desenvolvida, como na Universidade da Pensilvânia (Mellinger, et al., 2010), em que um conjunto de quadricópteros foi utilizado para realizar manobras precisas inerentes em tarefas complexas, e também na Universidade técnica de Praga (Krajník, et al., 2011), na qual foi desenvolvido uma plataforma de pesquisa em robótica. Outros trabalhos também são de destaque em áreas como navegação

autônoma visual com desvio de obstáculo (Eresen, et al., 2012) e rastreamento visual por pontos de fuga (Gomez-Balderas, et al., 2012).

O Instituto de Sistemas Dinâmicos e Controle do Instituto Federal de Tecnologia de Zurique, através do grupo de pesquisa do Professor Raffaello D'Andrea, vem desenvolvendo uma série de aplicações usando veículos quadrirrotores. A *Flying Machine Arena* (Figura 5) uma arena com dimensões de 10mx10mx10m e mais uma sala de controle anexa, ainda contando com uma infraestrutura com câmeras de captura de movimentos, espumas de absorção de impactos e plataformas de recarga de baterias, é onde são testados e validados todos os algoritmos de controle proposto na universidade. Nela foram desenvolvidos diversos trabalhos no campo de construção robótica aérea (Willmann, et al., 2012), habilidades esportivas[2] e geração de trajetórias sem colisões de um frota (Augugliaro, et al., 2012).

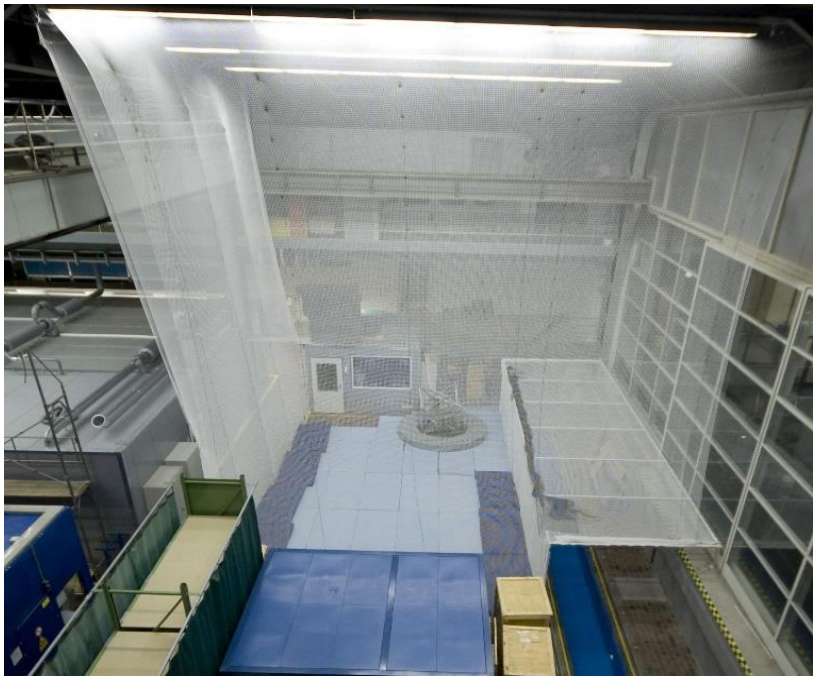

Figura 5 - Flying Machine Arena
Fonte: http://www.idsc.ethz.ch/Research_DAndrea/Flying_Machine_Arena/infrastructure

A cooperação de veículos aéreos para realizações de tarefas é um tema bastante promissor tendo em vista as possibilidades de utilização dos veículos aéreos para que trabalhando em, conjunto possam manipular e transportar cargas

[2] http://www.ted.com/talks/raffaello_d_andrea_the_astounding_athletic_power_of_quadcopters.html

que apenas um veículo não conseguiria (Figura 6). As aplicações desta área de pesquisa incluem desde missões de resgate de indivíduos em situações de emergência, transporte de carga perigosa, entre muitas outras que envolve o problema de içamento de cargas (Michael, et al., 2011).

Figura 6 - Cooperação entre veículos
Fonte: http://phys.org/news198481187.html

1.2 Servovisão Robótica

Em robótica, a técnica que utiliza realimentação visual para controle de um robô é chamada de Servovisão (em inglês, *Visual Servoing*), que consiste em utilizar como sensor câmeras que capturam e processam imagens para obtenção de informações e geração dos sinais para controle do robô.

Essa técnica surgiu a partir da convergência de diversas áreas, tais como computação em tempo real, processamento digital de sinais e teoria de controle, entre outras (Hutchinson, et al., 1996), e hoje em dia é amplamente utilizada para o controle de manipuladores robóticos (Figura 7), controle de UAV's em busca de um alvo (em inglês, *visual servoing target search*), (Darma, et al., 2013) e controle de veículos remotamente operados

(ROV – *Remoted operated vehicle*) (Marchand, et al., 2011), vide Figura 8.

Figura 7 - Manipulador cirúrgico
Fonte: (Marchand, et al., 2011)

Figura 8 - Braço Robótico ROV
Fonte: (Darma, et al., 2013)

1.3 Objetivo do Projeto

O objetivo proposto com o presente projeto é integrar o sistema de captura de movimento Vicon System com o quadricóptero AR.Drone, criando uma plataforma de pesquisa para o controle de robôs móveis utilizando servovisão. Esta integração se dará por meio de algoritmos computacionais, que irão gerar ações de controle para o quadricóptero a partir de informações de posição obtidas pelo sistema de câmeras. A partir dessa plataforma desenvolvida, será possível o desenvolvimento futuro de pesquisa que foque na concepção de novos algoritmos de controle, visto que a integração do sistema já estará estabelecida.

1.4 Organização do texto

O texto é composto por sete capítulos organizados da seguinte maneira:

- Capítulo 1: introduz o trabalho e faz uma breve descrição sobre o tema proposto;
- Capítulo 2: Apresenta a plataforma AR.Drone e suas características;
- Capítulo 3: descrição do esquema de servovisão utilizando o sistema Vicon de câmeras e suas aplicações;

- Capítulo 4: apresentação de como será realizada a integração dos sistemas:
- Capitulo 5: será apresentada a modelagem dinâmica do quadricóptero.
- Capítulo 6: será desenvolvida a técnica de controle utilizada para resolução do problema;
- Capítulo 7: apresentação dos resultados obtidos com o controlador proposto;
- Capítulo 8: Conclusões obtidas com o projeto, possíveis melhorias e trabalhos futuros.

2 A PLATAFORMA AR.DRONE

Este capítulo descreve a plataforma AR.Drone como o veículo quadrirrotor adotado no projeto, descrevendo seu hardware, software e seu modelo dinâmico usado para seu controle.

2.1 Introdução

O AR.Drone é um quadricóptero desenvolvido pela empresa francesa Parrot em 2010, inicialmente projetado para ser um brinquedo de alta tecnologia, utilizando como controle remoto, um aplicativo nas plataformas *iOS e Android* (Sistemas operacionais de plataformas móveis desenvolvido pela *Apple* e *Google*, respectivamente). Com diversas aplicações na área de jogos de realidade aumentada, foi crescente a popularização desta tecnologia como ambiente de desenvolvimento de jogos e aplicativos para plataformas móveis.

2.2 Mecânica e *Hardware*

O AR.Drone, como é um veículo quadrirotor, possui um estrutura em formato de “X” de fibra de carbono e partes de material pástico. Possui em cada extremidade um motor sem escovas de 35000 RPM e 15W de potência, além de um módulo de controle eletrônico composto por um microcontrolador de 8 bits e um conversor A/D (Analógico para Digital) de 10 bits, que são alimentados por uma bateria de polímero de Lítio de 1000mAh e 11,1V.

2.3 Sensores

Localizada no centro e embaixo da estrutura, uma unidade de navegação composta por diversos sensores é responsável por garantir estabilidade de voo. Ela é composta por:

- Um sensor ultrassônico para medir altitude;
- Um altímetro para aumentar a acurácia e corrigir variações de altitude;
- Um acelerômetro digital de 3 eixos;
- Um giroscópio de 2 eixos e um giroscópio piezoelétrico de precisão;
- Um magnetómetro de 3 eixos;

O sensor ultrassônico fornece medidas de altitude do veículo para a estabilização automática de altura. Já a câmera, localizada ao lado do sensor ultrassônico, é responsável por fornecer medidas de velocidade do solo, para o seu controle de estabilização interno. O funcionamento dos sensores estão exemplificados na Figura 9.

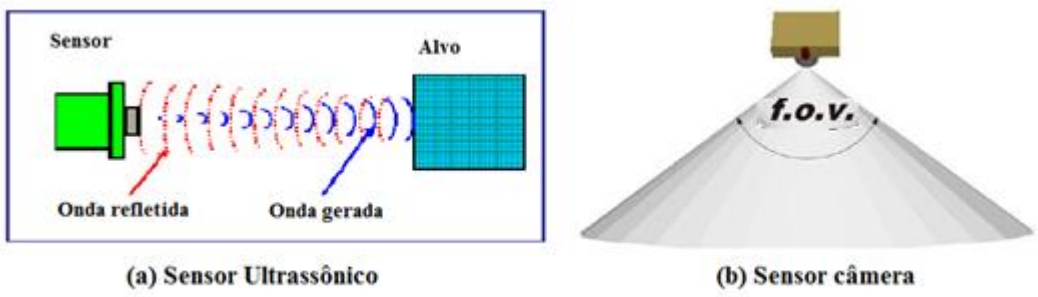

Figura 9 – Funcionamento dos sensores ultrassônico e câmera
Fonte: AR.Drone SDK Manual 1.8 - Adaptado

Junto com a unidade de navegação, se localiza a unidade central de controle, responsável por todo o processamento de informações do AR.Drone. Esta é composto por um microcontrolador Parrot P6, com uma CPU (Unidade Central de Processamento, em inglês) ARM926 de 32 Bits e 486MHz de frequência de operação com 128MB de memória RAM e 128MB de memória flash. Nessa unidade

fica também o módulo de conexão *wireless*, responsável por gerenciar o ponto de acesso de transmissão e recepção de dados sem fio.

O AR.Drone é equipado com duas câmeras de vídeos, sendo uma localizada na parte frontal, de resolução de 640x480 pixels, com sensor *CMOS* e lente com 93° de angulação diagonal e uma câmera de resolução 176x144 pixels também equipada de um sensor CMOS, localizada na parte de baixo do veículo.

Marcações pintadas no corpo do veículo (Figura 10) podem ser detectadas pela câmera frontal, e podem ser utilizadas para a detecção de outro AR.Drone, por exemplo. Além destas marcações, são disponibilizados marcadores adesivos para a detecção de um objeto qualquer pelo AR.Drone.

Figura 10 - Marcações para detecção
Fonte: AR.Drone SDK Manual 1.8 - Adaptado

Quando utilizado em ambientes fechados, o AR.Drone necessita de uma estrutura (Figura 11) que protege os motores e suas hélices contra avarias. Essa estrutura é feita de poliestireno, que por ser leve, não altera a dinâmica de voo do veículo.

Para a utilização em ambientes externos, não é recomendável a utilização desta estrutura pois ela aumenta o arrasto aerodinâmico provocado pelo vento.

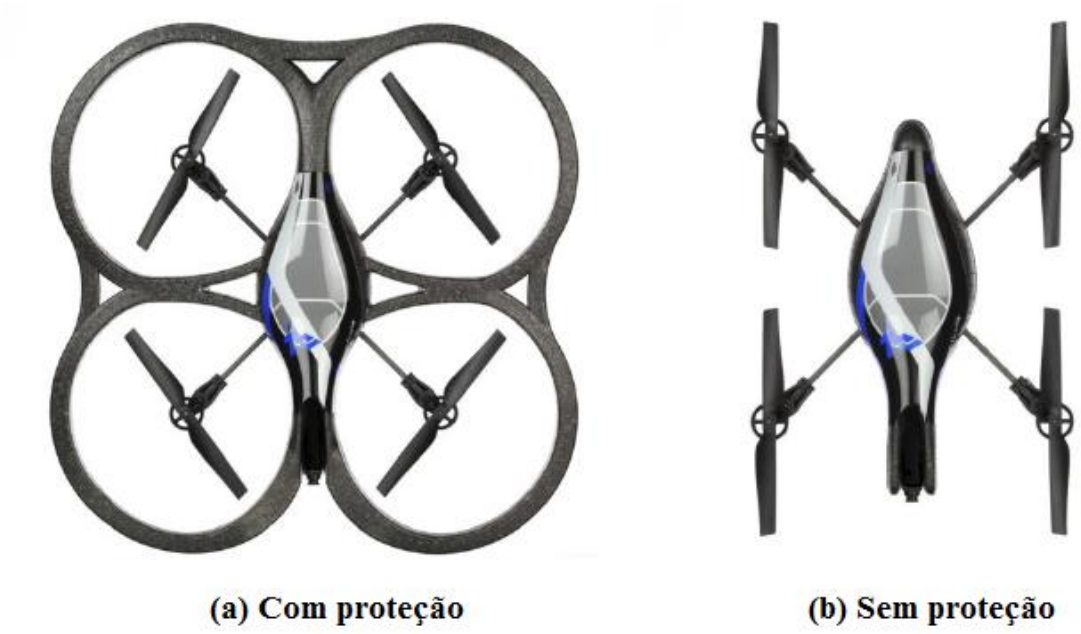

Figura 11 - Estrutura de proteção dos motores
Fonte: AR.Drone SDK Manual 1.8 - Adaptado

2.4 Software

O AR.Drone possui em sua unidade central de controle, uma versão da distribuição BusyBox do sistema operacional Linux, que gerencia o *software* de controle e estabilização da aeronave, o controle de manobras assistidas e a comunicação da rede sem fio.

Para desenvolvimento de *software* de outros fabricantes, a Parrot lançou uma API (Interface de Programação de Aplicativos, em inglês) de código fonte aberto que possibilitou a popularização do AR.Drone como plataforma de pesquisa e desenvolvimento na área acadêmica, além de ter ampliado seu emprego no mercado de entretenimento, com o desenvolvimento de jogos de realidade aumentada.

Com o lançamento da API de código aberto, foi disponibilizado também um kit de desenvolvimento de *software* (SDK, em inglês) que define os protocolos de comunicação, diretrizes de configuração e customização, *softwares* de exemplo

multi-plataformas e as bibliotecas do mecanismo de controle intuitivo de operação remota.

2.5 Mecânica dos movimentos

Um quadricóptero é um veículo sub-atuado (i.e., possuem mais graus de liberdade que entradas de controle) (Sono, 2008), onde quatro entradas controlam seis graus de liberdade. Diferente de um Helicóptero, onde o motor é de passo variável, o quadricóptero possui quatro motores com passo fixo, e através do controle de suas velocidades, são geradas as forças de propulsão desejadas (Minh & Ha, 2010).

Um veículo quadrirotor, conforme visto anteriormente na Seção 1.1, possui quatro rotores dispostos em uma estrutura fixa em forma de “X”. Sua dinâmica de movimento se dá através da mudança de velocidade dos rotores, que operam em pares que rodam em direções opostas, conforme Figura 12.

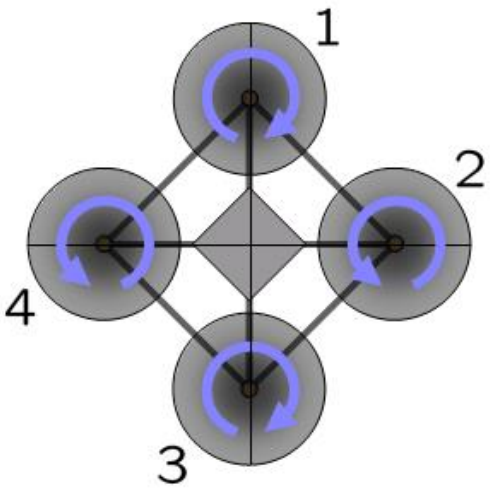

Figura 12 - Disposiçao dos motores
Fonte: http://en.wikipedia.org/wiki/Quadcopter - Adaptado

Com essa mudança de velocidade dos rotores, são produzidos quatro movimentos básicos, *Roll , Pitch, Yaw* e *Gaz* que são respectivamente, os ângulos de rotação (ângulos de Euler) no eixos x,y,z (Figura 13) da estrutura fixa, além do

movimento de ascenção vertical ou *Gaz*, que é um movimento ao longo do eixo z da estrutura fixa do veículo.

O movimento de *Roll*, ao aumentarmos as velocidades dos motores 1 ou 3 e mantermos constantes as velocidades dos motores 2 e 4, irá movimentar o quadricóptero para frente ou para trás, sendo assim uma rotação no eixo x do estrutura do veículo.

Já no *Pitch*, os motores 1 e 3 permanecem com suas velocidades constantes, enquanto os motores 2 ou 4 tem suas velocidades aumentadas, fazendo com que o quaricóptero se mova para esquerda ou direita, rotacionando-o no eixo y.

No movimento de *Yaw*, as velocidades dos motores 1 e 3 são aumentadas, com valores iguais para ambos os motores, enquanto as velocidades dos motores 2 e 4 são diminuídas, para que gire entorno de seu eixo z em um sentido, e ao diminuirmos a velocidade dos motores 1 e 3 e aumentarmos as velocidades dos motores 2 e 4, o quadricóptero gira no sentido contrário. Esses movimentos estão exemplificados na Figura 14.

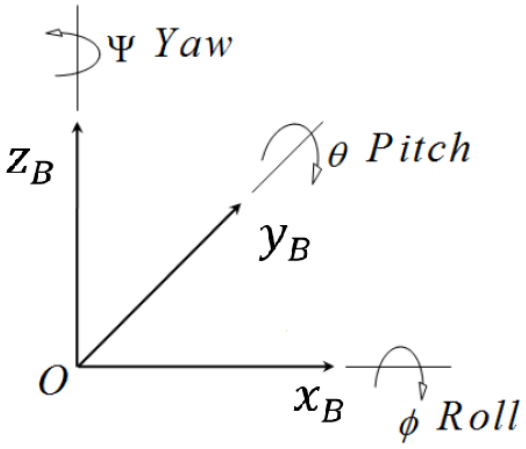

Figura 13 - Ângulos de Euler
Fonte: (Ahn, 2011) - Adaptado

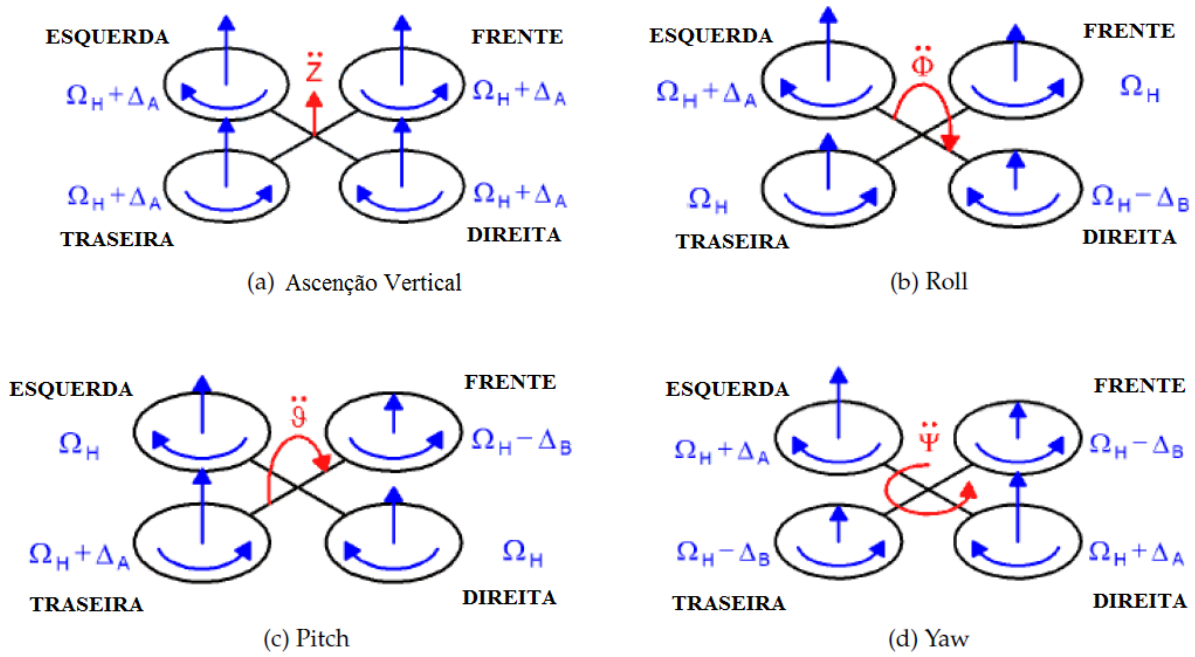

Figura 14 - Movimentos AR.Drone
Fonte: AR.Drone SDK Manual 1.8 - Adaptado

3 SISTEMA VICON

Esse capítulo apresenta o sistema responsável pela captura e processamento da imagem da servovisão. Seus componentes de *hardware*, principais características e aplicações serão aqui descritos.

O sistema Vicon é um sistema de captura de movimento avançado que proporciona maior precisão e desempenho nas áreas de pesquisa, entretenimento e ciência. Projetado para ser expansível e fácil de integrar em qualquer ambiente de trabalho. A combinação de seus componentes podem criar qualquer volume de captura.

A arquitetura de *hardware* é constituída pelo grupo de câmeras e a unidade de processamento, sendo interligados por cabos de rede de alta velocidade. A unidade de processamento é ligada a um microcomputador (*Host-PC*). No mesmo, há instalado o *software* do sistema Vicon que configura e cria os objetos a serem implementados no projeto. A Figura 15 ilustra um sistema com oito câmeras.

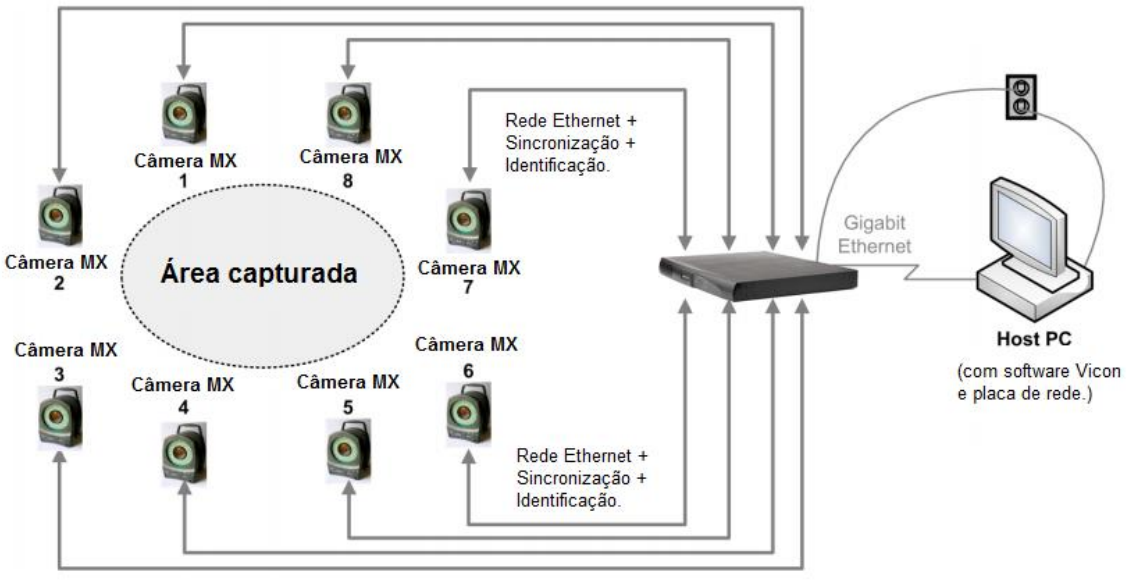

Figura 15 - Sistema com oito câmeras
Fonte: Vicon MX Hardware System Reference

3.1 Câmera Vicon

Cada câmera Vicon integra uma eletrônica interna que realiza a maioria do processamento de dados, uma unidade estroboscópica, uma lente de alta performance e filtro óptico. As câmeras geram pontos em tons de cinza dos marcadores refletidos no objeto capturado. Em seguida, usam algoritmos para encontrar o centróide e determinam qual dos seguintes objetos são susceptíveis de serem marcadores. Pode-se definir os tons de cinza que as câmeras irão gerar, configurando-os no *software* que está instalado no *host - PC*.

3.1.1 Tipos de Câmeras

O Sistema Vicon pode incluir três tipos diferentes de câmeras: Série F, MX e MX+, ver Figuras 16 e 17. Um único projeto pode ser constituído por vários tipos de câmeras, elas são completamente compatíveis.

Figura 16 - Câmera modelo MX+

Figura 17 - Câmera modelo MX

Fonte: Vicon MX Hardware System Reference

Todas as câmeras fornecem captura de movimento de alta velocidade, baixa latência e são equipadas com sensíveis sensores de estado sólido. As câmeras série F tem como sensores do fabricante Vegas, construído para captura de movimento. Os modelos MX+ e MX são equipados com sensores *CMOS*. Conforme Tabela 1 e Tabela 2, os modelos F40, MX-40, MX-40+ oferecem alta resolução. Em um projeto de baixo orçamento, no qual a área de cobertura é mais importante, são usados os modelos MX-3 e MX-3+ para ampliação do campo de captura.

Tabela 1 - Resolução Câmeras

Resolução	Série F	MX+	MX
4.0 Megapixels	MX-F40	MX-40+	MX-40
2.0 Megapixels	MX-F20	MX-20+	
1.3 Megapixels	-	MX-13+	MX-13
0.3 Megapixels	-	MX-3+	MX-3

Tabela 2 – Desempenho das câmeras

Desempenho	Resolução Megapixels 4	2	1.3	0.3
Resolução (pixels)	2352 x 1728	1600 x 1280	1280 x 1024	659 x 494
Tamanho do Sensor (mm)	16.5 x 12.1	11.20 x 8.96	15.4 x 12.3 h	6.5 x 4.9
Máximo *frame rate* (fps) em máxima resolução	Série F: 500 MX/MX+:160	Série F: 370 MX/MX+:219	- MX/MX+:482	- MX/MX+:242

Cada câmera é programada com *firmware* para controlar sua operação e realizar seu processamento de tons de cinza. Como todas as câmeras usam o mesmo *firmware*, a mistura de câmeras com diferentes sensores de imagem podem ser executadas em um mesmo projeto. A unidade de controle reconhece automaticamente as câmeras e suas características relevantes.

Como já citado, as câmeras Vicon processam a imagem inteira em tons de cinza, ao invés de aplicar um limiar de preto e branco. O processo com tons de cinza fornece mais informações e aumenta a precisão da medida em movimento. O reflexo do objeto gera manchas em tons de cinzas que, em seguida, usa algoritmos do tipo

centroidfitting para determinar quais destes objetos são realmente marcadores. Os dados gerados são enviados para a unidade de controle. O *software* Vicon instalado no *Host-PC* recebe esses dados da unidade de controle e os usa para visualização e processamentos específicos.

3.1.2 Lentes

A função da lente da câmera é coletar a luz refletida de um cenário e formar uma imagem focada no plano do sensor. O tipo de lente que irá proporcionar um bom desempenho em uma aplicação dependerá de vários fatores, como por exemplo: o campo de visão, a abertura da câmera e o filtro da lente.

3.1.3 Campo de Visão

O fator mais importante para determinar qual lente será usada na câmera Vicon é a área total que a câmera visualizará, chamado de campo de visão.

O cenário contido no campo de visão tem largura horizontal (H) e altura vertical (V). Sabendo que a distância (L) entre a câmera e um plano define qual será seu campo de visão. Com essa informação, podemos escolher o tipo de lente que atenderá nossa aplicação.

As lentes são descritas em termos do seu comprimento focal (f), definido como a distância em milímetros que o centro da lente está do sensor. O comprimento focal determina o ângulo de visão através da lente, como mostrado na Figura 18.

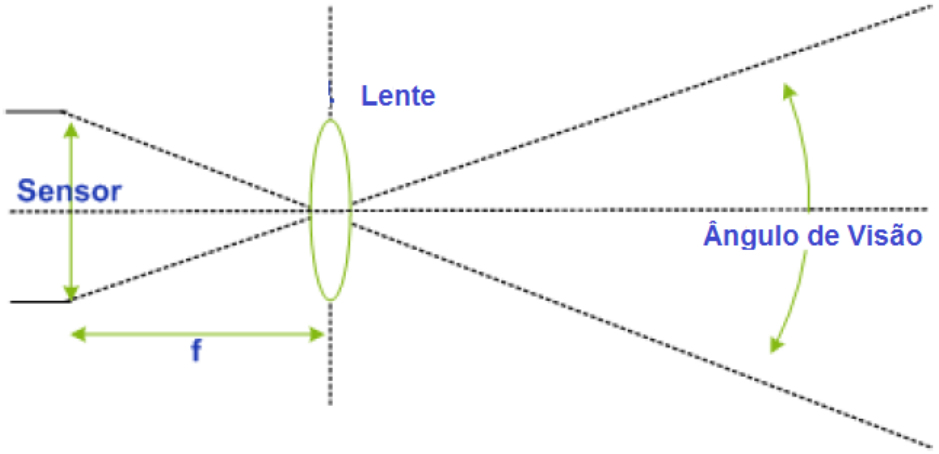

Figura 18 - Ângulo de visão
Fonte: Vicon MX Hardware System Reference

Normalmente, o ângulo de visão é calculado com base na distância focal conhecida e o tamanho do sensor. Isto, por sua vez, determina o campo de visão, baseado na distância (L) do objeto capturado. A Figura 19 ilustra como o campo de visão é calculado.

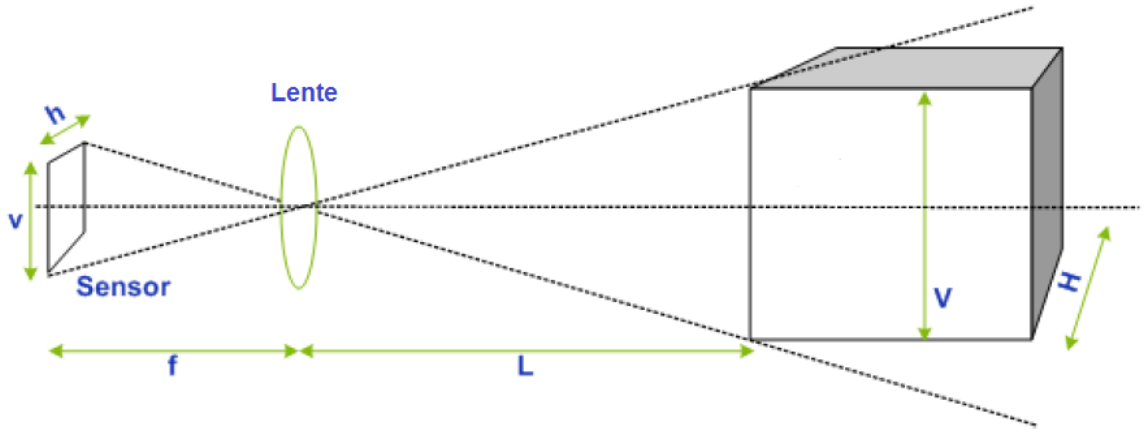

Figura 19 - Campo de visão
Fonte: Vicon MX Hardware System Reference

A área do sensor utilizada para a captura de dados através de cada câmera determina o campo de visão máximo disponível para uma combinação específica de câmera e lente. Conforme a Figura 20, a área do sensor é determinada pela sua largura horizontal (H) e a altura vertical (V).

A taxa de quadros de captura para cada câmera pode ser configurada no *software* Vicon Tracker 1.3 . A taxa de quadros configurados também afetará o campo de visão.

Maiores taxas de quadros reduzem a área do sensor que é usada para captura. Outra característica do sensor da câmera é o círculo de imagem. Este representa a imagem circular nítida que a lente da câmera lança sobre o sensor.

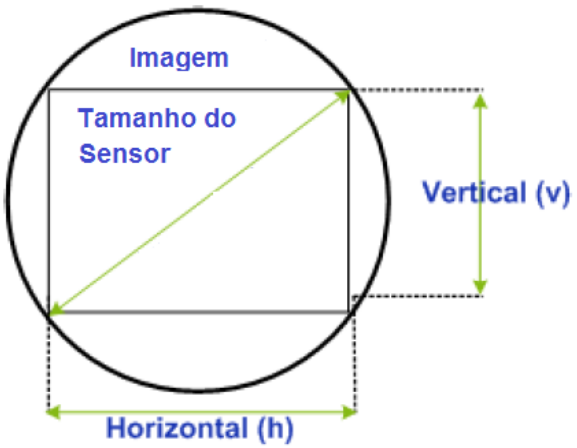

Figura 20 - Círculo de Imagem
Fonte: Vicon MX Hardware System Reference

O diâmetro do círculo de imagem é a área máxima sem perda de qualidade que a lente pode produzir. A maioria das lentes produzem sua melhor imagem no centro da lente, diminuindo o desempenho nos extremos da lente. Isso resulta num aumento gradual do preto em relação aos cantos da imagem.

Para um maior desempenho, o círculo projetado deve ser maior do que a área do sensor usado pela câmera. Isto assegura que todo o sensor seja utilizado e elimina a queda de luz que ocorre no sentido da borda da imagem. Se a área de

imagem de uma lente é muito pequena para o tamanho do sensor, as bordas e cantos da imagem são perdidos.

Se você deseja mudar a lente fornecida, deve-se considerar o tamanho do sensor da nova lente. Mesmo se a distância focal da nova lente for a mesma, um sensor de tamanho diferente produzirá um campo de visão diferente.

A seguir, a Tabela 3 com as fórmulas utilizadas para calcular o ângulo de visão e o campo de visão para combinações de câmera e lente. Estas fórmulas requerem as dimensões da área do sensor utilizado para cada uma das câmeras, onde:

f = Comprimento focal da lente (mm)
h = Largura horizontal do sensor (mm)
v = Altura vertical do sensor (mm)
L = Distância da lente para o objeto (m)

Tabela 3 - Calculo dos Ângulos e campos de visão

Ângulo de visão Horizontal (°)	Ângulo de visão Vertical (°)	Campo de Visão Horizontal Largura H (m)	Campo de Visão Vertical Altura V (m)
$2\ x\ \tan^{-1}\left(\frac{h}{2xf}\right)$	$2\ x\ \tan^{-1}\left(\frac{v}{2xf}\right)$	$h\ x\ \left(\frac{L}{f}\right)$	$v\ x\ \left(\frac{L}{f}\right)$

3.1.4 Abertura da câmera

Um outro fator importante para a escolha do tipo de lente é a sua abertura, ou seja, a velocidade com que a lente absorve luz num determinado momento. A abertura é a razão entre a distância focal da lente e o diâmetro de sua abertura, na qual é determinada a quantidade de luz que pode passar através da mesma.

A abertura afeta a profundidade do campo. Nesta área, a imagem permanece no foco. Fora deste regime, o foco torna-se progressivamente menos acentuado. Assim, com a diminuição da distância focal e da abertura, a profundidade aumenta. Por outro lado, se aumentarmos a abertura e o comprimento focal, a profundidade diminui.

3.1.5 Filtros de Lentes

Para otimizar o desempenho das câmeras, cada lente está equipada com um filtro óptico para atenuar os comprimentos de onda de luz diferente do que aqueles emitidos pelos diodos emissores de luz (LEDs). As câmera MX são equipadas com uma série de filtros passa-baixa, permitindo que só a luz de baixa frequência possa passar.

3.1.6 Unidade estroboscópica

A unidade estroboscópica consiste-se de LEDs que estão posicionados na frente das câmeras (Figura 21 e Figura 22) tem como objetivo iluminar o volume capturado. A unidade gera uma *flash* de luz brilhante, que ilumina os marcadores (revestidos com um material altamente refletido) presos no objeto que se deseja rastrear. O *flash* de luz coincide com o momento em que o obturador da câmera está aberta. A luz estroboscópia refletida passa através de um filtro óptico com uma resposta espectral. Ou seja, de tal forma que apenas a luz com mesmas características passe para lente. A lente recebe a luz e forma uma imagem focalizada dos marcadores na placa de sensores da câmera. A câmera converte o padrão de luz em dados que representam a posição e raio de cada marcador.

Figura 21 - Strobe MX+
Fonte: Vicon MX Hardware System Reference

Figura 22 - Strobe MX

3.2 Unidade de Processamento

A unidade de processamento, também chamada de Giganet (Figura 23), é responsável pela sincronização das câmeras e transmissão de dados, via cabo ethernet, para o microcomputador Host-PC. Todas as câmeras Vicon são conectadas, via cabos com conectores tipo lemo 10 pinos, na unidade Giganet (Figura 15).

Utilizando uma rede gigabit ethernet, dez vezes mais rápida que o normal, o sistema consegue uma pré-visualização melhorada, podendo ver uma imagem de vídeo *full frame* em uma taxa de quadros alta.

O Giganet possui um *switch* de rede ethernet de cinco portas para conexão com o Host-PC, outro cliente PC ou até mesmo dispositivos externos incluídos no projeto.

Figura 23 - Unidade central de processamento – Giganet
Fonte: http://www.vicon.com

4 INTEGRAÇÃO DOS SISTEMAS

Este capítulo abordará o a integração da plataforma AR.Drone e o sistema de câmeras Vicon. Descrevendo inicialmente os pacotes SDK do Ardrone e da Vicon.

4.1 SDK Ardrone

O pacote SDK AR.Drone oferece ao desenvolvedor bibliotecas em linguagem C para auxiliar na criação em seus projetos. Com o pacote, pode-se projetar em plataformas Windows, Linux, iOS ou Android. Para o projeto proposto, foi utilizado o sistema operacional Linux.

O pacote SDK inclui :

- um documento descrevendo o uso do pacote;
- biblioteca ArdroneLIB que fornece as APIs necessárias para a comunicação e configuração da rede Wi-Fi;
- biblioteca ArdroneTool auxilia o desenvolvedor a integrar seu próprio código aos já implantados no SDK.

4.1.1 Funções de controle AR.Drone

Conforme dito anteriormente, o SDK do AR.Drone fornece algumas funções de controle necessárias para construção de um programa em linguagem C. Funções estas que dentre outras coisas, permitem o controle dos movimentos do veículo, transmissão de dados de navegação (altura, ângulos do giroscópio, etc...), transmissão de vídeo e etc. Estas funções estão declaradas no arquivo *ardrone_api.h* do SDK.

Para este projeto foram usadas as seguintes funções:

ardrone_tool_set_ui_pad_start() – Responsável por ligar e desligar o AR.Drone. Quando a função recebe o valor “1” o veículo é ligado, e quando recebe “0” desliga-o.

ardrone_at_set_progress_cmd() – Responsável por mover o AR.Drone, composto por quatro movimentos básico, estes descritos anteriormente (Figura 14). É composta por cinco argumentos, um para ativar ou desativar a função, e os quatro restantes para controlar os movimentos(*Roll, Pitch, Yaw, Gaz*, ver item 2.4). Esses argumentos recebem valores entre -1,0 e 1,0 tabela , inclusive, para alterar o movimento do veículo, excetuando-se o primeiro argumento, que recebe somente “0” para desativar e “1” para ativar a função.

Tabela 4 - Argumentos da funçao de Controle AR.Drone

Argumento	**Valores**		
	-1	0	1
Phi	Máxima inclinação à esquerda	Nenhuma inclinação	Máxima inclinação à direita
Theta	Máxima inclinação para frente	Nenhuma inclinação	Máxima inclinação para trás
Gaz	Máxima velocidade vertical para baixo	Mantém altura padrão do modo planar	Máxima velocidade vertical para baixo
Yaw	Máxima velocidade angular sentido anti-horário	Nenhuma rotação	Máxima velocidade angular sentido horário

4.1.2 Comunicação AR.Drone

Segundo (SDK manual) o AR.Drone é controlado por qualquer dispositivo cliente que suporta o modo ad-hoc . O AR.Drone cria uma rede *Wi-Fi* com ESSID[3] (*Extended Service Set Identification*) nomeada de ardrone_xxx. O usuário conecta o dispositivo cliente na mesma rede. O servidor DHCP AR.Drone concede ao cliente um endereço IP.

O AR.Drone utiliza três portas principais para comunicação, UDP 5554, UDP 5555 e UDP 5556. O controle e configuração é feito pela porta UDP 5556. Os comandos devem ser enviados em um tempo regular, que geralmente configurado em 30 vezes por segundo.

[3] ESSID – Conjunto único de caracteres que identifica uma rede sem fio. Fonte: Wikipédia

As informações de posição, velocidade, rotação do motor e etc , são enviadas pela porta UDP 5554. Essas informações são chamados de dados de navegação ou NAVDATA. Esses mesmos dados também são enviados 30 vezes por segundo.

A porta 5555 é reservada para o transmissão de vídeo das câmeras frontal e inferior.

4.1.3 Arquitetura SDK AR.Drone

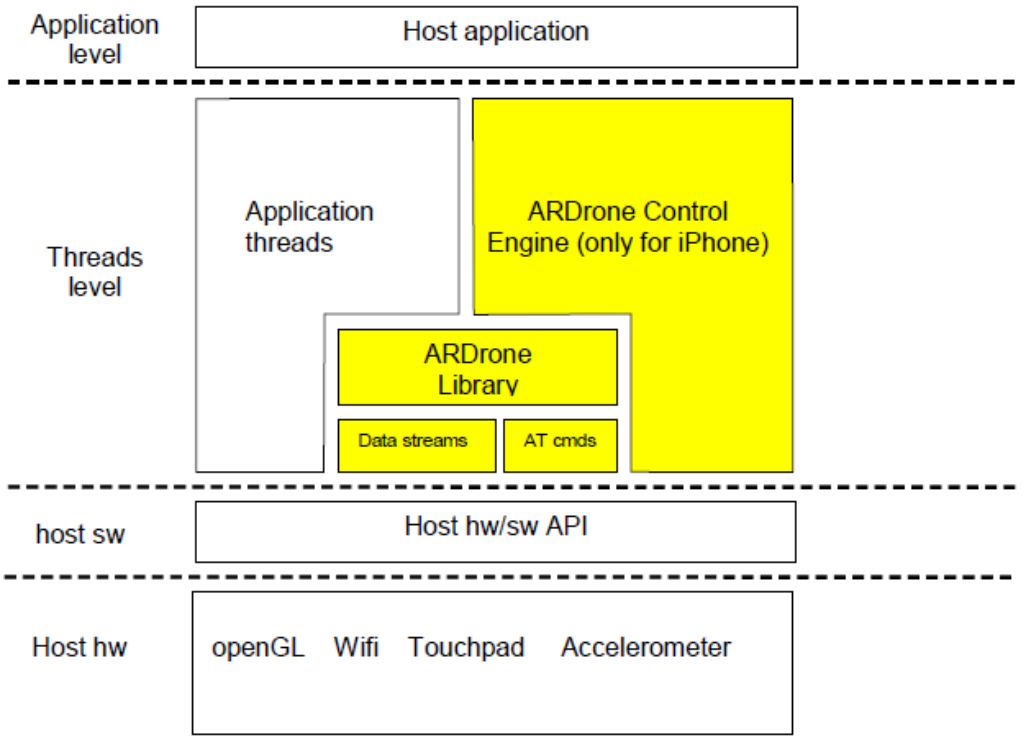

Figura 24 - Arquiterura em camadas
Fonte: AR.Drone SDK Manual 1.8

Este SDK provê meios para o desenvolvedor criar um conjunto de threads de execução paralela, especificar como os dados serão compartilhados entre as threads, declarar variáveis compartilhadas ou privadas e sincronizar as threads.

A thread é uma entidade de execução que é capaz de executar um fluxo de instrução independente e é escalonada pelo Sistema Operacional (CHAPMAN, et al., 2008).Se múltiplas threads colaboram para executar um programa, estas irão compartilhar o mesmo espaço de endereçamento do processo. Porém, cada thread possui seu próprio conjunto de registradores, apontador de instrução e contador de programa.

Existe, por padrão do pacote, uma thread para o vídeo, uma para as informações NAVDATA e uma para o controle. Executando assim, tarefas paralelamente.

Para criar um nova aplicação ou função com o SDK do AR.Drone, o manual indica criar uma nova *thread*.

4.2 Criando novo aplicativo no SDK AR.Drone

Ao instalar o SDK AR.Drone na plataforma Linux, seguimos para o diretório C:\\...\Ardrone_SDK_2_0\Examples\Linux\sdk_demo. Temos as pastas :

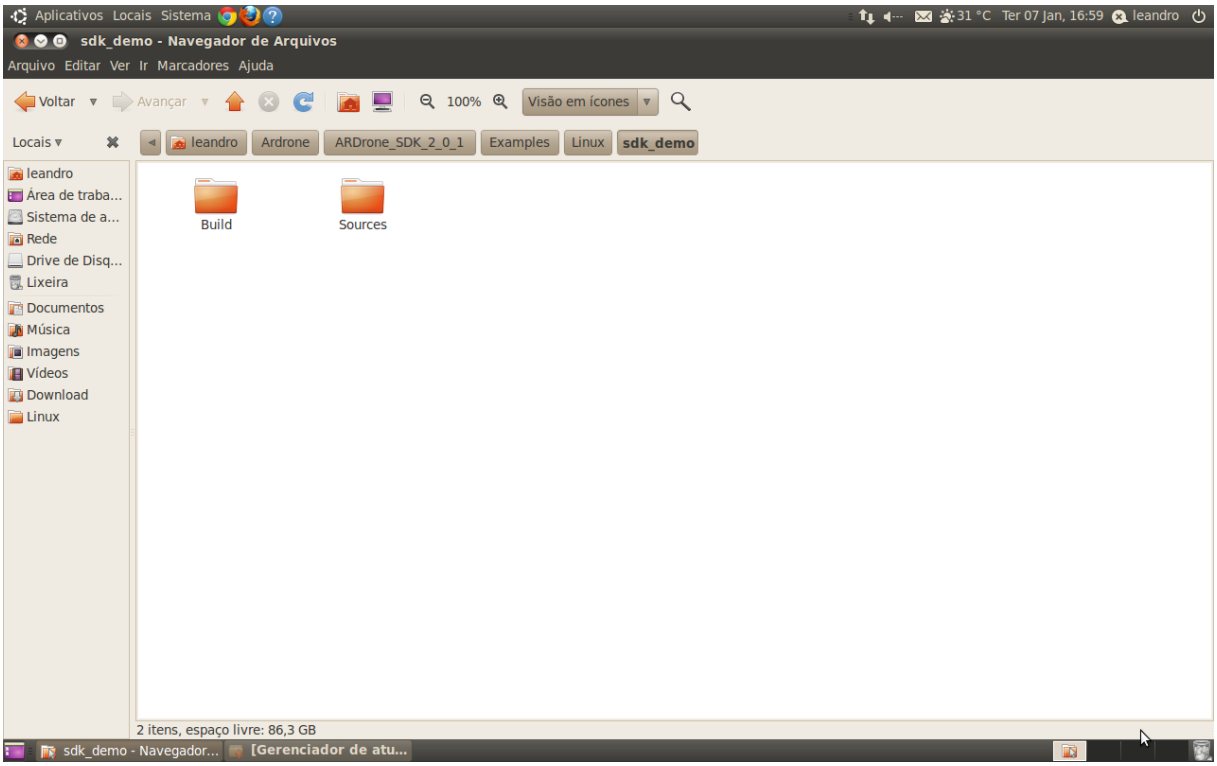

Figura 25 - Pasta origem do SDK AR.Drone

A pasta *Build* está o arquivo *Makefile*, onde o mesmo é o responsável pela compilação do nosso projeto. Na pasta Sources, existe as pastas: Navdata, UI e Vídeo. A pasta Navdata ficam os arquivos fontes que responsáveis por coletar os dados de navegação (nível da bateria, informações do altímetro e giroscópio). A pasta UI ficam os arquivos fontes de controle via teclado e *joystick*, e a pasta Vídeo estão os arquivos fontes de stream de video.

Para criar seu programa em outro arquivo fonte devemos criar um arquivo fonte chamado exemplo.c e um exemplo.h dentro do diretório *Sources*. Os arquivos completos estão no Apêndice E .

No arquivo exemplo.h devemos acrescentar as headers config.h e VP_Api\vp_api_thread_helper.h e criar uma thread com a seguinte linha:

exemplo.h

```
´´´
PROTO_THREAD_ROUTINE(exemplo, data);
```

No arquivo exemplo.c nos definimos a thread e dentro a rotina de aplicação. Seguindo como :

exemplo.c

```
...
DEFINE_THREAD_ROUTINE(exemplo, data)
{
```

```
//rotina
}

…
```

Modificamos o arquivo ardrone_testing_tool.c acrescentando a linha na seguinte sequência:

ardrone_testing_tool.c

```
…
  /* Start all threads of your application */
  START_THREAD( video_stage, NULL );
 START_THREAD( exemplo, NULL ); // inicializa a thread criada
...

…
  /* Relinquish all threads of your application */
  JOIN_THREAD( video_stage );
  JOIN_THREAD( exemplo );
…
…
/* Implementing thread table in which you add routines of your application and those provided by the SDK */
BEGIN_THREAD_TABLE
  THREAD_TABLE_ENTRY( ardrone_control, 20 )
  THREAD_TABLE_ENTRY( navdata_update, 20 )
  THREAD_TABLE_ENTRY( video_stage, 20 )
  THREAD_TABLE_ENTRY( exemplo, 20 ) // thread criada
END_THREAD_TABLE
…
```

Modificamos o arquivo Makefile no diretório C:\\...\Examples\Linux\sdk_demo\Build.

Makefile

```
…
# Define application source files
GENERIC_BINARIES_SOURCE_DIR:=$(SRC_DIR)

GENERIC_BINARIES_COMMON_SOURCE_FILES+=                    \
  UI/ui.c  \
  UI/gamepad.c \
  Navdata/navdata.c    \
  Video/video_stage.c \
  exemplo.c     # LINHA INCLUIDA
…
```

Compilamos o arquivo makefile e executamos o exemplo. O arquivo executável no diretório C:\\...\Examples\Linux\sdk_demo\Build.

4.3 SDK Vicon

O pacote SDK VICON oferece ao desenvolvedor bibliotecas em linguagem C++ ou MATLAB para auxiliar na criação em seus projetos. Com o pacote, pode-se projetar em plataformas Windows ou Linux. Para o projeto proposto, foi utilizado o sistema operacional Linux e linguagem C++.

O pacote SDK inclui :

- o documento descrevendo o uso do pacote;
- o instalador de 32 bits e 64 bits para Windows;
- o instalador para Linux;
- código do programa demostração em C++.

O SDK da Vicon fornece um programa demonstração em C++, onde nele está exemplificado o uso de todas as funções do SDK.

Foi utilizado esse programa como base do algoritmo de integração dos sistemas, o qual foi modificado para atender a demanda do projeto.

O Manual do SDK traz toda a informação necessária para utilização das funções, assim como instruções de instalação.

4.4 Integração dos Sistemas

Para integração com o sistema AR.Drone, foi necessário acrescentar uma placa de rede TP-Link Gigabit no microcomputador disponibilizado no laboratório de Controle e automação da UERJ. Uma rede foi criada para a transmissão de dados da unidade de processamento para o *Host-PC* e uma outra rede para a transmissão de dados do *Host-PC* para o *notebook* com o sistema AR.Drone. A Figura 26 apresenta a configuração feita na rede Vicon / *Host-PC*.

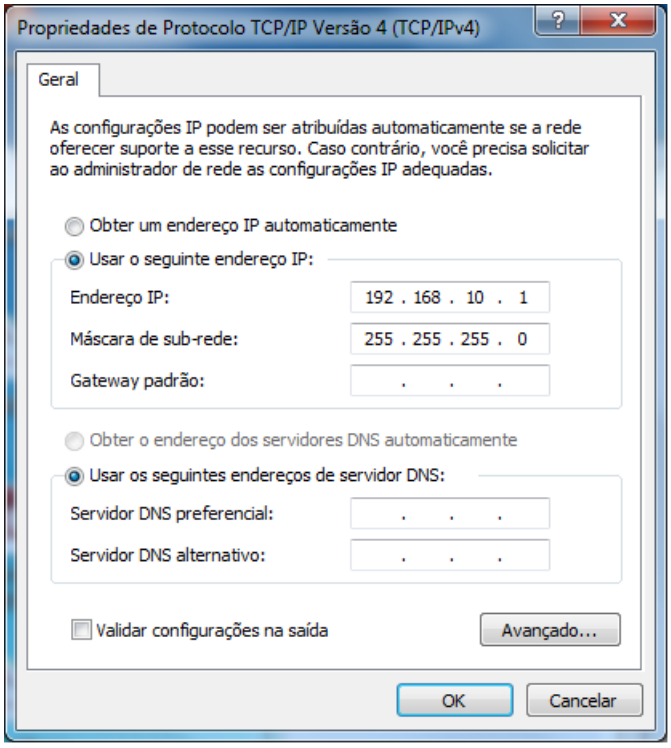

Figura 26 - Configuração de rede Vicon - Host PC

A rede ethernet que interliga o microcomputador *Host-PC* e o *notebook* com o aplicativo de controle do AR.Drone ficou com a configuração da Figura 27.

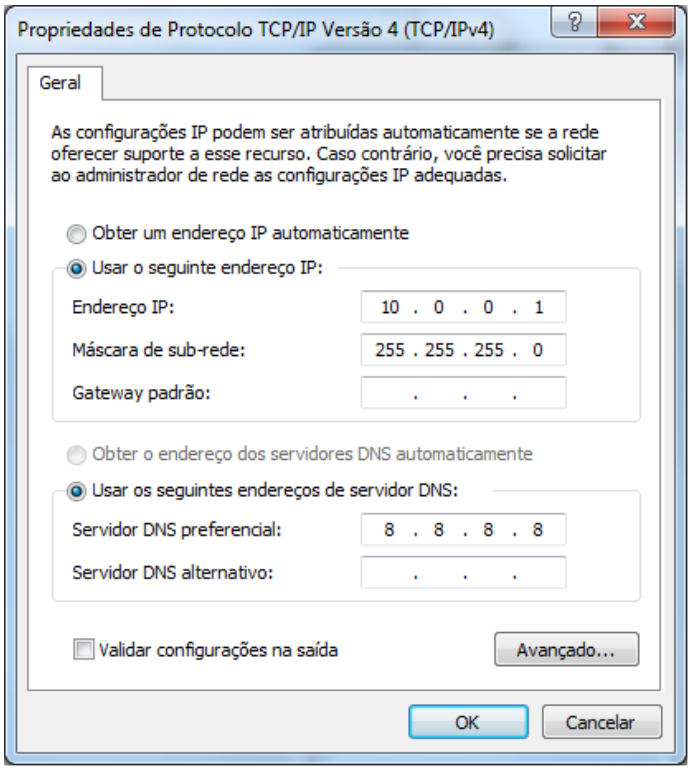

Figura 27 - Configuração de rede Host PC - Notebook

Como dito anteriormente, o AR.Drone cria uma rede Wi-Fi com ESSID nomeada de ardrone_xxx. Essa rede por padrão de configuração do AR.Drone disponibiliza ao notebook um IP na faixa 192.168.1.x.

Com o novo dispositivo de rede, o diagrama de blocos implementado com os dois sistema ficou conforme Figura 28. Todas as câmeras Vicon são conectadas, via cabos com conectores tipo lemo 10 pinos, na unidade Giganet. A unidade é ligada com cabo ethernet a porta de rede do *Host-PC*. O software Vicon Tracker 1.3 instalado no *Host-PC*, envia os dados processados para o *notebook*. A conexão entre eles é feito por cabo ethernet. O *notebook* executa o software criado neste projeto e envia o comando de controle para o AR.Drone via Wi-Fi.

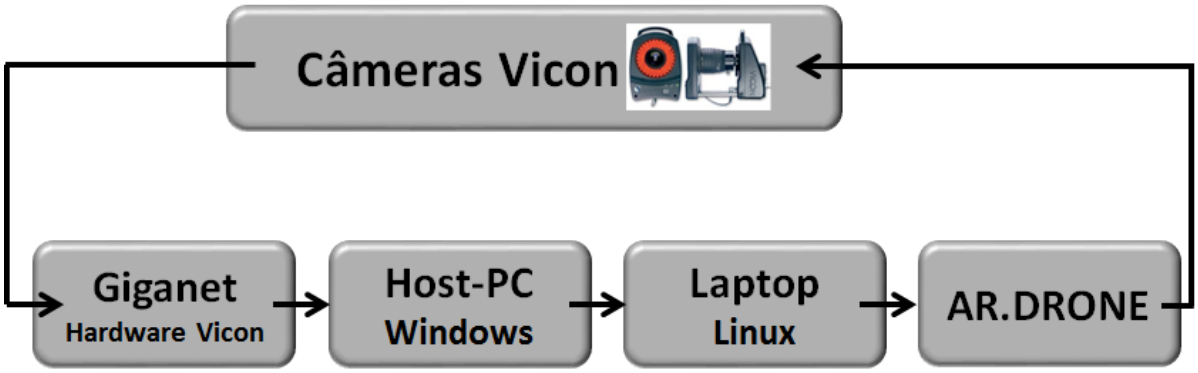

Figura 28 - Fluxo do sistema integrado

No *notebook* são instalados e configurados os pacotes SDK do AR.Drone e Vicon. Para criação do aplicativo unindo os dois pacotes foi implementado um artifício de programação chamado *extern "C"* que possibilita a usar funções ou threads escritas em linguagem C++ em um código desenvolvido em linguagem C ou vice-versa. O motivo desse feito, é devido as diferenças semânticas entre as linguagens. A maioria dos programas em linguagem C que não são triviais, não compilam em um código escrito em linguagem C++.

No algoritmo 1, é observado o algoritmo utilizado na integração dos dois sistemas.

Algoritmo 1 – Algoritmo de Integração

```
1: Início;
2:      Conectar AR.Drone via Wi-Fi;
3:      Conectar Vicon;
4:      Ligar AR.Drone;
5:      Enquanto ( Conexão == Verdade)
6:          Recebe quadro (frame);
7:          Recebe posição nos objetos selecionados no Vicon Tracker;
8:          Calcula controle;
9:          Envia controle;
10:     Fim Enquanto;
11:     Desconecta;
12:Fim;
```

5 MODELAGEM DINÂMICA DO AR.DRONE

Este capítulo apresenta a abordagem utilizada para obtenção da modelagem dinâmica do AR.Drone e onde se define os sistemas de coordenadas adotados no projeto.

A fim de se obter uma modelagem do quadricóptero, se faz necessária a especificação dos sistemas de coordenadas que serão utilizados, assim como a relação entre eles. A definição de diferentes sistemas de coordenadas é essencial para a identificação da localização e atitude do veículo em seis graus de liberdade (Morar & Nascu, 2013).

5.1 Sistemas de Coordenadas

Para este trabalho foram assumidos dois sistemas referenciais de coordenadas. O primeiro é inercial e fixo, e é também o referencial usado pelo sistema de captura Vicon System:

$$\boldsymbol{E} = \{\boldsymbol{x_E}, \boldsymbol{y_E}, \boldsymbol{z_E}\}, \quad (5.1)$$

que usaremos como sensor, conforme Figura 29. O segundo é um referencial móvel:

$$\boldsymbol{B} = \{\boldsymbol{x_B}, \boldsymbol{y_B}, \boldsymbol{z_B}\}, \quad (5.2)$$

com eixos fixados no centro de gravidade do veículo (Figura 29).

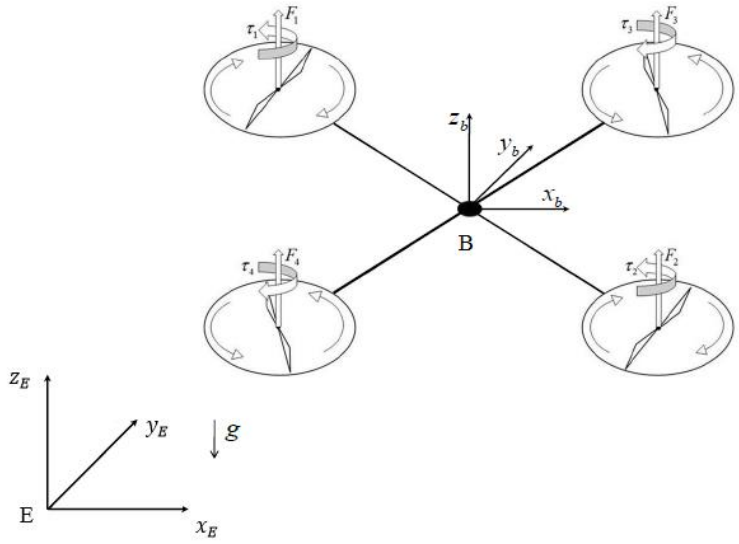

Figura 29 - Sistemas de coordenadas referenciais
Fonte: (Ahn, 2011) - Adaptado

Quando se deseja mapear a posição de um ponto de um sistema de coordenadas em relação à outro, utiliza-se uma matriz de rotação R. Definindo um vetor posição no sistema de coordenadas inercial $\boldsymbol{p}_E = \{\boldsymbol{x_E}, \boldsymbol{y_E}, \boldsymbol{z_E}\}$ e um vetor posição no sistema de coordenadas do veículo $\boldsymbol{p}_B = \{\boldsymbol{x_B}, \boldsymbol{y_B}, \boldsymbol{z_B}\}$, podemos obter a seguinte relação entre eles: $\boldsymbol{p}_B = R\boldsymbol{p}_E$, onde R é uma matriz de rotação nos eixos x,y,z (Siciliano, et al., 2010) :

$$R(\gamma,\alpha,\beta) = \begin{bmatrix} \cos\beta\cos\alpha & -\cos\gamma\,\mathrm{sen}\,\alpha & -\mathrm{sen}\,\beta \\ \cos\gamma\,\mathrm{sen}\,\alpha + \mathrm{sen}\,\gamma\,\mathrm{sen}\,\beta\cos\alpha & \cos\gamma\,\mathrm{sen}\,\alpha - \mathrm{sen}\,\gamma\,\mathrm{sen}\,\beta\,\mathrm{sen}\,\alpha & -\cos\beta\cos\alpha \\ \mathrm{sen}\,\gamma\,\mathrm{sen}\,\alpha - \cos\gamma\,\mathrm{sin}\,\beta\cos\alpha & \mathrm{sen}\,\gamma\cos\alpha + \cos\gamma\,\mathrm{sen}\,\beta\,\mathrm{sen}\,\alpha & \cos\gamma cos\beta \end{bmatrix}, \quad (5.3)$$

onde γ,α,β são respectivamente os ângulos entre os eixos x,y,z dos sistemas de coordenadas Inercial e do veículo. Vide exemplo da Figura 30.

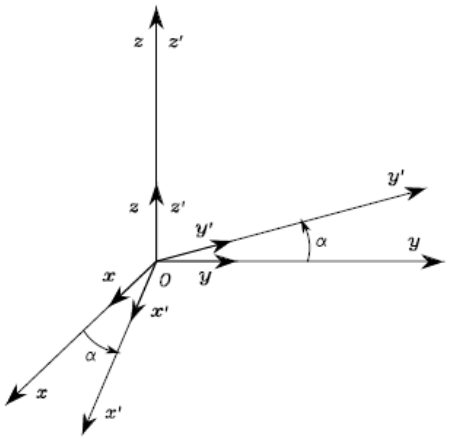

Figura 30 - Exemplo rotação em α sobre eixo z
Fonte: (Siciliano, et al., 2010)

5.2 Modelagem dinâmica

Um modelo matemático de um sistema dinâmico é definido como um conjunto de equações que representa a dinâmica de um dado sistema com precisão ou pelo menos, razoavelmente bem. O modelo matemático pode assumir diferentes formas, portanto dependendo do sistema considerado e das circunstâncias particulares, um modelo pode ser mais complexo que outros (Ogata, 2010).

Para se projetar um controle para veículos quadrirrotores, é necessário conhecer seu modelo dinâmico e seus parâmetros. Entretanto, ao invés de modelarmos o quadricóptero como um veículo quadrirrotor padrão, i.e., considerando as velocidades dos rotores como entradas e os ângulos de Euler como saída, foi usado um modelo considerando o controle interno embarcado do *drone* (Figura 31). Uma vez que este controle interno embarcado é capaz de definir e manter a orientação e velocidade vertical do veículo (Krajník, et al., 2011), não temos a que lidar com a complexidade do modelo matemático do quadricóptero (Beard, 2008).

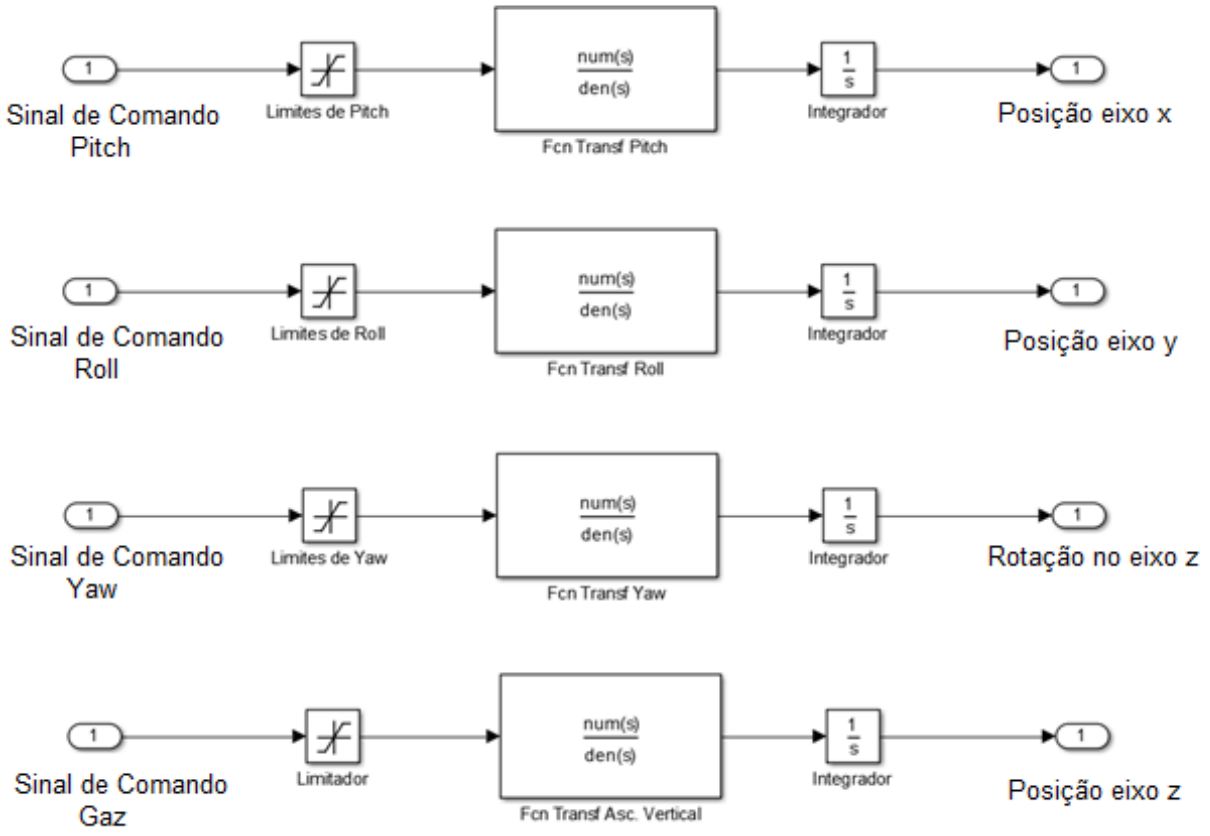

Figura 31 - Modelo Matemático de cada parâmetro de movimento AR.Drone
Fonte: (Krajník, et al., 2011) - Adaptado

Com intuito de obter-se um modelo que se aproxime da dinâmica de movimento do quadricóptero, foi usado a ferramenta computacional System Identification ToolBox (Figura 32) do MATLAB para identificar a função de transferência de cada uma das funções de controle (ver 4.1.1) do AR.Drone (*Roll, Pitch, Yaw* e *Gaz*). Para a obtenção desses parâmetros foi feito um código que aplicava um degrau em cada função de controle do quadricóptero, e com auxílio do sistema de captura Vicon, era medido a posição do veículo por intervalo de amostragem e os dados então guardados em um arquivo. Esse procedimento foi repetido diversas vezes para taxas de amostragem de 100, 200 e 300 Hz e para valor de entrada das funções de 0,01 (Figura 33). Foi escolhido esse valor de entrada pois como a área de captura é relativamente pequena (4 m²) em relação ao tamanho do veículo, valores superiores a esse fariam o veículo se mover rápido, o que reduziria a resolução das medidas.

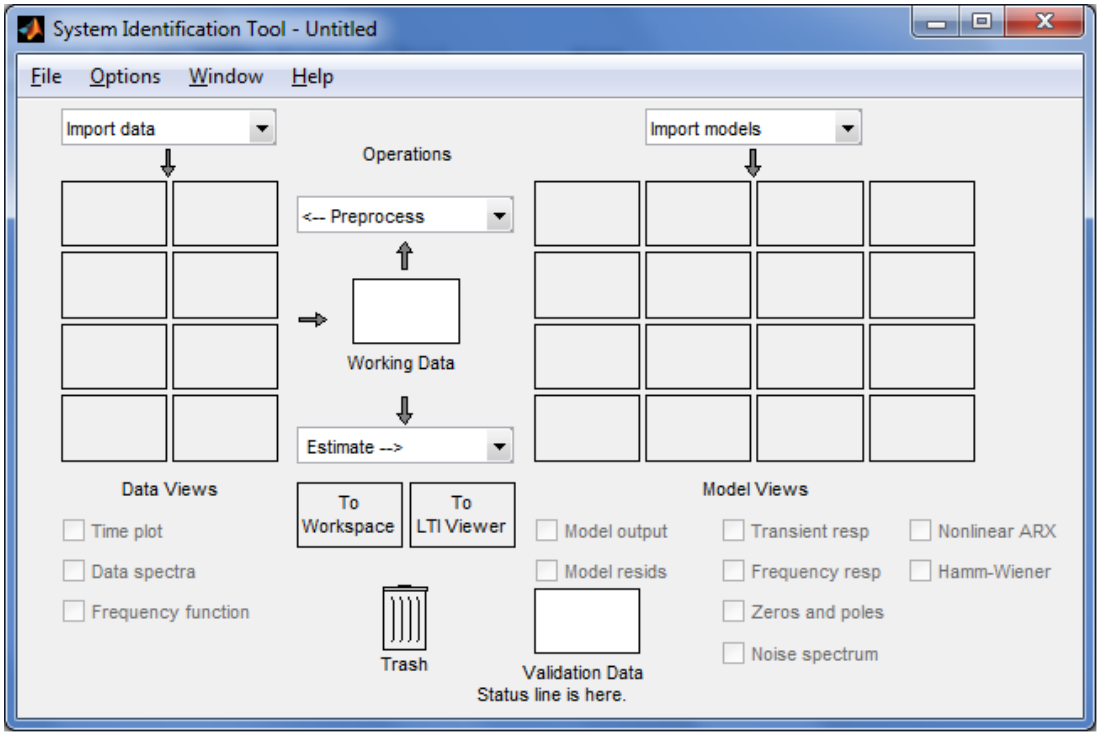

Figura 32 - Ferramenta System Identification Toolbox - MATLAB

Com os dados obtidos da resposta ao degrau do quadricóptero, utilizou-se iterativamente a ferramenta *System Identification Toolbox* para chegarmos a uma função de transferência que correspondesse ao modelo do veículo, ou seja, o gráfico obtido em simulação teria que ter um comportamento semelhante ao medido na prática para cada função de controle do AR.Drone.

Embora cada função de controle do AR.Drone deva apresentar um comportamento distinto, ou seja, cada parâmetro da função de controle teria que fornecer uma função de transferência particular, viu-se que essas apresentavam resultados similares quando utilizada a ferramenta *System Identification Toolbox*. Portanto decidiu-se adotar uma única função de transferência para todas as funções de controle.

Para a validação da função de transferência obtida:

$$\boldsymbol{H(s) = 2\frac{s+1}{s(s+0,05)}}, \qquad (5.4)$$

foi feito uma simulação através do *Simulink*, correspondente ao que foi feito com o AR.Drone, aplicando o mesmo valor de entrada e medindo a resposta à essa entrada.

Após seguidas tentativas, foi obtida a curva de resposta da função de transferência à entrada degrau 0,01 da (Figura 33), que teve uma assertividade de 94,91% com os valores obtidos do AR.Drone na prática. Para fazer essa comparação, foi usada a função *compare* do MATLAB, que compara dados de uma resposta ao degrau à uma função de transferência. O resultado obtido pode ser visto no gráfico da Figura 34.

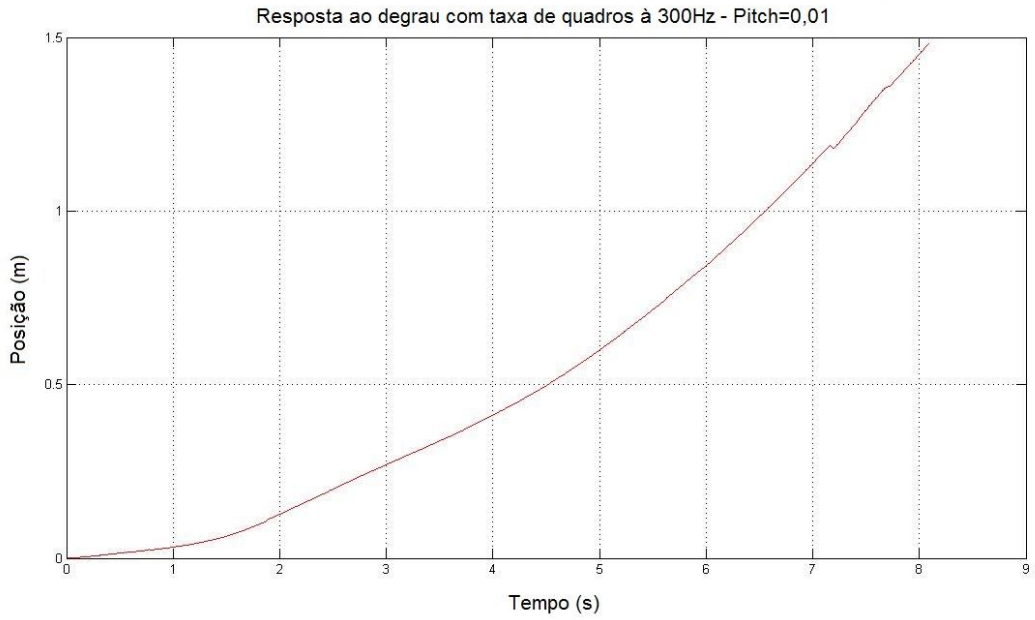

Figura 33 - Resposta ao degrau – Pitch

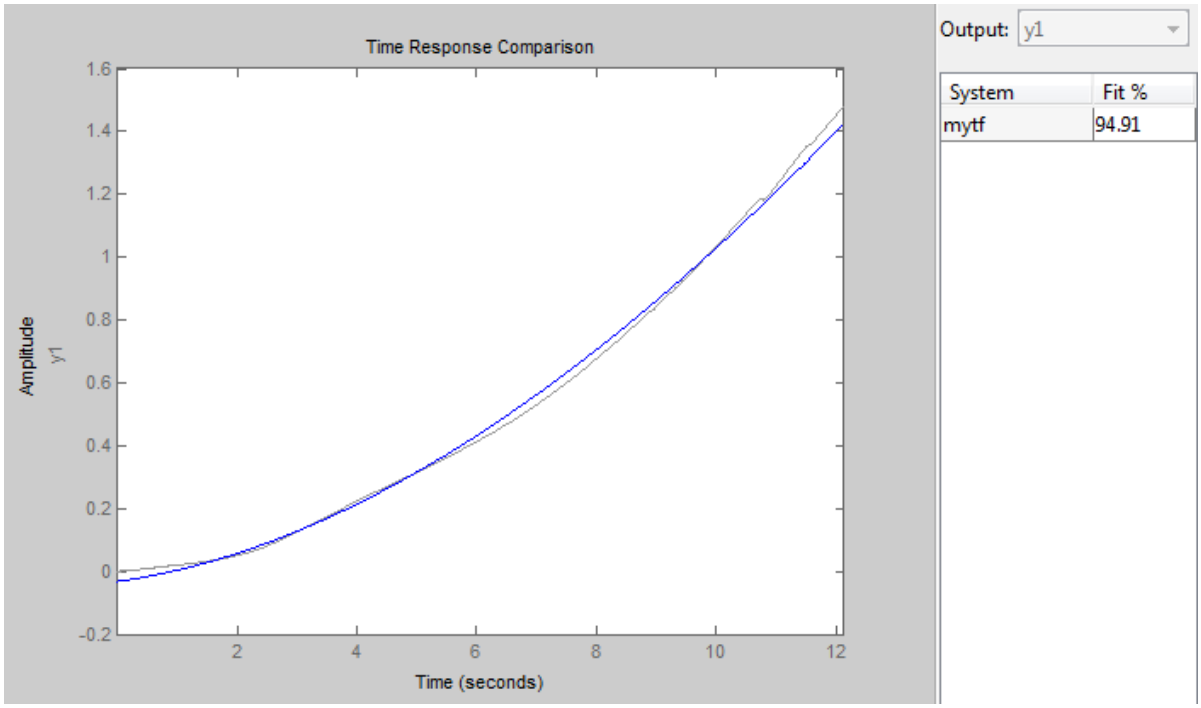

Figura 34 - Gráfico de ajuste entre o modelo gerado e a resposta ao degrau do AR.Drone

6 CONTROLE

Neste capítulo será descrito a técnica usada para o controle do veículo quadrirrotor, assim como explicações do algoritmo implementado.

6.1 Introdução

Uma vez que foi feita a modelagem dinâmica do AR.Drone, pode-se então começar a projetar um controle que se adeque ao modelo do veículo e o que se adapte melhor a necessidade da aplicação proposta. Para o projeto do controle, foi proposto o problema de rastreamento de um alvo móvel.

Avaliando o modelo dinâmico do quadricóptero, foi proposto um controle com ações Proporcional e Derivativa (PD), pois como a função de transferência do AR.Drone possui um integrador, um controlador do tipo PD atenderia os requisitos do problema proposto.

O controlador PD é proporcional a taxa de variação proporcional do sinal de erro atuante o que pode produzir uma correção significativa deste erro antes que o mesmo se torne demasiadamente grande. É portanto um controlador que antecipa o erro atuante e inicia uma ação corretiva mais cedo, para estabilizar o sistema. Possui como desvantagem à amplificação de sinais de ruído. Contudo, em nossa aplicação, como os sensores de medição de posição tem alta precisão, esse problema é minimizado (Ogata, 2010). Um controlador PD clássico em tempo contínuo é descrito pela equação:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt}, \qquad (6.1)$$

onde qual $u(t)$ é o sinal de controle e a constante de tempo T_d tempo derivativo. O sinal de erro

$$e(t) = y_{ref}(t) - y(t), \qquad (6.2)$$

representa a diferença entre o sinal de referência e a saída e os parâmetros de ajuste K_p e K_d são constantes.

6.2 Controle proposto

O sistema de captura Vicon, oferece uma amostragem máxima de 1000Hz. Entretanto, foi avaliado que com esta frequência de amostragem, não seria possível obter todos os quadros que o sistema enviaria para o computador onde seria desenvolvido o controle. Logo escolhemos uma frequência de 200Hz para realização do controle, pois assim temos fidelidade dos dados obtidos a cada quadro, sem a perda eventual de quadros o que ocorre a taxas mais altas.

O AR.Drone não nos permite controlar a velocidade de seus rotores, pois possui um controle de estabilidade embarcado. Entretanto, em sua biblioteca de funções, existe uma função que possibilita controlar a velocidade do veículo indiretamente, controlando apenas a angulação do mesmo em torno de seus eixos, tendo como resultado os movimentos descritos na Seção (2.5).

Como o sistema de captura amostra os dados obtidos, foi necessário pensar em um controle em tempo discreto, visto que assim tratamos o problema de uma forma mais realista. Então foi feita a discretização do controlador PD descrito pela equação (6.1) .

Seguindo a aproximação pelo método *Backward* Euler (Åström & Wittenmark, 2011), tem-se:

$$\frac{de(t)}{dt} \cong \frac{e(t) - e(t-h)}{h} \qquad (6.3)$$

Substituindo e $t = kh$ e $x(t)$ para $e(t)$,

$$\frac{de(t)}{dt} \cong \frac{e(kh) - e(kh-h)}{h}, \qquad (6.4)$$

onde h é o intervalo de tempo entre as amostras.

Então substituindo em (6.4)(6.1), temos:

$$u(k) = K_p e(kh) + K_d \frac{e(kh) - e(kh-h)}{h}, \qquad (6.5)$$

onde $k \in \mathbb{Z}^+$ representa o número da amostra.

6.3 Simulação do sistema de controle

Com o controlador proposto foram feitas simulações no MATLAB, em que embora o modelo dinâmico fora obtido em tempo contínuo, devido ao sistema de aquisição de posição amostrar os dados, o controle foi projetado em tempo discreto, conforme Figura 35.

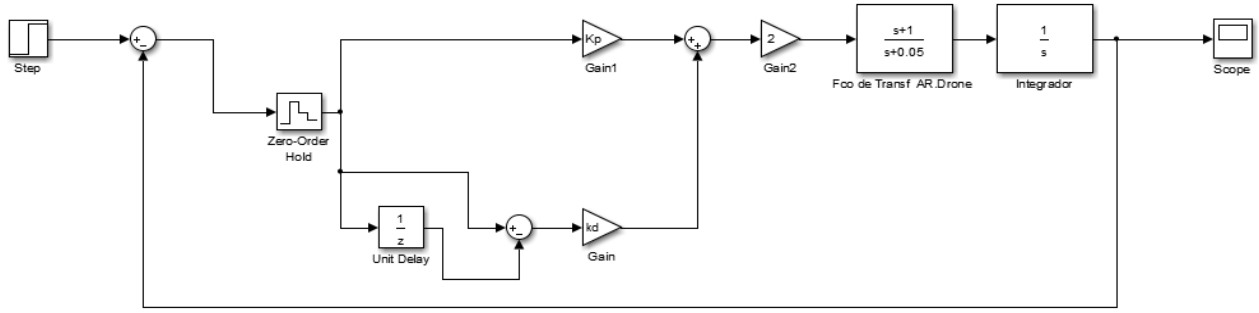

Figura 35 - Diagrama da implementação do sistema de controle na simulação

A partir de iterações, utilizando o método por aproximações sucessivas, foram obtidos os parâmetros K_p e K_d de maneira que o controle apresentasse uma resposta satisfatória, ou seja, foi proposto como critério mínimo aceitação de que a saída da resposta ao degrau apresentasse um sobrepasso (*overshoot*, em inglês) de no máximo 25%. Com esta definição, foi obtido a resposta do sistema controlado apresentado na Figura 36, com $K_p = 1$ e $K_d = 0{,}07$.

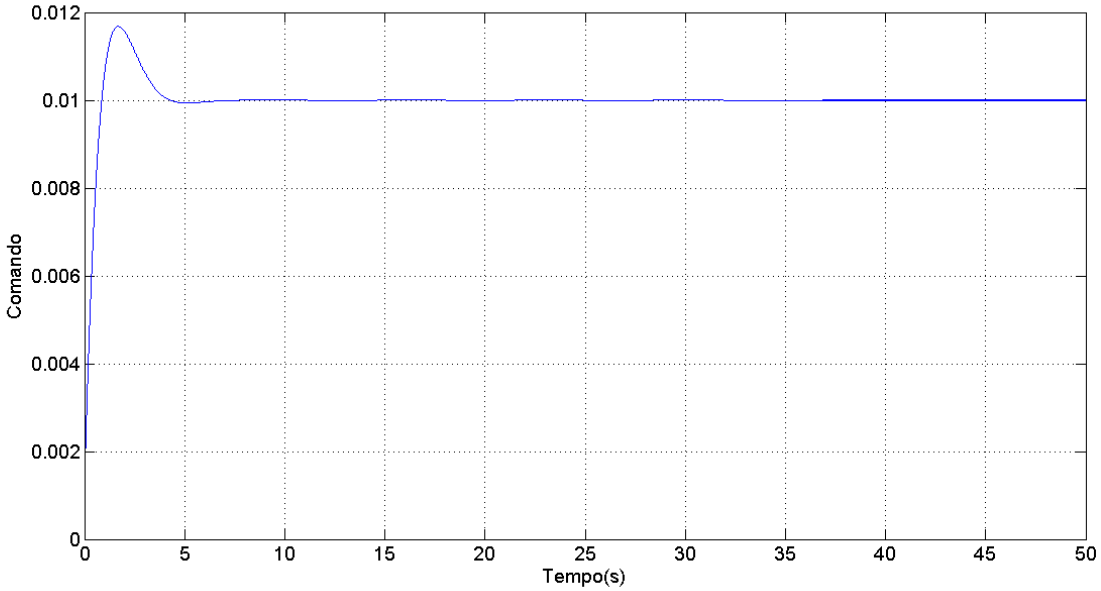

Figura 36 - Resposta ao degrau na simulação

6.4 Implementação do algoritmo de controle

Com a estratégia de controle definida, deve-se então elaborar-se um algoritmo em C++ que implemente no *notebook* o controle proposto anteriormente adequando-se ao problema de rastreamento de alvo.

No problema de rastreamento de alvo, o sinal de erro $e(t)$, é a distância entre o veículo e o alvo. Embora o quadricóptero possa operar no espaço tridimensional, neste projeto foi proposto um controle que atue no plano XY (referente ao sistema de coordenadas inercial) e sem o controle de orientação, onde além de ser mais simples a validação da estratégia de controle e a integração dos sistemas, a área de captura do sistema Vicon instalada no laboratório de Controle da UERJ, é limitada em 50 cm de altura no eixo de z do sistema inercial de coordenadas.

Conforme exemplificado na Seção 5.1, para mapearmos a posição do AR.Drone em relação ao seu sistema de coordenadas necessitamos multiplicar as posições obtidas pelo sistema Vicon, por uma matriz de rotação. Nesse caso, foi criado no algoritmo um vetor de orientação de modo que este é representado pelas posições (x_E, y_E) dos marcadores centrais conforme Figura 37. O intuito de se criar esse vetor é que assim pode-se calcular em cada amostra o ângulo de rumo, ou seja, o ângulo que o vetor orientação do veículo faz com o eixo y_E do sistema de

coordenadas inercial. Isso se faz necessário pois para calcularmos o erro, tanto a posição do alvo, quanto do veículo, devem estar mapeada em relação ao sistema de coordenadas fixo do veículo.

Para o cálculo do ângulo de rumo, definimos o vetor orientação como:

$$\boldsymbol{R_o} = \{x_b, y_b\} \tag{6.6}$$

E como o ângulo de rumo foi definido como o ângulo entre $\boldsymbol{R_o}$ e o vetor unitário do eixo do referencial fixo $\boldsymbol{y_e}$.

Então, temos o ângulo entre dois vetores (θ) definido como:

$$\theta = cos^{-1} \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{||\boldsymbol{A}||.||\boldsymbol{B}||} \tag{6.7}$$

Logo, temos que o ângulo de rumo é dado por:

$$A\_rumo = \cos^{-1} \frac{\boldsymbol{R_o} \cdot \boldsymbol{y_e}}{||\boldsymbol{R}_o||.||\boldsymbol{y_e}||} \tag{6.8}$$

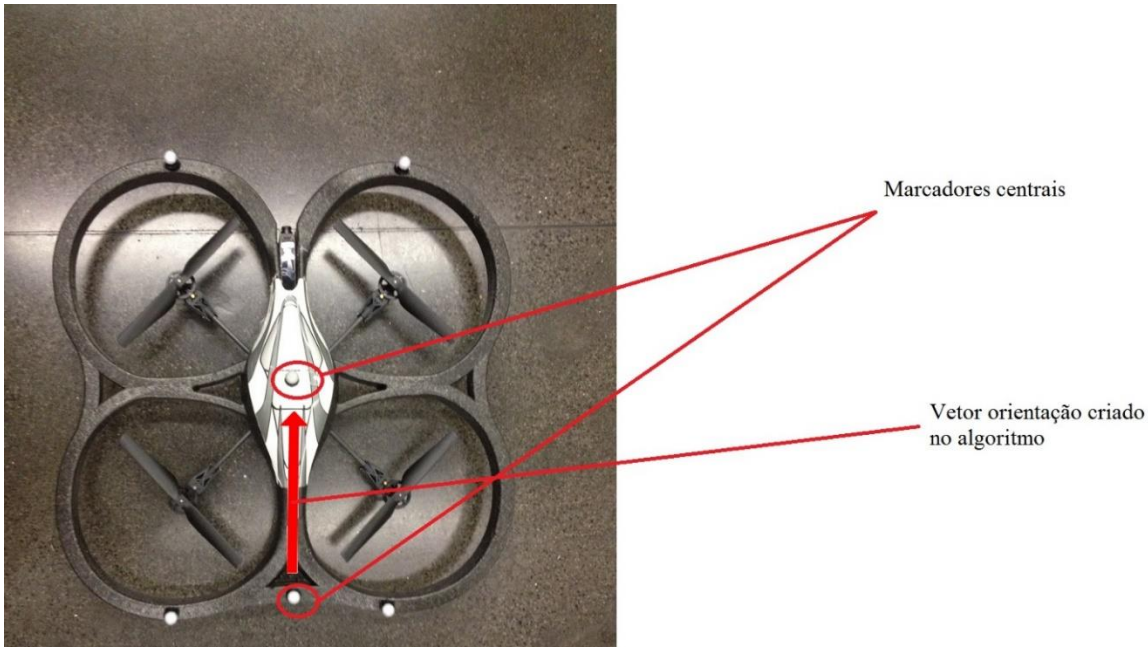

Figura 37 – Marcadores do vetor orientação

O algoritmo 2 exemplifica o tratamento dos dados lidos pelo sistema de aquisição de posição, assim como a implementação do controle PD proposto em 3.2.

Algoritmo 2 – Algoritmo de implementação do controle

...

1. Início
2. Lê a posição (x_E, y_E) do posições em x e y dos marcadores centrais;
3. Grava as posições e em um vetor bidimensional *R[2]* (Vetor Orientação);
4. Calcula-se a norma do vetor orientação *R[2]*;
5. Calcula-se o ângulo de rumo *a_rumo*;
6. Lê-se a posição em x e y do Alvo e grava em *x_wand* e *y_wand*;
7. Lê-se a posição em x e y do AR.Drone e grava *x_ardrone* e *y_ardrone*;
8. Cria-se vetores posição bidimensionais *pos_wand[2]* e *pos_ardrone[2]*, com as posições lidas anteriormente;
9. Multiplica-se os vetores posição por uma matriz de rotação referente ao angulo de Rumo;
10. Calcula-se o erro em x e y, já rotacionado para o sistema de coordenadas do AR.Drone;
11. Calculo dos termos proporcionais do controle *propx* e *propy*
12. Calculo dos termos derivativos do controle
13. Calculo do controle PD para cada eixo(x,y);
14. Computa-se o erro atual como o erro anterior para o quadro seguinte (próxima amostra);
15. Limita-se para garantir manobras suaves e limitar a velocidade máxima;
16. Envia-se para o AR.Drone a ação a ser tomada, de acordo com os valores calculados no controle;

7 TESTES E RESULTADOS EXPERIMENTAIS

Este capítulo apresenta os resultados e testes realizados do controle servovisual do AR.Drone utilizando o sistema de câmeras da Vicon.

A fim de validar o controlador proposto e comparar com os resultados simulados, foram realizados testes experimentais no Laboratório de Controle e Automação da Universidade do Estado do Rio de Janeiro (UERJ).

Nesta etapa foram definidas as posições iniciais do alvo e do AR.Drone, o tipo de alvo a ser seguido e como estes seriam coletados e tratados, para uma futura comparação dos resultados obtidos no Simulink/Matlab.

7.1 Ambiente de Teste

Para realização do experimento, foi definido um ambiente de teste, localizado no Laboratório de Controle e Automação da UERJ, local onde o sistema de câmeras está instalado (Figura 38).

Primeiramente foi demarcada a área de captura das câmeras do sistema Vicon no chão do laboratório para se ter uma dimensão da área útil disponível e também foram definidos os pontos de partida do AR.Drone e do alvo, que está exemplificado na Figura 39.

A instalação das câmeras foi concebida para projetos nos quais não fossem necessário o mapeamento no eixo z_E, pois para objetos que possam operar em alturas acima de 1,3 metros, o sistema não é capaz de captar com exatidão a posição do objeto.

Figura 38 - Laboratório de controle e automação

Figura 39 - Área de captura útil com marcações do Alvo e AR.Drone

7.2 Configuração do experimento

Para a realização do experimento, foram demarcadas as posição de partida do AR.Drone e do alvo, de modo que a distância entre eles seja máxima, considerando o marcador central de ambos como referencial na posição marcada na área de captura (Figura 39). Após isto foi configurado como alvo a ser seguido, um carro de controle remoto (Figura 40), e posicionados o AR.Drone e o alvo conforme a Figura 41.

No computador do laboratório, onde está instalado o *software* de tratamento de dados Vicon Tracker 1.3, foi criado o objeto virtual, onde nele são capturados as imagens dos marcadores e gerado um objeto com formato referente à disposição dos marcadores (Figura 42). O *software* então calcula automaticamente o centroide de cada objeto criado.

No *notebook* , onde está instalado o Linux, é executado o *software* de controle e também é gerado um arquivo no qual fica gravado o registro das posições do *drone* para geração de gráficos que serão usados para comparação com o modelo simulado.

Figura 40 - Alvo escolhido: Carro de controle remoto

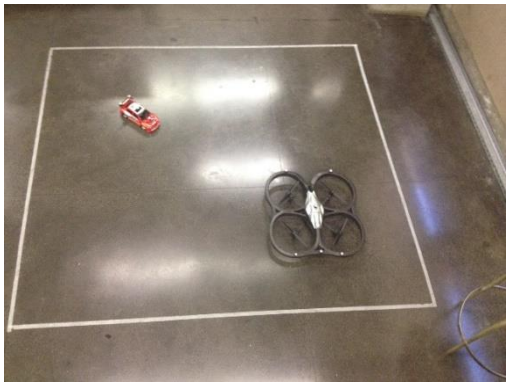

Figura 41 - Posicionamento do alvo e AR.Drone.

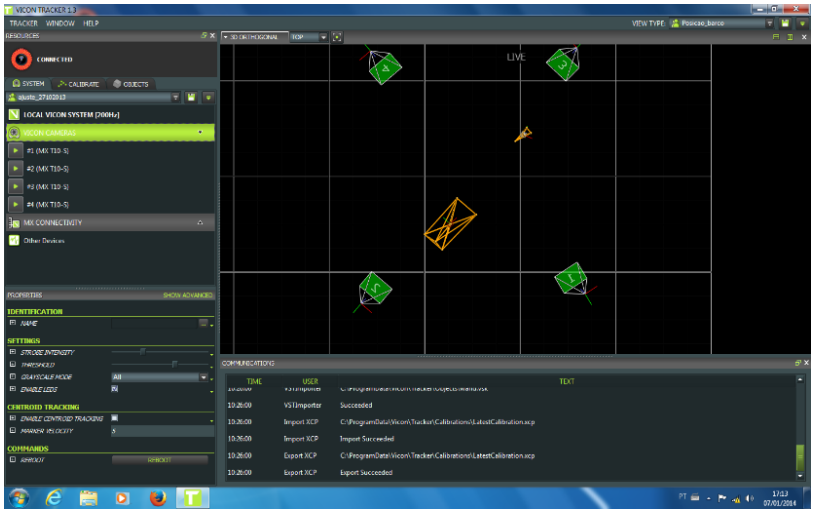

Figura 42 - AR.Drone e Alvo no ambiente virtual do software Vicon Tracker 1.3

7.3 Resultados Experimentais

Com a execução do programa criado, usando os parâmetros do controle simulado, foi constatado que os ganhos K_p e K_d necessitavam de alguns ajustes, pois o comportamento do veículo não estava exatamente de acordo com a simulação.

Após alguns ajustes, K_p e K_d receberam os valores de 0,6 e 0,2, respectivamente. Os gráficos da Figura 43 e Figura 44 apresentam o resultado da resposta ao degrau experimentalmente, com os novos valores de K_p e K_d.

.

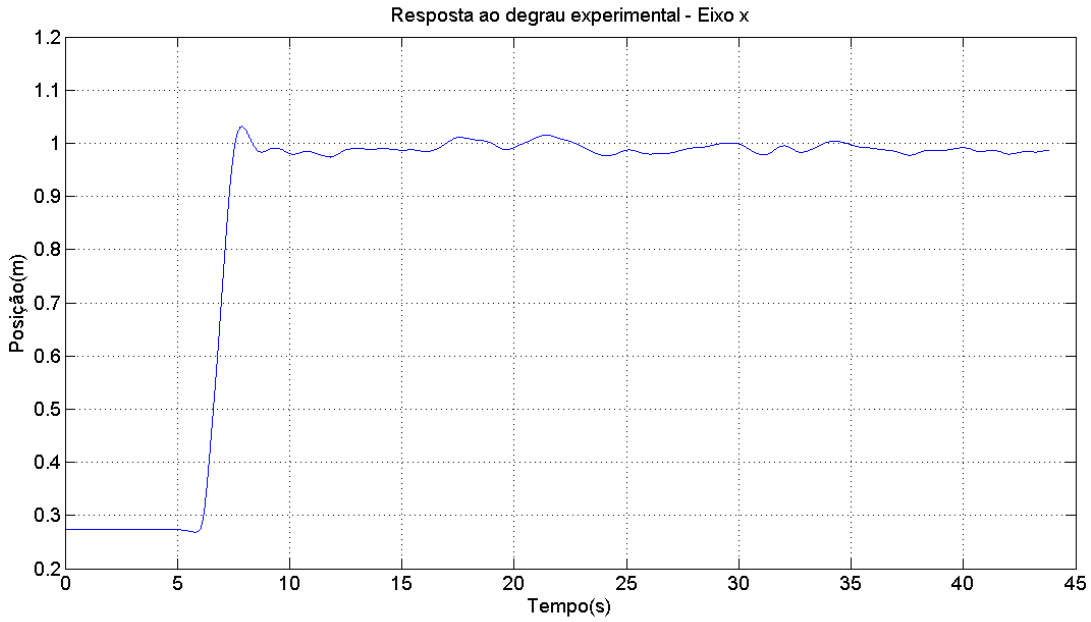

Figura 43 - Gráfico da resposta ao degrau em malha fechada referente ao eixo x

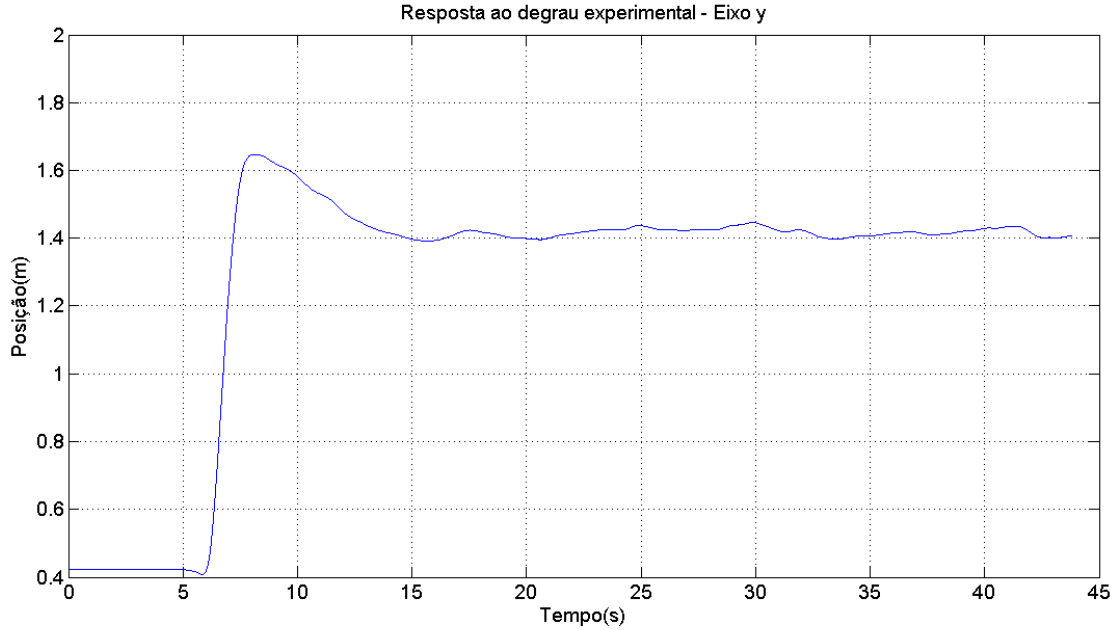

Figura 44 - Gráfico da resposta ao degrau de malha fechada referente ao eixo y

Os resultados obtidos experimentalmente foram comparados com os simulados e são apresentados nos gráficos da Figura 45 e Figura 46

.

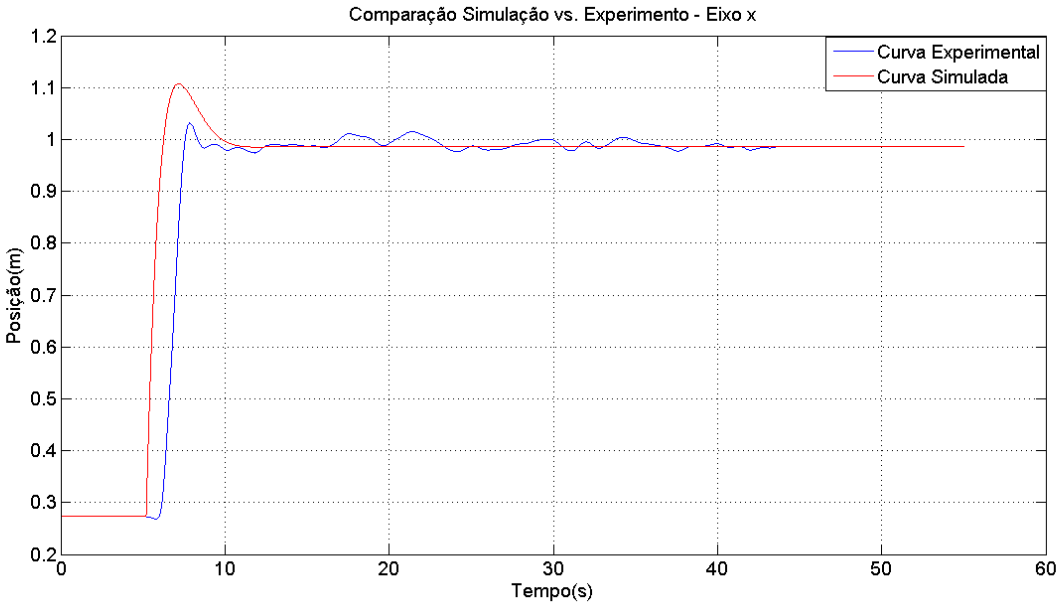

Figura 45 - Comparação entre a simulação e o experimento referente ao eixo x

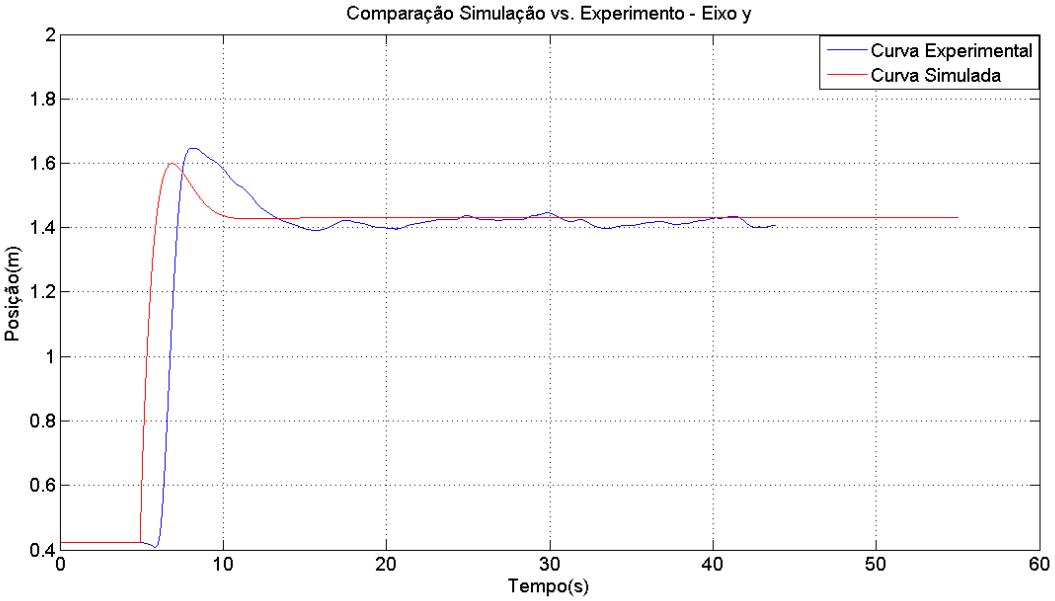

Figura 46 - Comparação entre a simulação e o experimento referente ao eixo y

Pode-se verificar que o sistema embora possua todos os polos e zeros no semi-plano complexo esquerdo, existe um transitório em que a resposta do sistema se comporta como um sistema de fase não-mínima. Isto pode ser relacionado ao

sistema de controle para a estabilização embarcado, e também à dinâmica de decolagem do veículo. Também verifica-se um atraso de cinco segundos, em média, para a partida do veículo. Isso se deve aos atrasos gerados pelos encadeamentos de rede (Figura 28).

No gráfico das trajetórias no plano XY, pode-se concluir que quando o alvo se move com uma velocidade baixa (Figura 47) o AR.Drone tem uma performance melhor ao segui-lo. Ao aumentarmos a velocidade do alvo, este rendimento cai, porém ainda é satisfatório, pois verifica-se no gráfico da Figura 48, uma variação de no máximo 0,15m entre as posições do AR.Drone e do Alvo.

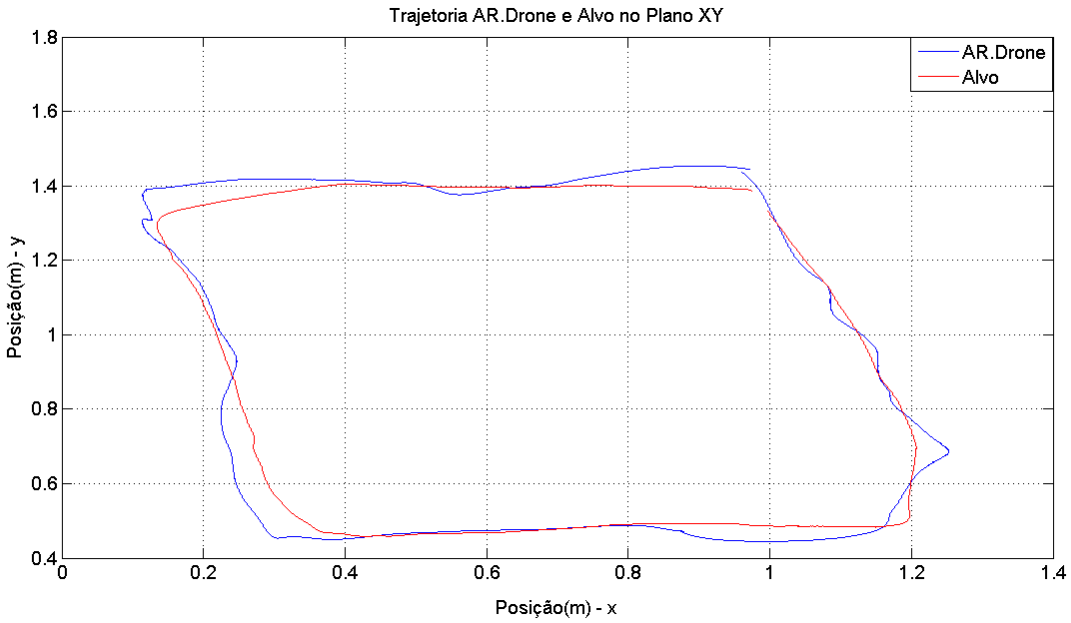

Figura 47 - Gráfico da trajetória no plano XY com velocidade baixa do alvo

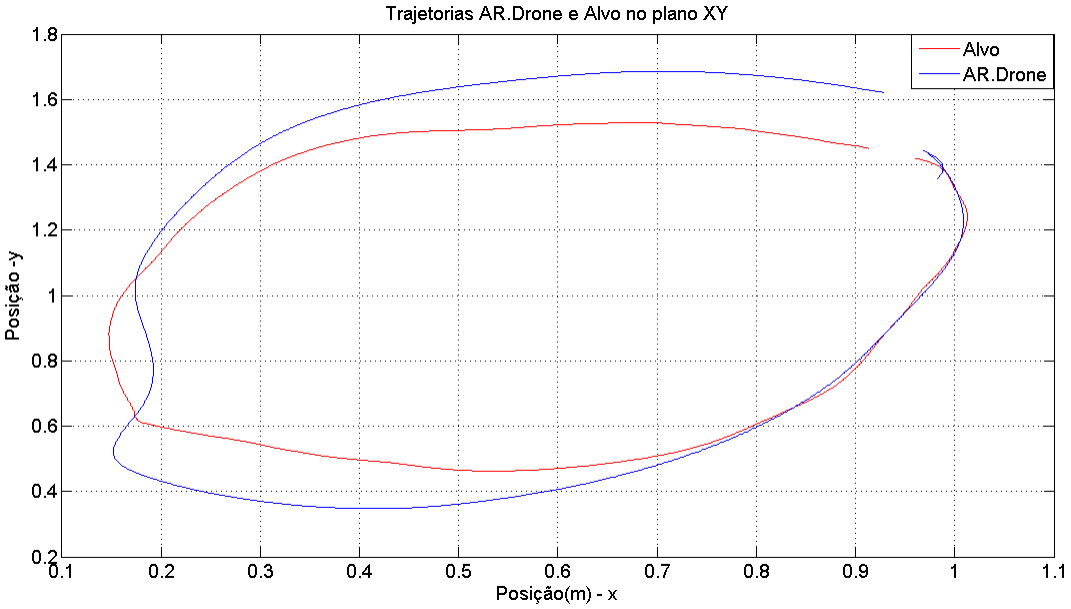

Figura 48 - Gráfico da trajetória no plano XY com velocidade moderada do alvo

8 CONCLUSÕES

Neste capítulo será apresentado uma síntese das conclusões obtidas no desenvolvimento do trabalho juntamente com indicações para possibilidades para continuação e aperfeiçoamento do projeto.

8.1 Considerações Finais

Durante o desenvolvimento do trabalho, pode-se observar a eficácia da estratégia de controle PD na resolução do problema do proposto.

Pode-se verificar a dificuldade em se obter, com exatidão, um modelo matemático do veículo, levando-se em consideração o seu sistema de controle embarcado, que embora facilite sua estabilização e operação do mesmo, este controle interno acaba introduzindo não linearidades ao sistema (Quadricóptero AR.Drone mais seu sistema de controle embarcado). Apesar disso, utilizando-se da ferramenta System identification toolbox do MATLAB, foi obtido um modelo razoável que não leva em consideração as não-linearidades do controle interno, e possui 94% de correspondência com o comportamento real do veículo conforme visto na Seção 5.2.

Nos testes preliminares do *software* criado, foi verificado que com o sistema Vicon, operando na sua taxa de quadros padrão de 1000Hz, o *notebook* onde o *software* é executado, não era capaz de processar as informações provenientes de cada quadro. Portanto houve a necessidade de diminuir a taxa de quadros para 200Hz.

No Capítulo 4, que aborda a integração os sistemas, foi visto que era preciso usar um artifício (***extern "c"***) para permitir a compilação do *software* de integração, pois o SDK do AR.Drone embora seja escrito em linguagem C, apresenta uma grande complexidade e um encadeamento de *makefiles*, o que gera uma incompatibilidade com o compilador *GCC*, quando este compila arquivos de bibliotecas mistas (C e C++).

O controle proposto apresentou resultados satisfatórios pois embora o modelo matemático utilizado não prever algumas não-linearidades e transitórios devido a decolagem e outros aspectos externos (efeito instabilizante da

proximidades com paredes, etc), o controle manteve a resposta do sistema de acordo com o proposto teoricamente.

Espera-se uma larga utilização deste trabalho em futuras aplicações, pois tanto o Sistema Vicon, quanto o AR.Drone, possuem um grande potencial para o desenvolvimento de diversos trabalhos em múltiplas áreas de conhecimento, tais como identificação, controle cooperativo e controle de formação de sistemas multiagentes.

8.2 Trabalhos Futuros

A partir das aplicações desenvolvidas neste trabalho, abre-se precedentes para criações de projetos utilizando-se dos sistemas aqui envolvidos em áreas como robótica, controle, computação e afins

Um aumento da área de captura pelas câmeras do sistema Vicon, se faz necessária para aplicações onde se queira usar mais de um veículo, pois essa área de captura pequena foi um fator limitante neste projeto.

Como proposta para outros projetos, sugere-se obtenção de um modelo mais real do veículo, o que pode beneficiar no desenvolvimento de sistemas de controle que sejam mais precisos. A proposta de algoritmos de controle robustos ou com algum tipo de adaptação são também de grande interesse.

Foi verificado que o Altímetro do AR.Drone é sensível a perturbações quando o alvo se move rapidamente embaixo do veículo. Em certas situações o veículo tende a alterar a sua altura padrão (1m) rapidamente, mas em seguida retorna ao seu valor de repouso sem grandes implicações ao algoritmo de controle.

Outra melhoria seria o desenvolvimento de uma estratégia de controle que utiliza-se todas as funções de controle do AR.Drone, pois assim seria possível uma melhor desempenho de manobrabilidade e realização de tarefa mais complexas. Porém esta utilização só será possível com um aumento da área de captura das câmeras.

REFERÊNCIAS

Ahn, Y. M., 2011. *AUTONOMOUS NAVIGATION AND LOCALIZATION OF A QUADROTOR,* Urbana-Champaign: s.n.

Anon., s.d. *http://ardrone2.parrot.com/.* [Online] [Acesso em 20 10 2013].

Augugliaro, F., Schoellig, A. P. & D'Andrea, R., 2012. *Generation of collision-free trajectories for a quadrocopter fleet:,* Zürich: s.n.

Beard, R. W., 2008. *Quadrotor Dynamics and Control,* Provo,Utah: s.n.

CHAPMAN, B., JOST, G. & PAS, R. V. D., 2008. *Shared memory parallel programming.* England: MIT Press.

Darma, S. et al., 2013. *Visual Servoing Quadrotor Control in Autonomous Target Search.* Shah Alam, Malaysia, s.n., pp. 319 - 324.

Eresen, A., ImamoĞlu, N. & Efe, M. Ö., 2012. Autonomous quadrotor flight with vision-based obstacle avoidance. *Expert Systems with Applications,* Janeiro, 39(1), p. 894–905.

Gomez-Balderas, J., Castillho, P., Guerrero, J. & Lozano, R., 2012. Vision Based Tracking for a Quadrotor. *Journal of Intelligent & Robotic Systems,* Volume 65.

Haulman, D. L., 1991-2003. *U.S Unmanned Aerial Vehicle in Combat.* s.l.:Air Force Historical Research Agency.

Hutchinson, S., Hager, G. D. & Corke, P. I., 1996. A Tutorial on Visual Servo Control. *IEEE Transactions on Robotics and Automation,* October, 12(5), pp. 651-671.

Krajník, T., VojtĚch, V., Fisěr, D. & Faigl, J., 2011. *AR-Drone as a Platform for Robotic Research,* Praga, República Tcheca: s.n.

Marchand, É., Chaumette, F., Spindler, F. & Perrier, M., 2011. *Controlling an uninstrumented ROV manipulator by visual servoing.* Honolulu, Havaí, s.n.

Mellinger, D., Michael, N. & Kumar, V., 2010. *Trajectory Generation and Control for Precise,* Philadélfia: s.n.

Michael, N., Fink, J. & Kumar, V., 2011. *Cooperative Manipulation and Transportation with Aerial Robots,* Filadélfia: s.n.

Minh, L. D. & Ha, C., 2010. *Modeling and Control of Quadrotor MAV using Vision-based Measurement.* s.l., s.n., p. 6.

Morar, I.-R. & Nascu, I., 2013. *Model Simplification of an Unmanned Aerial,* Cluj-Napoca, Romenia: s.n.

Müller, M., Lupashin, S. & D'Andrea, R., 2011. *Quadrocopter Ball Juggling.* São Francisco, s.n., pp. 5113-5120.

Ogata, K., 2010. *Engenharia de Controle Moderno.* 5 ed. São Paulo: Pearson Prentice Hall.

Podhradsky, M., 2012. *Visual Servoing for a Quadcopter Flight,* Praga: s.n.

Siciliano, B., Sciavicco, L., Villani, L. & Oriolo, G., 2010. *Robotics - Modelling,Planning and Control.* s.l.:Springer.

Sono, T. S. P., 2008. *PROJETO DE UM SISTEMA DE CONTROLE SUB-ATUADO PARA UMA PRÓTESE DE MÃO,* Rio de Janeiro: s.n.

Wei, G.-Q., Arbter, K. & Hirzinger, G., 1997. Real-Time Visual Servoing Laparoscopic Surgery. *IEEE Engineering in Medicine and Biology,* 16(1), pp. 40-46.

Willmann, J. et al., 2012. Aerial Robotic Construction Towards a New Field of Architectural Research. *International Journal of Architectural Computing,* 10(3), pp. 439-459.

APÊNDICE A INSTALAÇÃO SDK AR.DRONE

Faça download do pacote SDK no site (https://projects.ardrone.org/). Descompacte o arquivo AR.Drone_API-x.x.x.tar.bz2 e coloque em uma pasta de sua escolha. Descompacte via Terminal do linux:

$ tar xjf ARDrone_API-«<ARDversion»>.tar.bz2

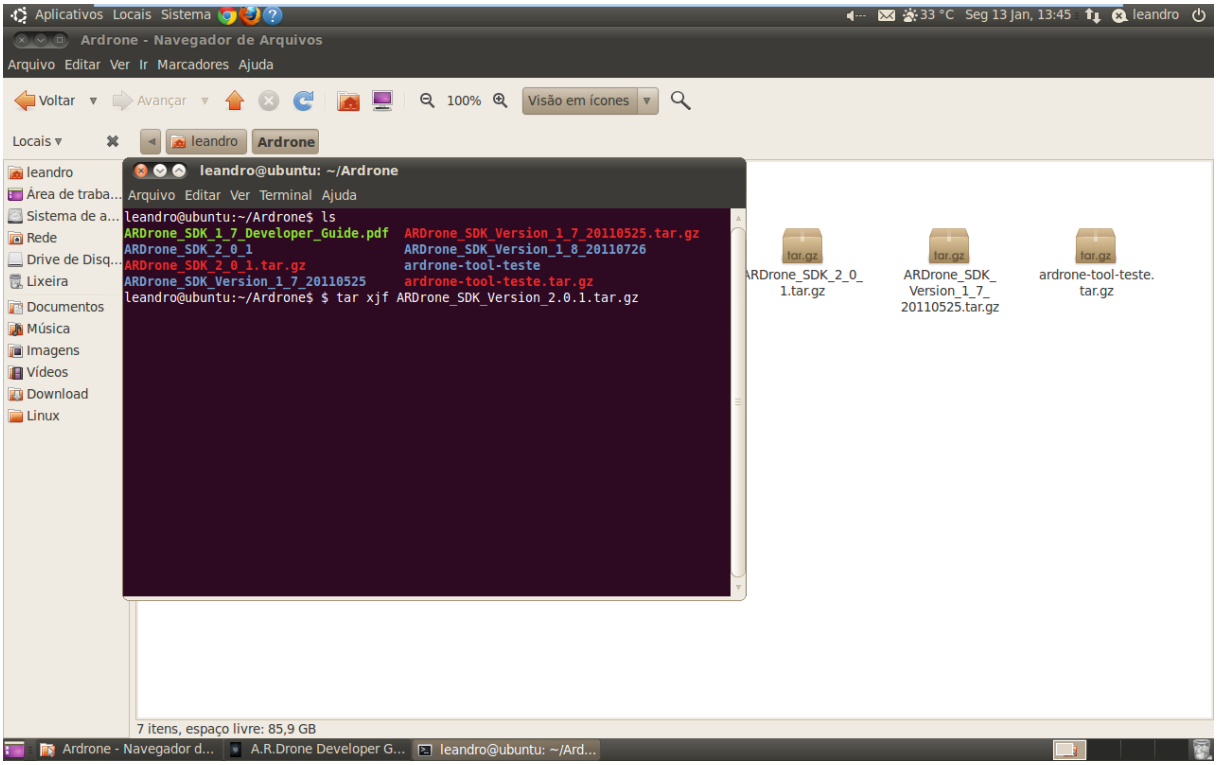

Figura 49 - Descompactando o pacote SDK AR.Drone

Para compilar os exemplos, você precisará de algumas bibliotecas, IW (controlador wireless), gtk (display para interface gráfica), and SDL (display para stream de vídeo). Instalamos com as seguintes linhas :

$ sudo apt-get update

$ sudo apt-get install libsdl-dev libgtk2.0-dev libiw-dev

Utilizando o sistema operacional Linux, devemos modificar o arquivo ARDroneLib/Soft/Build/custom.makefile. Edite a linha:

1 USE_LINUX=no

para

1 USE_LINUX=yes

Agora construa as bibliotecas do ARDroneLIB , usando o comando make:

$ cd SDK/ARDroneLib/Soft/Build
$ make

A compilação teve sucesso se a ultima linha da execução for:

"ar rcs libpc_ardrone.a ..."

O segundo passo é compilar o makefile da pasta Build :

$ cd SDK/Examples/Linux/sdk_demo/Build
$ make

Para executar o linux_sdk_demo , configure o Wi-Fi do AR.Drone como servidor DHCP , entre no diretório do linux_sdk_demo, abra o terminal do Linux e digite:

$ ./ linux_sdk_demo

APÊNDICE B INSTALAÇÃO SDK VICON

Descompacte o arquivo *Vicon_DataStream1.2.59611 Linux 64 Installer* e coloque em uma pasta de sua escolha.

Na pasta, faça no terminal do Linux para transferir as bibliotecas para pasta /usr/lib :

```
$ sudo cp libViconDataStreamSDK_CPP.so  /usr/lib/
```

APÊNDICE C CRIANDO OBJETO NO VICON TRACKER

Na aba *Objects* e com o objeto na campo de visão das câmeras, clique em cada marcador com o botão ctrl apertado.

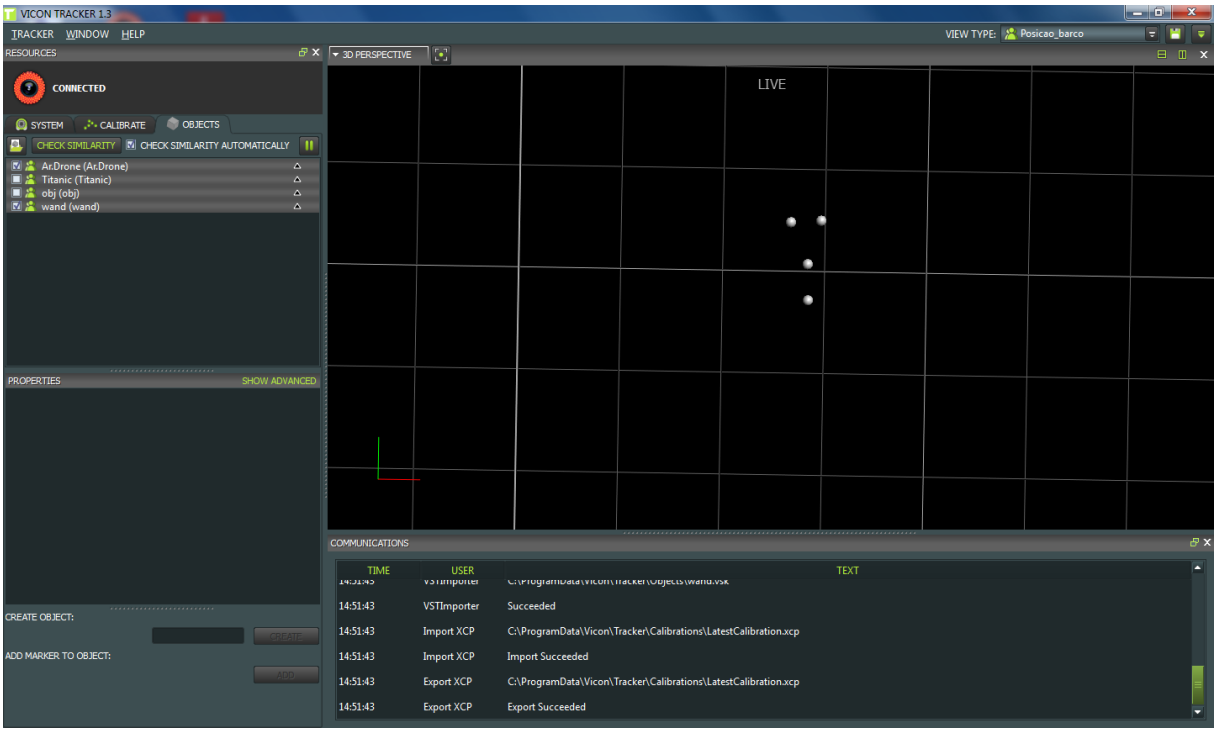

Figura 50 - Visualização dos marcadores do objeto no Vicon Tracker 1.3

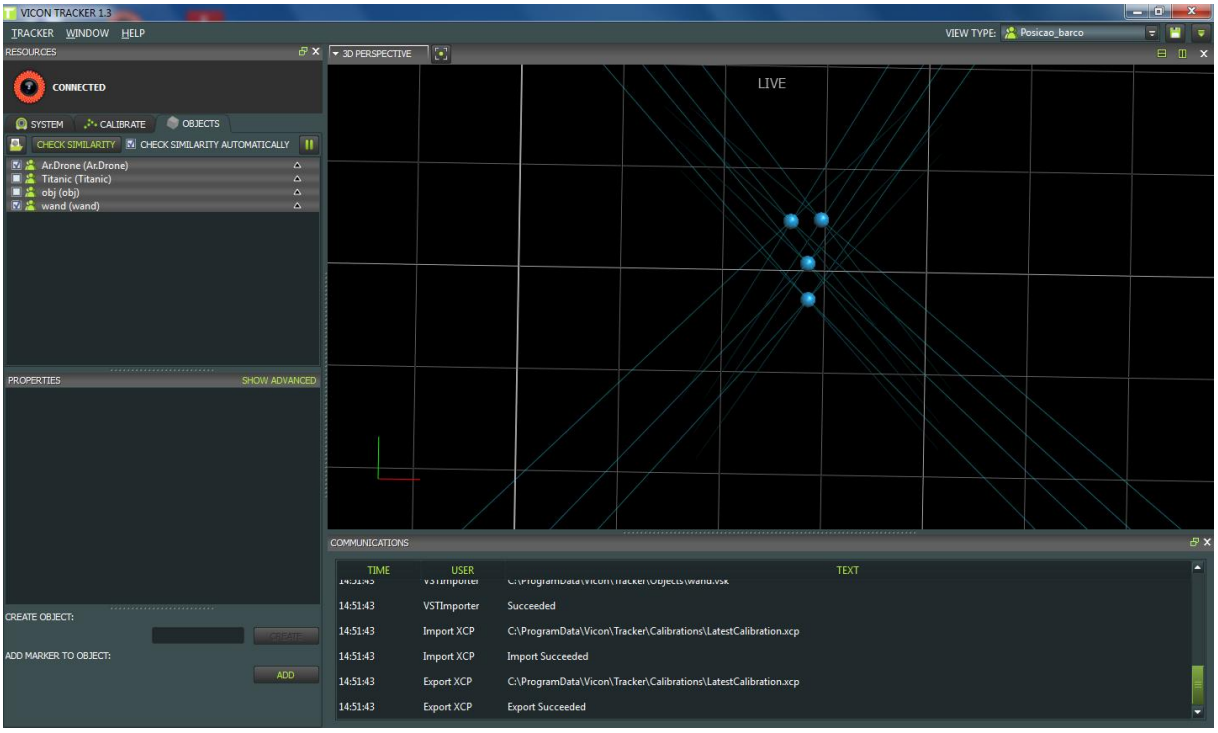

Figura 51 - Selecão dos marcadores do objeto

Em *create object* , nomeie o objeto. Aperte o botão *Create* para confirmar a criação.

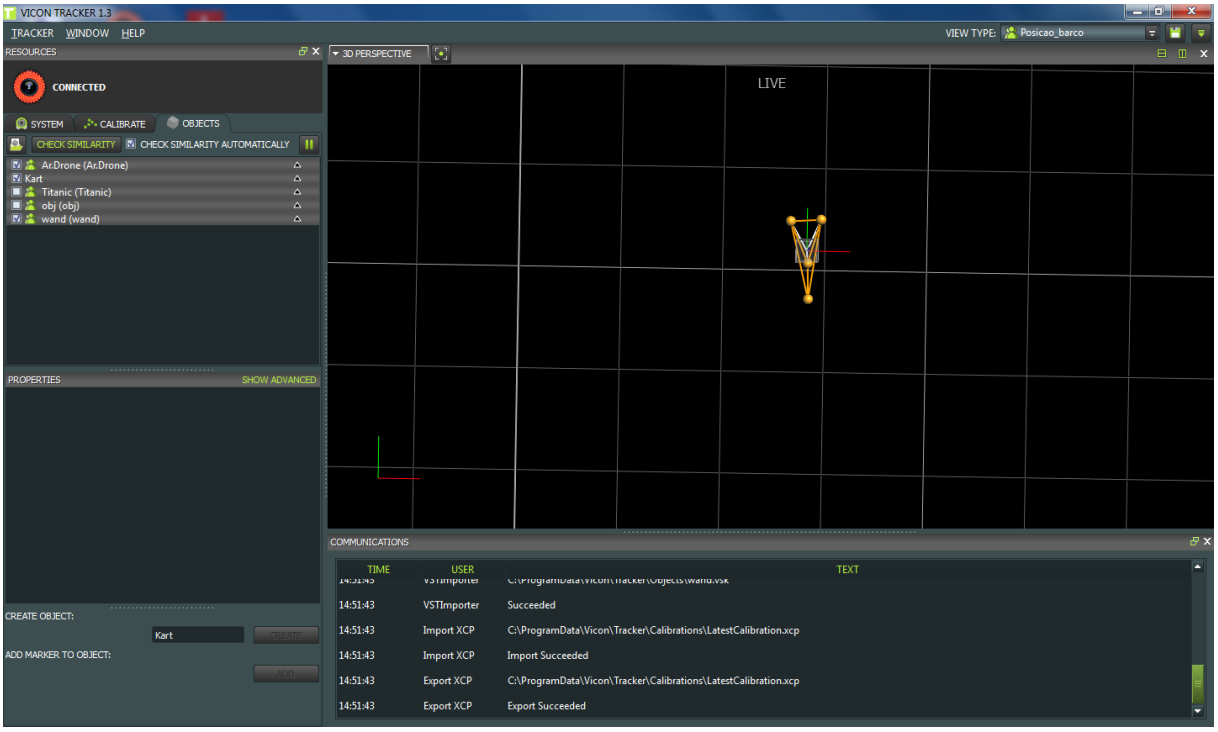

Figura 52 - Objeto Criado

Clique com o botão esquerdo do mouse no objeto criado e em seguida clique em *Save Object*.

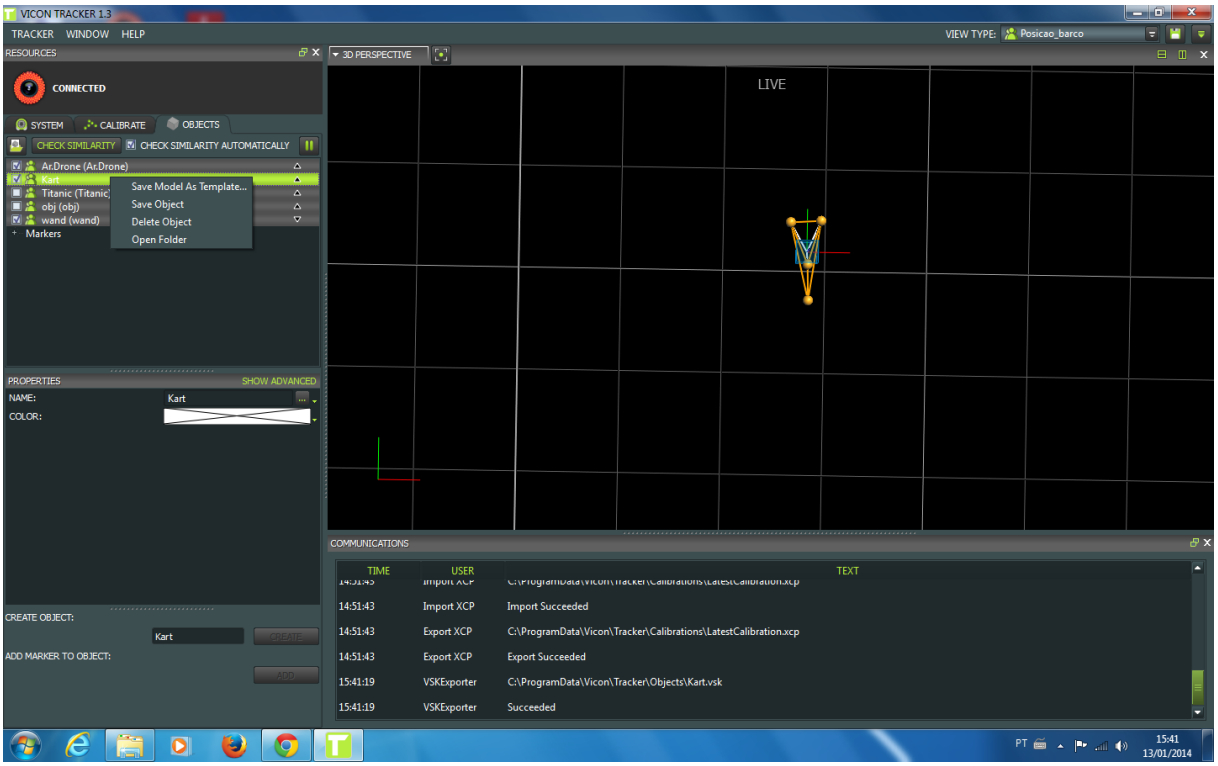

Figura 53 - Objeto criado sendo salvo

APÊNDICE D CALIBRANDO AS CÂMERAS VICON

Na aba *Calibrate* > *Calibrate Câmeras,* clique em no botão *Start.*

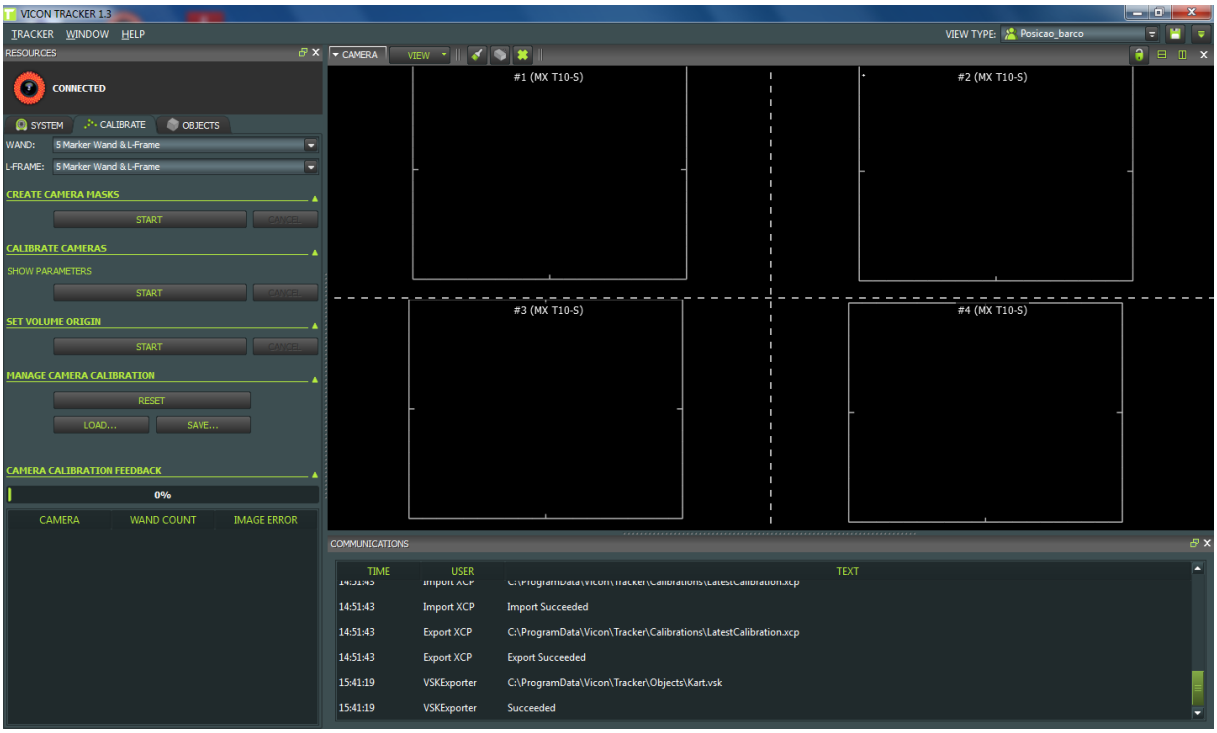

Figura 54 - Aba de calibração

Com a vara de calibração (wand), faça movimentos em toda a área até ficar toda ocupada.

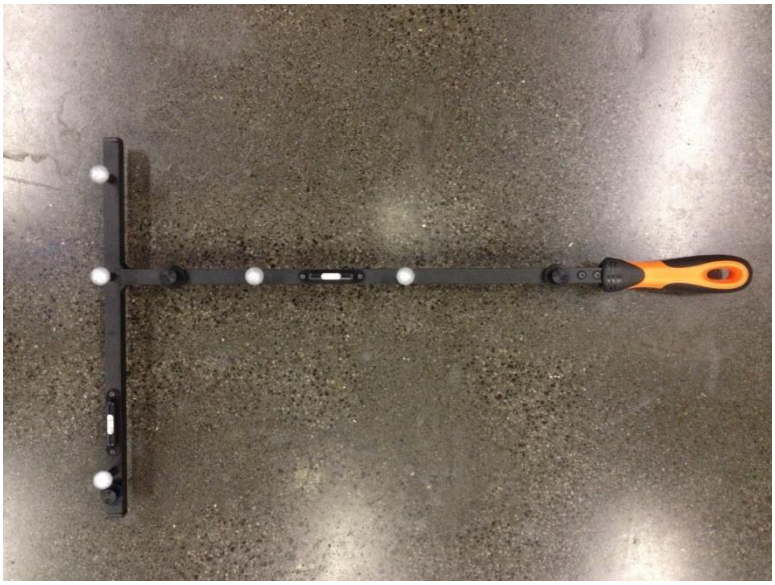

Figura 55 - Vara de calibração – Wand

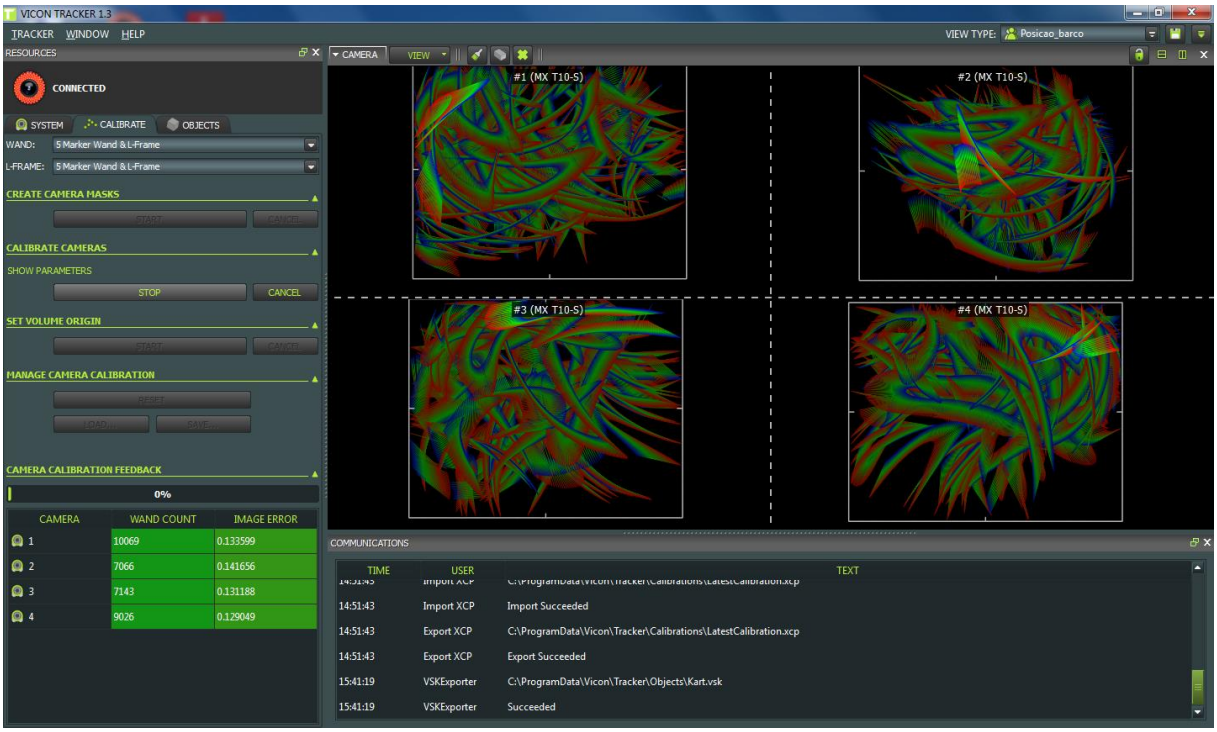

Figura 56 - Calibrando as câmeras

Em seguida, definimos o ponto de origem o (0,0,0) do seu ambiente 3D. Com a vara de calibração coloque no ponto desejado e em *Set Volume Origin* aperte o botão *start* e depois no botão *Set Origin.*

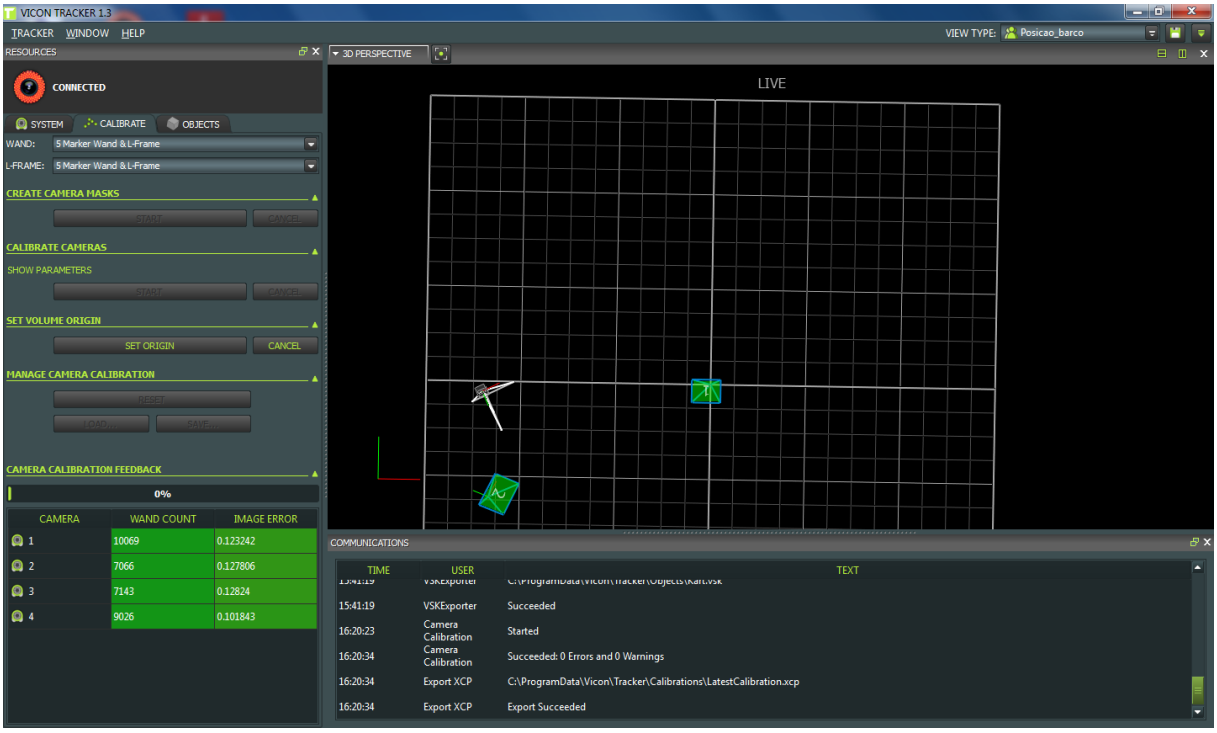

Figura 57 - Configuração do ponto origem do sistema

APÊNDICE E CÓDIGOS-FONTE ALGORITMO

```
#  Programa: makefile
#   Última Atualização: 13/01/14
#
# FEN/UERJ - Faculdade de Engenharia da Universidade do
# Estado do Rio de Janeiro
# Descrição: makefile para compilacao conjunta dos SDK's
# Vicon e AR.Drone
#
# Autores: Leandro Gomes
#                  Lucas Leal
#
# Email:leandrolgms@gmail.com
#       lupleal@gmail.com
#
##############################################################################


SDK_PATH:=$(shell pwd)/../ARDrone_SDK_Version_1_8_20110726/ARDroneLib
PC_TARGET=yes
USE_LINUX=yes

CC =g++

CFLAGS =-Wall -DGNU_LINUX -DUSE_NEW_ATCODEC -
DNO_ARDRONE_MAINLOOP -DTARGET_CPU_X86=1 -DUSE_WIFI -DUSE_VLIB

LDFLAGS =-static

SRC =ardrone_testing_tool.cpp

INCLUDES = -I$(SDK_PATH)/ \
                    -I$(SDK_PATH)/Soft/Lib/ \
                    -I$(SDK_PATH)/Soft/Lib/ardrone_tool/ \
                    -I$(SDK_PATH)/Soft/Common/ \
                    -I$(SDK_PATH)/VP_SDK/ \
                    -I$(SDK_PATH)/VP_SDK/VP_Os/linux/ \
```

```
LIB_PATHS = -L/. \
                    -
L$(SDK_PATH)/Soft/Build/targets_versions/ardrone_lib_PROD_MODE_vlib_Linux_3.
2.0-41-generic-pae_GNU_Linux_gcc_4.6.3 \
                    -
L$(SDK_PATH)/Soft/Build/targets_versions/sdk_PROD_MODE_vlib_Linux_3.2.0-41-
generic-pae_GNU_Linux_gcc_4.6.3 \
                    -
L$(SDK_PATH)/Soft/Build/targets_versions/vlib_PROD_MODE_Linux_3.2.0-41-
generic-pae_GNU_Linux_gcc_4.6.3 \

LIBS =-lpc_ardrone \
     -lgthread-2.0 \
              -lsdk \
              -lpthread \
              -lvlib \
              -lrt \
              -lgtk-x11-2.0 \
         -lViconDataStreamSDK_CPP \

BIN = teste_ardrone


all:
              $(CC) $(CFLAGS) $(LDFLAGS) -o $(BIN) $(SRC) $(INCLUDES)
$(LIB_PATHS) $(LIBS)
```

```
/*****************************************************************
* Programa: control.cpp
*
* Última Atualização: 13/01/14
*
* FEN/UERJ - Faculdade de Engenharia da Universidade do
* Estado do Rio de Janeiro
* Descrição: Thread de controle e integração dos sistemas
* Vicon e AR.Drone
*
* Autores: Leandro Gomes
*                Lucas Leal
*
* Email:leandrolgms@gmail.com
*       lupleal@gmail.com
*
*****************************************************************/



#include <iostream>
#include <fstream>
#include <cassert>
#include <ctime>
#include <cmath>
#include "Client_vicon.h"
//#define PI 3.14159265358979323846264338327950288 4;

extern "C"
{

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <termios.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
```

```
#include <sys/time.h>
#include <time.h>

#include <VP_Api/vp_api.h>
#include <VP_Api/vp_api_error.h>
#include <VP_Api/vp_api_stage.h>



#include <config.h>
#include <VP_Os/vp_os_malloc.h>
#include <VP_Os/vp_os_delay.h>

#include <ardrone_tool/ardrone_tool.h>
#include <ardrone_tool/Com/config_com.h>



#include <ardrone_testing_tool.h>
#include <stdio.h>



//ARDroneLib
#include <ardrone_tool/ardrone_time.h>
#include <ardrone_tool/Navdata/ardrone_navdata_client.h>
#include <ardrone_tool/Control/ardrone_control.h>
#include <ardrone_tool/UI/ardrone_input.h>

//Common
#include <config.h>
#include <ardrone_api.h>

//VP_SDK
#include <ATcodec/ATcodec_api.h>
#include <VP_Os/vp_os_print.h>
#include <VP_Api/vp_api_thread_helper.h>
#include <VP_Os/vp_os_signal.h>
#include <VP_Os/vp_os_delay.h>

//Sistema Vicon
```

```
#include "control.h"



}

// Inicio do Programa

using namespace ViconDataStreamSDK::CPP; //Define o namespace de trabalho


#define output_stream if(!LogFile.empty()) ; else std::cout

namespace
{
  std::string Adapt( const bool i_Value )
  {
    return i_Value ? "True" : "False";
  }

  std::string Adapt( const Direction::Enum i_Direction )
  {
    switch( i_Direction )
    {
      case Direction::Forward:
        return "Forward";
      case Direction::Backward:
        return "Backward";
      case Direction::Left:
        return "Left";
      case Direction::Right:
        return "Right";
      case Direction::Up:
        return "Up";
      case Direction::Down:
        return "Down";
      default:
        return "Unknown";
    }
  }
```

```
            char arq_name[50];
            FILE *fp;



std::string Adapt( const DeviceType::Enum i_DeviceType )
{
  switch( i_DeviceType )
  {
    case DeviceType::ForcePlate:
      return "ForcePlate";
    case DeviceType::Unknown:
    default:
      return "Unknown";
  }
}

std::string Adapt( const Unit::Enum i_Unit )
{
  switch( i_Unit )
  {
    case Unit::Meter:
      return "Meter";
    case Unit::Volt:
      return "Volt";
    case Unit::NewtonMeter:
      return "NewtonMeter";
    case Unit::Newton:
      return "Newton";
    case Unit::Kilogram:
      return "Kilogram";
    case Unit::Second:
      return "Second";
    case Unit::Ampere:
      return "Ampere";
    case Unit::Kelvin:
      return "Kelvin";
    case Unit::Mole:
      return "Mole";
```

```
    case Unit::Candela:
      return "Candela";
    case Unit::Radian:
      return "Radian";
    case Unit::Steradian:
      return "Steradian";
    case Unit::MeterSquared:
      return "MeterSquared";
    case Unit::MeterCubed:
      return "MeterCubed";
    case Unit::MeterPerSecond:
      return "MeterPerSecond";
    case Unit::MeterPerSecondSquared:
      return "MeterPerSecondSquared";
    case Unit::RadianPerSecond:
      return "RadianPerSecond";
    case Unit::RadianPerSecondSquared:
      return "RadianPerSecondSquared";
    case Unit::Hertz:
      return "Hertz";
    case Unit::Joule:
      return "Joule";
    case Unit::Watt:
      return "Watt";
    case Unit::Pascal:
      return "Pascal";
    case Unit::Lumen:
      return "Lumen";

    case Unit::Lux:
      return "Lux";
    case Unit::Coulomb:
      return "Coulomb";
    case Unit::Ohm:
      return "Ohm";
    case Unit::Farad:
      return "Farad";
    case Unit::Weber:
      return "Weber";
```

```
      case Unit::Tesla:
        return "Tesla";
      case Unit::Henry:
        return "Henry";
      case Unit::Siemens:
        return "Siemens";
      case Unit::Becquerel:
        return "Becquerel";
      case Unit::Gray:
        return "Gray";
      case Unit::Sievert:
        return "Sievert";
      case Unit::Katal:
        return "Katal";

      case Unit::Unknown:
      default:
        return "Unknown";
    }
  }
#ifdef WIN32
  bool Hit()
  {
    bool hit = false;
    while( _kbhit() )
    {
      getchar();
      hit = true;
    }
    return hit;
  }
#endif
}


DEFINE_THREAD_ROUTINE(control, data)                    //Inicio da Thread control criada
no ARDronetestingtool

//Declaracao variaveis
```

```
{
        float lim=0.15;
        float kpx=0.6;
        float kpy=0.4;
        float kdx=0.26;
        float kdy=0.4;
        int count=0;
        float ht=0.005;
        float phi=0;
        float thet=0;
        float propx=0;
    float propy=0;
        float dervx=0;
        float dervy=0;
    float erro_x_f;
        float erro_y_f;
        float erro_x_1=0;
        float erro_y_1=0;
        double x_wand;
        double y_wand;
        double x_ardrone;
        double y_ardrone;
        double erro_x;
        double erro_y;
        double norm_R , a_rumo, vec1, vec1_2 ,vec2, vec2_2 ;



        //Ligar Ar.Drone
        ardrone_tool_set_ui_pad_start(1);
        ardrone_at_set_progress_cmd(0,0.0,0.0,0.0,0.0);
        vp_os_delay( 50 );
        vp_os_delay( 50 );
        vp_os_delay( 50 );
        vp_os_delay( 50 );


 // Configuraçao de rede
```

```
  std::string HostName = "10.0.0.1:801"; // IP do computador Host - Vicon Tracker
  std::string LogFile = "";
  std::string MulticastAddress = "244.0.0.0:44801"; //IP de multicast(não utilizado)
  bool ConnectToMultiCast = false;
  bool EnableMultiCast = false;

  std::ofstream ofs;


//Geraçao do arquivo de log
sprintf(arq_name,"log.txt");
            if ((fp=fopen(arq_name, "w")) == NULL)
                    {
                     printf(" O arquivo nao pode ser aberto. \n");
                    exit(1);
                    }



  // Criaçao de um novo client
  Client MyClient;

  for(int i=0; i != 3; ++i) // repeat to check disconnecting doesn't wreck next connect
  {
    // Connect to a server
    std::cout << "Connecting to " << HostName << " ..." << std::flush;
    while( !MyClient.IsConnected().Connected )
    {
      // Direct connection

      bool ok = false;
      if(ConnectToMultiCast)
      {
        // Multicast connection
        ok = ( MyClient.ConnectToMulticast( HostName, MulticastAddress ).Result ==
Result::Success );

      }
      else
```

```
    {
      ok =( MyClient.Connect( HostName ).Result == Result::Success );
    }
    if(!ok)
    {
      std::cout << "Warning - connect failed..." << std::endl;
    }


    std::cout << ".";
#ifdef WIN32
    Sleep( 200 );
#else
    sleep(1);
#endif
  }
  std::cout << std::endl;

  // Enable some different data types
  MyClient.EnableSegmentData();
  MyClient.EnableMarkerData();
  MyClient.EnableUnlabeledMarkerData();
  MyClient.EnableDeviceData();

  std::cout << "Segment Data Enabled: " << Adapt( MyClient.IsSegmentDataEnabled().Enabled ) << std::endl;
  std::cout << "Marker Data Enabled: " << Adapt( MyClient.IsMarkerDataEnabled().Enabled ) << std::endl;
  std::cout << "Unlabeled Marker Data Enabled: " << Adapt( MyClient.IsUnlabeledMarkerDataEnabled().Enabled ) << std::endl;
  std::cout << "Device Data Enabled: " << Adapt( MyClient.IsDeviceDataEnabled().Enabled ) << std::endl;

  // Set the streaming mode
  MyClient.SetStreamMode( ViconDataStreamSDK::CPP::StreamMode::ClientPull );
  //MyClient.SetStreamMode( ViconDataStreamSDK::CPP::StreamMode::ClientPullPreFetch );
  //MyClient.SetStreamMode( ViconDataStreamSDK::CPP::StreamMode::ServerPush );

  // Set the global up axis
```

```
  MyClient.SetAxisMapping( Direction::Forward,
                           Direction::Left,
                           Direction::Up ); // Z-up
  // MyClient.SetGlobalUpAxis( Direction::Forward,
  //                           Direction::Up,
  //                           Direction::Right ); // Y-up

  Output_GetAxisMapping _Output_GetAxisMapping = MyClient.GetAxisMapping();
  std::cout << "Axis Mapping: X-" << Adapt( _Output_GetAxisMapping.XAxis )
                          << " Y-" << Adapt( _Output_GetAxisMapping.YAxis )
                          << " Z-" << Adapt( _Output_GetAxisMapping.ZAxis ) << std::endl;

  // Discover the version number
  Output_GetVersion _Output_GetVersion = MyClient.GetVersion();
  std::cout << "Version: " << _Output_GetVersion.Major << "."
                           << _Output_GetVersion.Minor << "."
                           << _Output_GetVersion.Point << std::endl;

  if( EnableMultiCast )
  {
    assert( MyClient.IsConnected().Connected );
    MyClient.StartTransmittingMulticast( HostName, MulticastAddress );
  }

  size_t FrameRateWindow = 100; // frames
  size_t Counter = 0;
  clock_t LastTime = clock();




  // Loop para Recebimento de Quadro
#ifdef WIN32
  while( !Hit() )
#else
  while( true)
#endif
  {
    // Get a frame
```

```
    output_stream << "Waiting for new frame...";
    while( MyClient.GetFrame().Result != Result::Success )
    {
      // Sleep a little so that we don't lumber the CPU with a busy poll
      #ifdef WIN32
        Sleep( 200 );
      #else
        sleep(1);
      #endif

      output_stream << ".";
    }
    output_stream << std::endl;
    if(++Counter == FrameRateWindow)
    {
      clock_t Now = clock();
      double FrameRate = (double)(FrameRateWindow * CLOCKS_PER_SEC) / (double)(Now -
LastTime);
      if(!LogFile.empty())
      {
        time_t rawtime;
        struct tm * timeinfo;
        time ( &rawtime );
        timeinfo = localtime ( &rawtime );

        ofs << "Frame rate = " << FrameRate << " at " <<  asctime (timeinfo)<< std::endl;
      }

      LastTime = Now;
      Counter = 0;
    }

    // Get the frame number
    Output_GetFrameNumber _Output_GetFrameNumber = MyClient.GetFrameNumber();
    output_stream << "Frame Number: " << _Output_GetFrameNumber.FrameNumber <<
std::endl;

    Output_GetFrameRate Rate = MyClient.GetFrameRate();
    std::cout << "Frame rate: "          << Rate.FrameRateHz          << std::endl;
```

```
    // Get the latency
    output_stream << "Latency: " << MyClient.GetLatencyTotal().Total << "s" << std::endl;

    for( unsigned int LatencySampleIndex = 0 ; LatencySampleIndex < MyClient.GetLatencySampleCount().Count ; ++LatencySampleIndex )
    {
      std::string SampleName = MyClient.GetLatencySampleName( LatencySampleIndex ).Name;
      double SampleValue = MyClient.GetLatencySampleValue( SampleName ).Value;

      output_stream << "  " << SampleName << " " << SampleValue << "s" << std::endl;
    }
    output_stream << std::endl;

    // Count the number of subjects
    unsigned int SubjectCount = MyClient.GetSubjectCount().SubjectCount;
    output_stream << "Subjects (" << SubjectCount << "):" << std::endl;
    for( unsigned int SubjectIndex = 0 ; SubjectIndex < SubjectCount ; ++SubjectIndex )
    {
      output_stream << "  Subject #" << SubjectIndex << std::endl;

      // Get the subject name
      std::string SubjectName = MyClient.GetSubjectName( SubjectIndex ).SubjectName;
      output_stream << "    Name: " << SubjectName << std::endl;

      // Get the root segment
      std::string RootSegment = MyClient.GetSubjectRootSegmentName( SubjectName ).SegmentName;
      output_stream << "    Root Segment: " << RootSegment << std::endl;

      // Count the number of segments
      unsigned int SegmentCount = MyClient.GetSegmentCount( SubjectName ).SegmentCount;
      output_stream << "    Segments (" << SegmentCount << "):" << std::endl;
      for( unsigned int SegmentIndex = 0 ; SegmentIndex < SegmentCount ; ++SegmentIndex )
      {
        output_stream << "      Segment #" << SegmentIndex << std::endl;
```

```
        // Get the segment name
        std::string SegmentName = MyClient.GetSegmentName( SubjectName, SegmentIndex
).SegmentName;
        output_stream << "        Name: " << SegmentName << std::endl;


        // Get the global segment translation
        Output_GetSegmentGlobalTranslation _Output_GetSegmentGlobalTranslation =
          MyClient.GetSegmentGlobalTranslation( SubjectName, SegmentName );
        output_stream     <<     "                                    Global     Translation:     ("     <<
_Output_GetSegmentGlobalTranslation.Translation[ 0 ]  << ", "
                                    <<  _Output_GetSegmentGlobalTranslation.Translation[  1  ]
<< ", "
                                    <<  _Output_GetSegmentGlobalTranslation.Translation[  2  ]
<< ") "
                                    <<  Adapt(  _Output_GetSegmentGlobalTranslation.Occluded
) << std::endl;
        // Get the local segment translation
        Output_GetSegmentLocalTranslation _Output_GetSegmentLocalTranslation =
          MyClient.GetSegmentLocalTranslation( SubjectName, SegmentName );
        output_stream     <<     "                                    Local     Translation:     ("     <<
_Output_GetSegmentLocalTranslation.Translation[ 0 ]  << ", "
                                   << _Output_GetSegmentLocalTranslation.Translation[ 1 ]  <<
", "
                                   << _Output_GetSegmentLocalTranslation.Translation[ 2 ]  <<
") "
                                   <<  Adapt(  _Output_GetSegmentLocalTranslation.Occluded  )
<< std::endl;


            }
      // Count the number of markers
      unsigned int MarkerCount = MyClient.GetMarkerCount( SubjectName ).MarkerCount;
      output_stream << "    Markers (" << MarkerCount << "):" << std::endl;
      for( unsigned int MarkerIndex = 0 ; MarkerIndex < MarkerCount ; ++MarkerIndex )
      {
        // Get the marker name
        std::string MarkerName = MyClient.GetMarkerName( SubjectName, MarkerIndex
).MarkerName;
```

```
        // Get the marker parent
        std::string MarkerParentName = MyClient.GetMarkerParentName( SubjectName,
MarkerName ).SegmentName;

        // Get the global marker translation
        Output_GetMarkerGlobalTranslation _Output_GetMarkerGlobalTranslation =
          MyClient.GetMarkerGlobalTranslation( SubjectName, MarkerName );

        output_stream << "      Marker #" << MarkerIndex           << ": "
                                  << MarkerName           << " ("
                                  << _Output_GetMarkerGlobalTranslation.Translation[ 0 ]  << ", "
                                  << _Output_GetMarkerGlobalTranslation.Translation[ 1 ]  << ", "
                                  << _Output_GetMarkerGlobalTranslation.Translation[ 2 ]  << ") "
                                  <<    Adapt(    _Output_GetMarkerGlobalTranslation.Occluded    )    <<
std::endl;
       }
     }

     // Get the unlabeled markers
     unsigned int UnlabeledMarkerCount = MyClient.GetUnlabeledMarkerCount().MarkerCount;
     output_stream << "    Unlabeled Markers (" << UnlabeledMarkerCount << "):" << std::endl;
     for(    unsigned    int    UnlabeledMarkerIndex    =    0    ;    UnlabeledMarkerIndex    <
UnlabeledMarkerCount ; ++UnlabeledMarkerIndex )
     {
       // Get the global marker translation
       Output_GetUnlabeledMarkerGlobalTranslation
_Output_GetUnlabeledMarkerGlobalTranslation =
         MyClient.GetUnlabeledMarkerGlobalTranslation( UnlabeledMarkerIndex );

       output_stream << "      Marker #" << UnlabeledMarkerIndex   << ": ("
                                 << _Output_GetUnlabeledMarkerGlobalTranslation.Translation[ 0 ] << ",
"
                                 << _Output_GetUnlabeledMarkerGlobalTranslation.Translation[ 1 ] << ",
"
                                 <<  _Output_GetUnlabeledMarkerGlobalTranslation.Translation[  2  ]  <<
")" << std::endl;
     }
```

```
    // Count the number of devices
    unsigned int DeviceCount = MyClient.GetDeviceCount().DeviceCount;
    output_stream << "  Devices (" << DeviceCount << "):" << std::endl;
    for( unsigned int DeviceIndex = 0 ; DeviceIndex < DeviceCount ; ++DeviceIndex )
    {
      output_stream << "    Device #" << DeviceIndex << ":" << std::endl;

      // Get the device name and type
      Output_GetDeviceName _Output_GetDeviceName = MyClient.GetDeviceName( DeviceIndex );
      output_stream << "      Name: " << _Output_GetDeviceName.DeviceName << std::endl;
      output_stream << "      Type: " << Adapt( _Output_GetDeviceName.DeviceType ) << std::endl;

      // Count the number of device outputs
      unsigned int DeviceOutputCount = MyClient.GetDeviceOutputCount( _Output_GetDeviceName.DeviceName ).DeviceOutputCount;
      output_stream << "      Device Outputs (" << DeviceOutputCount << "):" << std::endl;
      for( unsigned int DeviceOutputIndex = 0 ; DeviceOutputIndex < DeviceOutputCount ; ++DeviceOutputIndex )
      {
        // Get the device output name and unit
        Output_GetDeviceOutputName _Output_GetDeviceOutputName =
          MyClient.GetDeviceOutputName( _Output_GetDeviceName.DeviceName, DeviceOutputIndex );

        unsigned int DeviceOutputSubsamples =
                MyClient.GetDeviceOutputSubsamples(
_Output_GetDeviceName.DeviceName,
                                        _Output_GetDeviceOutputName.DeviceOutputName
).DeviceOutputSubsamples;

        output_stream << "      Device Output #" << DeviceOutputIndex << ":" << std::endl;
        output_stream << "      Samples (" << DeviceOutputSubsamples << "):" << std::endl;

        for( unsigned int DeviceOutputSubsample = 0; DeviceOutputSubsample < DeviceOutputSubsamples; ++DeviceOutputSubsample )
        {
          output_stream << "        Sample #" << DeviceOutputSubsample << ":" << std::endl;
```

```
        // Get the device output value
        Output_GetDeviceOutputValue _Output_GetDeviceOutputValue =
          MyClient.GetDeviceOutputValue( _Output_GetDeviceName.DeviceName,
                                         _Output_GetDeviceOutputName.DeviceOutputName,
                                         DeviceOutputSubsample );

        output_stream << "             "' << _Output_GetDeviceOutputName.DeviceOutputName
<< "' "
                                  << _Output_GetDeviceOutputValue.Value                 << " "
                                  << Adapt( _Output_GetDeviceOutputName.DeviceOutputUnit ) << " "
                                  <<  Adapt(  _Output_GetDeviceOutputValue.Occluded  )            <<
std::endl;
       }
      }
     }



          //Inicio Controle

          //Calculo Vetor Orientacao



          Output_GetMarkerGlobalTranslation _marker1 =
        MyClient.GetMarkerGlobalTranslation( "Ar.Drone", "centro" );
          double x_ardrone_centro = _marker1.Translation[0];
          double y_ardrone_centro = _marker1.Translation[1];

          Output_GetMarkerGlobalTranslation _marker2 =
        MyClient.GetMarkerGlobalTranslation( "Ar.Drone", "centro_tras" );
          double x_ardrone_centro_tras = _marker2.Translation[0];
          double y_ardrone_centro_tras = _marker2.Translation[1];

          double R[2];
          //double i[2];

          R[0] = x_ardrone_centro - x_ardrone_centro_tras ;
          R[1] = y_ardrone_centro - y_ardrone_centro_tras ;
```

```
            vec1=R[0];
            vec1_2=vec1*vec1;
            vec2=R[1];
            vec2_2=vec2*vec2;

            //norma dos vetores
            norm_R = sqrt(vec1_2+vec2_2);

            //Calculo Angulo de Rumo

            if (norm_R == 0) {

                  norm_R=1;
                  phi=0.0;
                  thet=0.0;
                  count++;
                  a_rumo = (PI/2) - acos((vec2)/(norm_R)); // Angulo em radianos;
                        }

else {

            a_rumo = (PI/2) - acos((vec2)/(norm_R)); // Angulo em radianos;

            //Recebe as posiçoes em x e y do Alvo e Ar.drone

            Output_GetSegmentGlobalTranslation _teste1 =
              MyClient.GetSegmentGlobalTranslation( "wand", "wand" );
            x_wand = _teste1.Translation[ 0 ];
            y_wand = _teste1.Translation[ 1 ];

            Output_GetSegmentGlobalTranslation _teste =
              MyClient.GetSegmentGlobalTranslation( "Ar.Drone", "Ar.Drone" );
            x_ardrone = _teste.Translation[ 0 ];
            y_ardrone = _teste.Translation[ 1 ];
```

```
        //Multiplicaçao das Posiçoes Obtidas por uma Matrix de Rotaçao

        double pos_ardrone[2] = {x_ardrone, y_ardrone};
        double pos_wand[2] = {x_wand, y_wand};

        double                                                  pos_ardrone_R[2]=
{(pos_ardrone[0]*cos(a_rumo))+(pos_ardrone[1]*sin(a_rumo)),
                        (-pos_ardrone[0]*sin(a_rumo))+(pos_ardrone[1]*cos(a_rumo))};


        double pos_wand_R[2] = {(pos_wand[0]*cos(a_rumo))+(pos_wand[1]*sin(a_rumo)),
                        (-pos_wand[0]*sin(a_rumo))+(pos_wand[1]*cos(a_rumo))};

        //Calculo dos Erros em x e y

        erro_x = (pos_wand_R[0] - pos_ardrone_R[0])/1000;
        erro_y = (pos_wand_R[1] - pos_ardrone_R[1])/1000;

        //conversao de double para float
        erro_x_f = (float) (erro_x);
        erro_y_f = (float) (erro_y);


// Controle PD

propx= kpx*erro_x_f;
propy= kpy*erro_y_f;
dervx= (kdx)*(erro_x_f - erro_x_1)/ht;
dervy= (kdy)*(erro_y_f - erro_y_1)/ht;

//Definicao dos paramentros de açao de movimento

thet=propx + dervx;
phi=propy + dervy;
erro_x_1=erro_x_f;
erro_y_1=erro_y_f;
count++;
```

```
//limitação da angulacao(aceleracao) do AR.Drone


                if (phi>=lim) {
                        phi=lim;
                }

                else {

                if (phi <= (-lim)) {
                        phi=(-lim);
                                }
                   }



                if (thet>=lim) {
                        thet=lim;
                }

                else {

                if (thet <= (-lim)) {
                        thet=(-lim);
                                }
                   }



//Adequacao ao valores dos movimentos segundo SDK.
thet=thet*(-1);
phi=phi*(-1);


if (thet==-0.0) {
                        thet=0.0;
                                }
```

```
                    if (phi==-0.0) {
                            phi=0.0;
                                    }



//Envia a acao de movimento pro AR.Drone
ardrone_at_reset_com_watchdog();
ardrone_at_set_progress_cmd(1,phi,thet,0.0,0.0);

//Geraçao de Log

fprintf(fp,"%f %f %d %f %f %f %f \n", x_ardrone/1000, y_ardrone/1000, count, thet, phi, x_wand/1000, y_wand/1000);

//fprintf(fp,"Phi= %f, Theta= %f , erro_x_f= %f, erro_y_f= %f , angulo= %f ,propx= %f, propy= %f ,dervx= %f, dervy= %f,Frame_Number = %d Contador= %d \n\n", phi,thet,erro_x_f,erro_y_f,a_rumo, propx, propy, dervx , dervy , frame_number , count);

}



}



//Desconexao da Vicon

   if( EnableMultiCast )
   {
     MyClient.StopTransmittingMulticast();
   }
   MyClient.DisableSegmentData();
   MyClient.DisableMarkerData();
   MyClient.DisableUnlabeledMarkerData();
   MyClient.DisableDeviceData();

   // Disconnect and dispose
```

```
    int t = clock();
    std::cout << " Disconnecting..." << std::endl;
    MyClient.Disconnect();
    int dt = clock() - t;
    double secs = (double) (dt)/(double)CLOCKS_PER_SEC;
    std::cout << " Disconnect time = " << secs << " secs" << std::endl;
            ardrone_tool_set_ui_pad_start(0);//desligar
  }
return (THREAD_RET)0; // Fim da Thread
}
```

```
/*****************************************************************
* Programa: ardrone_testing_tool.c
*
* Última Atualização: 13/01/14
*
* FEN/UERJ - Faculdade de Engenharia da Universidade do
* Estado do Rio de Janeiro
* Descrição: Programa principal do SDK Ar.Drone
*
*
* Autores: Leandro Gomes
*                Lucas Leal
*
* Email:leandrolgms@gmail.com
*       lupleal@gmail.com
*
*****************************************************************/
#ifdef __cplusplus
extern "C"                    // Necessário caso utilize junto com o Sistema Vicon
{
#endif

#include <ardrone_testing_tool.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <ncurses.h>
#include <string.h>


//ARDroneLib
#include <ardrone_tool/ardrone_time.h>
#include <ardrone_tool/Navdata/ardrone_navdata_client.h>
#include <ardrone_tool/Control/ardrone_control.h>
#include <ardrone_tool/UI/ardrone_input.h>
```

```
//Common
#include <config.h>
#include <ardrone_api.h>
#include<config_keys.h>
#include <ardrone_tool/Control/ardrone_control_configuration.h>
#include <ardrone_tool/ardrone_tool_configuration.h>

//VP_SDK
#include <ATcodec/ATcodec_api.h>
#include <VP_Os/vp_os_print.h>
#include <VP_Api/vp_api_thread_helper.h>
#include <VP_Os/vp_os_signal.h>
#include <VP_Os/vp_os_delay.h>

//Local project
#include <control.h>

//Sistema Vicon
//#include "Client_vicon.h"

//#include <UI/gamepad.h>
//#include <Video/video_stage.h>

static int32_t exit_ihm_program = 1;
//static Joystick *joypad;

/* Implementing Custom methods for the main function of an ARDrone application */


/* The delegate object calls this method during initialization of an ARDrone application */
C_RESULT ardrone_tool_init_custom(int argc, char **argv)
{

  /* Registering for a new device of game controller */
  //ardrone_tool_input_add( &gamepad );


  /* Start all threads of your application */
```

```
   //START_THREAD( video_stage, NULL );

  START_THREAD( control, NULL ); //Começa a Thread Criada de controle


  return C_OK;

}

/* The delegate object calls this method when the event loop exit */
C_RESULT ardrone_tool_shutdown_custom()
{
            ardrone_tool_set_ui_pad_start(0);
  /* Relinquish all threads of your application */

   //JOIN_THREAD( video_stage );
  JOIN_THREAD( control );                //Junta a Thread ao Programa principal do Ardrone

  /* Unregistering for the current device */


  return C_OK;
}

/* The event loop calls this method for the exit condition */
bool_t ardrone_tool_exit()
{
  return exit_ihm_program == 0;
}

C_RESULT signal_exit()
{
  exit_ihm_program = 0;

  return C_OK;
}
```

```
BEGIN_THREAD_TABLE
  //THREAD_TABLE_ENTRY( video_stage, 20 )
   THREAD_TABLE_ENTRY( ardrone_control, 20 )
  //THREAD_TABLE_ENTRY( navdata_update, 20 )
  THREAD_TABLE_ENTRY( control, 20 )                    //Declaracao da thread Criada

END_THREAD_TABLE

#ifdef __cplusplus
}
#endif
```