JOSÉ RENATO BULHÕES VICENTE

INSTRUMENTAÇÃO PARA A MODELAGEM, IDENTIFICAÇÃO E CONTROLE DE UM VANT DE PEQUENA ESCALA

São Carlos, Brasil

Novembro de 2014

JOSÉ RENATO BULHÕES VICENTE

INSTRUMENTAÇÃO PARA A MODELAGEM, IDENTIFICAÇÃO E CONTROLE DE UM VANT DE PEQUENA ESCALA

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em Eletrônica

Orientador: Prof. Dr. Marco Henrique Terra

São Carlos, Brasil

Novembro de 2014

Vicente, José Renato Bulhões
V632i Instrumentação para a modelagem, identificação e controle de um VANT de pequena escala / José Renato Bulhões Vicente; orientador Marco Henrique Terra. São Carlos, 2014.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Instrumentação. 2. Identificação de Modelos. 3. Modelo de Espaço de Estado. 4. Modelagem. 5. VANT autônomo. 6. Controle. I. Título.

FOLHA DE APROVAÇÃO

Nome: José Renato Bulhões Vicente

Título: "Instrumentação para a modelagem, identificação e controle de um VANT de pequena escala"

Trabalho de Conclusão de Curso defendido e aprovado em 27 / 11 / 2014,

com NOTA 9,0 (nove, zero), **pela Comissão Julgadora:**

Prof. Associado Marco Henrique Terra - (Orientador - SEL/EESC/USP)

Profa. Titular Vilma Alves de Oliveira - (SEL/EESC/USP)

Mestre Samuel Lourenço Nogueira - (Doutorando - SEM/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Agradecimentos

Gostaria de agradecer ao Prof. Dr. Marco Henrique Terra pela oportunidade de poder desenvolver este trabalho no LASI, gostaria de agradecer também ao Prof. Dr. Roberto Santos Inoue por orientar nos momentos de dúvida, e também ao aluno de mestrado Willian Leão e ao aluno de doutorado Maurício Nakai, pelas contribuições a este trabalho. Também gostaria de agradecer ao CNPq por financiar a pesquisa de iniciação científica que deu origem a este trabalho.

Resumo

Este trabalho apresenta uma instrumentação de *software* e *hardware* para a identificação e controle de um veículo aéreo não tripulado (VANT), baseando-se em um modelo de espaço de estados obtido na literatura. Essa instrumentação envolve o desenvolvimento de códigos de aquisição e de tratamento de dados, e também de comunicação entre o computador e o VANT.

Palavras-chaves: Instrumentação; Identificação de Modelos; Modelo de Espaço de Estado; Modelagem; VANT Autônomo; Controle.

Abstract

This work aims the instrumentation of hardware and software to identify and control an unmanned aerial vehicle (UAV), based on a state space model obtained from the literature. This involves the development of instrumentation codes acquisition and data processing, as well as communication between the computer and the UAV.

***Keywords**: Instrumentation; Model Identification; State-space model; Modelling; Autonomous AUV; Control.*

Lista de figuras

Lista de tabelas

Sumário

Introdução

Hoje, em todos os setores econômicos, os sistemas automáticos estão muito presentes, dentre estes setores agrícola, industrial, comercial e predial. Isto deve-se aos avanços das tecnologias de controle e automação, que possibilitaram a popularização de máquinas e robôs autônomos, em especial dos veículos autônomos, cuja utilização nos campos privados, comercial e militar cresce expressivamente. Dentre estes veículos, destacam-se os VANTs, acrônimo de Veículos Aéreos Não Tripulados. Após a popularização das baterias de íons de lítio, de microcontroladores e microprocessadores de alto desempenho, avanços em tecnologias de transmissão de dados e de posicionamento global, os custos desses veículos têm diminuído, além de contribuírem para o desenvolvimento de veículos mais confiáveis, versáteis e com maior autonomia.

Um dos principais tipos de VANTs são os robôs helicópteros (INOUE, 2011), que comparados aos quadricópteros, tem uma eficiência energética maior pois contam com um número menor de motores, no máximo dois.

Um robô helicóptero é um dispositivo mecatrônico de alta complexidade e manobrabilidade, o que provê a capacidade de deslocamento em qualquer direção, podendo realizar vôos pairados (velocidade zero), em altas velocidades e sobrevoar à baixas e altas altitudes. Portanto, o desenvolvimento de um robô helicóptero não tripulado não é algo trivial, pois requer conhecimentos interdisciplinares nas áreas de engenharia espacial, engenharia elétrica, comunicações, ciência da computação, controle de sistema, inteligência artificial, operações em sistema em tempo real, entre outras. Por possuir uma dinâmica é não linear e está sujeito a perturbações externas como rajadas de vento, além de oferecer alto risco ao operador, visto que se trata de um dispositivo mecânico controlado eletronicamente com altas velocidades de rotação em suas hélices.

A construção de um robô helicóptero autônomo requer alto grau de robustez devido à própria natureza do dispositivo. Para controlar de forma satisfatória o veículo, é necessário um conhecimento profundo das características dinâmicas do robô helicóptero. O desejo de controlar uma aeronave utilizando técnicas avançadas de controle requer um modelo matemático adequado que represente de maneira satisfatória as características dinâmicas de vôo. Desde 1990, notou-se o crescimento do número de pesquisas realizadas com modelagem dinâmica de helicópteros de pequena escala. Entretanto, esse tema ainda apresenta um vasto campo de oportunidades a serem desenvolvidas.

O presente projeto insere-se nas pesquisas realizadas no Laboratório de Sistemas Inteligentes (LASI) do Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade de São Paulo.

Parte I

Objetivos e Revisão Bibliográfica

1.0 Objetivos

A proposta inicial deste projeto era modelar, fazer a aquisição de dados e projetar um controlador que a partir da base (Figura 1) desenvolvida no Laboratório de Sistemas Inteligentes (LASI). No qual o objetivo está em estabilizar o robô helicóptero como se estivesse em um voo pairado.

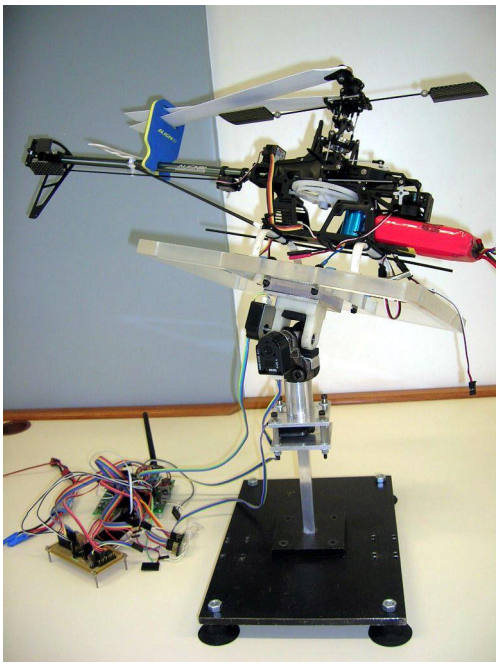

Figura 1 – Base 3D desenvolvida no LASI.

Uma das grande desvantagens do uso da base 3D é que o modelo de espaço de estado do helicóptero é alterado quando é preso a base, pois com a adição do peso e da dinâmica da base todo o momento de inércia e dinâmica do conjunto é modificado fazendo com que o controlador desenvolvido não possa ser usado com o helicóptero livre, esse problema fez com que a proposta inicial fosse modificada. A nova proposta é explicada a seguir.

1.1 Nova proposta

Além dos fatores já citados, no início de 2014 o LASI adquiriu um sistema de deteção de movimento que consiste em um conjunto de câmeras do modelo *T-Series T40S* (Figura 2), da marca Vicon, cuja especificações podem ser vistas da Tabela 1 que utiliza a triangulação das

imagens obtidas pelas câmeras (Figura 3) dos marcadores refletivos (Figura 4) colocados no objeto de estudo, possibilitando uma melhor identificação do objeto no espaço.

Tabela 1 – Tabela com especificações da câmera modelo *T-Series T40S*

Resolução(Megapixels)	4.0
Número máximos de quadros na resolução máxima	515
Sensor	Vicon Vegas-S-4
Processamento dos marcadores *on-board*	Sim
Processamento da seleção dos dados *on-board*	Sim
Alimentação	Giganet
Lente	12,5 mm
Campo de visão (H○ x V○)	67 x 52

Figura 2 – Câmera Vicon modelo *T-series T40S*.

Com isso o objetivo do projeto foi alterado, pois com a utilização dessa nova tecnologia possibilita-se uma identificação mais precisa dos objetos, além de facilitar a aquisição dos dados, pois todo o trabalho de captura dos dados é feito automaticamente através do *software Tracker*, também da empresa *Vicon*. A obtenção do posicionamento dos marcadores é em tempo real, fazendo com que seja possível, a realimentação utilizando o próprio conjuto, *Tracker* e câmeras.

Com isto, o objetivo do projeto foi alterado para a identificação do modelo de espaço de estado de um helicóptero com um único rotor central e com hélices de passo variável, cujo modelo de espaço de estado será descrito posteriormente.

Figura 3 – Posicionamento das câmeras T40S no laboratório.

Figura 4 – Marcador refletivo.

2.0 Modelagem

Popularmente, nos dias de hoje, as pesquisas sobre controle de VANT estão voltadas em sua maioria para os quadricópteros, pois existe uma vasta bibliografia de modelagem, identificação e controle destes veículos (MELLINGER, 2012; NAIDOO R. STOPFORTH, 2011; RICH, 2012), trabalhos desenvolvidos na *Swiss Federal Institute of Technology* exploram a dinâmica dos quadricópteros (HEHN, 2011), utilizando um sistema de detecção de movimento semelhante ao utilizado neste projeto.

Neste projeto a opção pelo estudo de helicópteros, ao invés de quadricópteros, deve-se ao fato de, por possuírem apenas um motor elétrico, sua eficiência energética é muito maior, melhorando a autonomia e consequente redução de peso, devido a uma bateria menor. Além disso, existe um grande campo inexplorado sobre desenvolvimento de helicópteros autônomos, pois em sua grande maioria os estudos, hoje em dia, são voltados aos quadricópteros.

Inicialmente o modelo escolhido para ser modelado é o do helicóptero de rádio controle Blade 130X, que pode ser visto na Figura 5. A escolha deste modelo deve-se a sua semelhança com a modelagem proposta, pois ambos possuem apenas um rotor, e são de hélices de passo variável.

Figura 5 – Helicóptero de radio controle Blade 130X, fonte site do fabricante.

Antes de iniciar o processo de modelagem deve-se compreender o funcionamento dos diversos mecanismos presentes em um helicóptero não-tripulado. O sinal de controle utilizado para helicópteros é o PCM (Modulação por Código de Pulsos, derivado do inglês *Pulse-code Modulation*), este é utilizado devido a ser altamente robusto. Este sinal, quando recebido pelo receptor do helicóptero, é demodulado e é enviado para os servomotores, cada servomotor é

uma atuador que responde a um pulso PWM (Modulação por largura de pulso) enviado pelo receptor, no helicóptero utilizado no laboratório são utilizados 5 servomotores, cada um atua em diferentes partes do helicóptero. Existe uma dependência do controle do rotor de cauda com o giroscópio presente do helicóptero, a ação do giroscópio é sobre o eixo de arfagem, com o sentido de estabilizar o helicóptero, como por exemplo quando existe um vento lateral agindo sobre o helicóptero, assim ação do giroscópio evita que o ângulo de arfagem seja alterado.

De acordo com a literatura (YI; SASTRY, 1998), na Figura 6, pode-se observar que a modelagem pode ser dividida em quatro subsistemas, dinâmica dos atuadores, dinâmica das hélices, geração de força e torque, e dinâmica do corpo rígido.

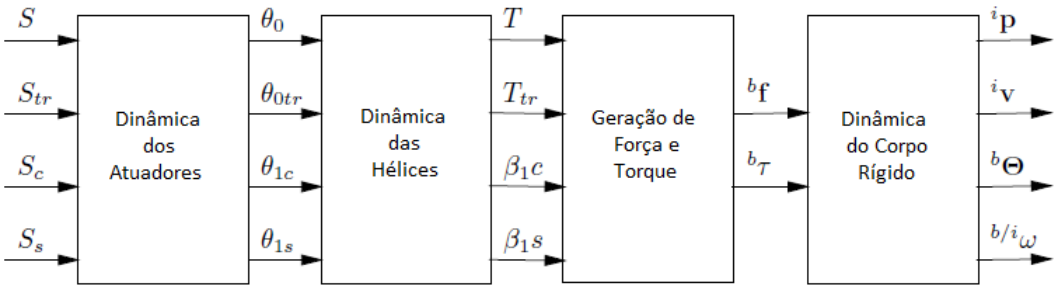

Figura 6 – Dinâmica do helicóptero.

2.1 Modelo dos servomotores

Antes de iniciar a modelagem dos servomotores, deve-se entender suas principais características. Servomotores podem ser do tipo analógicos e digitais, sua frequência característica de repetição pode ser entre 50 Hz (modelos analógicos) e 300 Hz (modelos digitais), o sinal de controle é do tipo PWM, a posição do eixo principal dependerá da razão cíclica do sinal PWM, existem mais duas características que definem cada servomotor, que são o tempo mínimo e máximo do pulso, e esses estão relacionados com a posição mínima e máxima do eixo do servomotor, respectivamente.

Os servomotores utilizados no helicóptero estudado são analógicos, a frequência de repetição é de 50Hz, o tempo mínimo de pulso (define a posição mínima do eixo) é de 1ms, o tempo máximo de pulso (define a posição máxima do eixo) é de 2ms, a posição central é obtida quando o tempo do pulso é de 1,52ms.

A modelagem dos atuadores sem carga em seu eixo foi obtida com a aquisição da posição do eixo do servomotor dada uma entrada degrau, assim podemos observar a sua resposta e encontrar o modelo mais apropriado para sua representação, na literatura (FOGH M., 2004) observamos que existem dois comportamento distintos, para o servo do rotor principal pode ser descrito por uma função de transferência de primeira ordem, definida pela Equação 2.1

$$\frac{\theta_0(s)}{S(s)} = \frac{K}{s + \frac{1}{\tau_t}} \tag{2.1}$$

onde a constante de tempo τ_t pode ser obtida quando a resposta ao degrau é igual a 63% do estado de equilíbrio. Têm-se uma resposta semelhante para os comportamentos de θ_{1c} e θ_{1s},

justificados pelo uso do mesmo atuador, assim temos a Equação 2.2.

$$\frac{\theta_0(s)}{S(s)} = \frac{\theta_{1c}(s)}{S_c(s)} = \frac{\theta_{1s}(s)}{S_s(s)} = \frac{K}{s + \frac{1}{\tau_t}} \tag{2.2}$$

Para o comportamento do rotor de cauda, tem-se uma configuração diferente porque este é ligado ao giroscópio, assim do mesmo modo que fora determinado para o rotor principal temos na Equação 2.3.

$$\frac{\theta_{0tr}(s)}{S_{tr}(s)} = \frac{K_{tr}}{s + \frac{1}{\tau_{tr}}} \tag{2.3}$$

Os valores obtidos, na literatura, podem ser observados na Tabela 2.

Tabela 2 – Valores de especificação dos atuadores

Constante	Valor
K	4,34
K_{tr}	12,2
τ_t	0,25 s
τ_{tr}	0,1 s

A modelagem dos outros subsistemas pode ser feita da mesma forma que a modelagem dos servomotores foi executada, assim como foi feito em (FOGH M., 2004) a interdependência entre os diversos subsistemas do helicóptero pode ser observada na Figura 7.

Pesquisando na literatura obtivemos um modelos de espaço de estado que é utilizado comumente para helicópteros (Yamaha RMAX (Figura 8) do mesmo tipo que o utilizado no laboratório, embora o modelo utilizado em laboratório seja menor e elétrico. Este modelo (METTLER B., 1999) é descrito na Seção 2.2. A Figura 9 mostra a diferença de tamanho entre o Yamaha RMAX descrito na literatura e o Blade 130X utilizado no laboratório para este projeto.

2.2 Modelo de espaço de estado

A modelagem no espaço de estado, para o modelo linearizado do helicóptero, foi obtida na literatura (METTLER B., 1999) para o modelo de helicóptero Yamaha R-50 , embora esse modelo seja muito maior que o modelo utilizado no laboratório, é possível, futuramente, determinar os parâmetros para o helicóptero utilizado neste trabalho.

O modelo no espaço de estado, pode ser visto na Equação 2.4, a matriz C dependerá do tipo de saída que deseja-se observar, assim existem diversos tipos de matrizes C possíveis.

$$\left\{ \begin{array}{c} \dot{x} = Ax + Bu \\ \dot{y} = Cx \end{array} \right\} \tag{2.4}$$

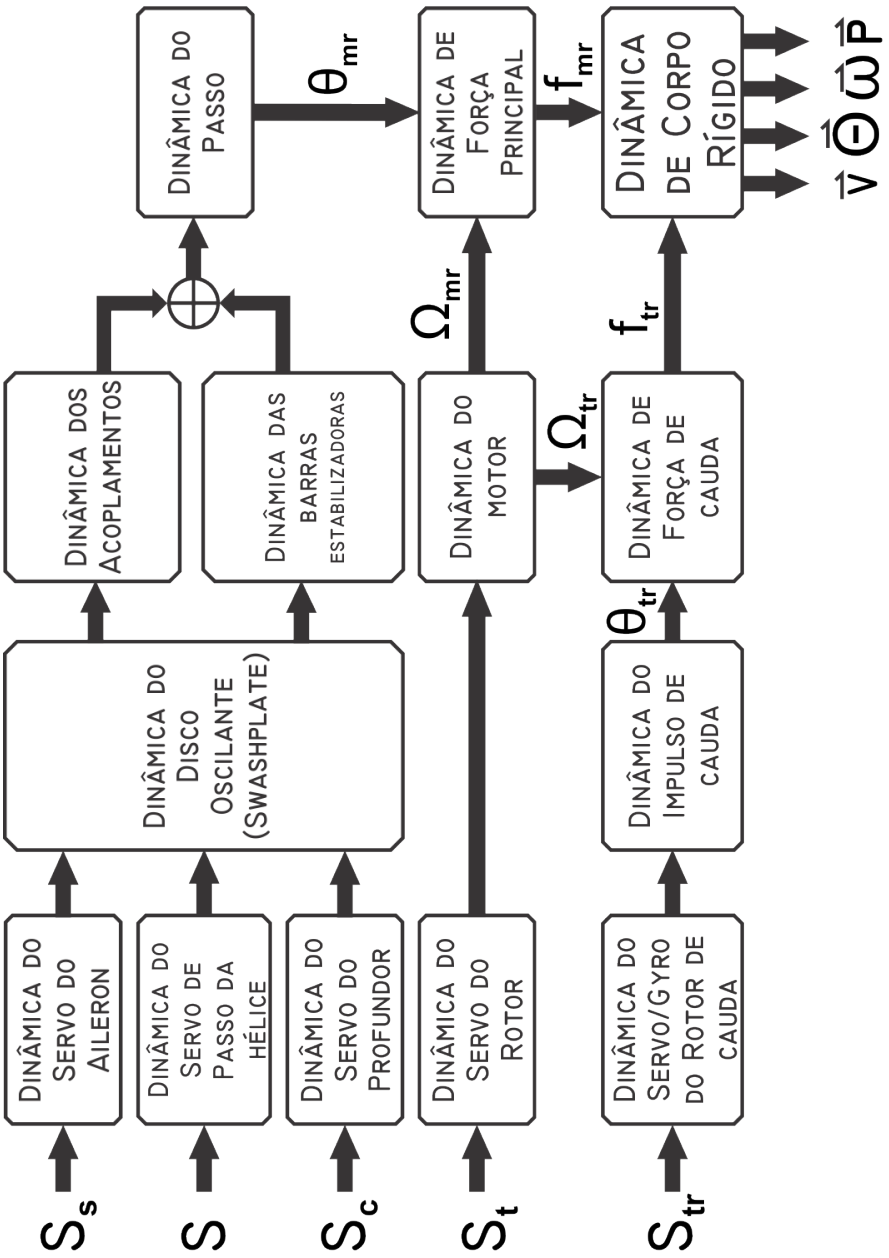

Figura 7 – Subsistemas considerados na modelagem de espaço de estados, e suas dependências.

onde as Equações 2.5, 2.6, 2.7 e 2.8, mostram, respectivamente, as matrizes A, B, x e u no

Figura 8 – Imagem do helicóptero Yamaha RMAX, fonte site do fabricante.

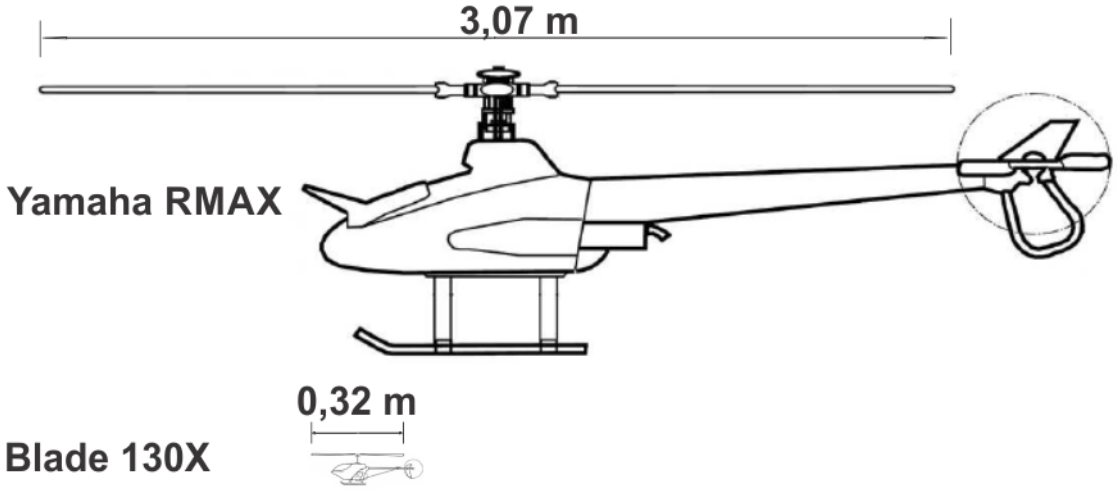

Figura 9 – Comparação de tamanho entre o Yamaha RMAX e o Blade 130X.

espaço de estado do modelo linearizado do helicóptero.

$$\mathbf{A} = \begin{bmatrix} X_u & 0 & 0 & 0 & 0 & -g & -X_{a1s} & 0 & 0 & 0 & 0 \\ 0 & Y_v & 0 & 0 & g & 0 & 0 & Y_{b1s} & 0 & 0 & 0 \\ L_u & L_v & 0 & 0 & 0 & 0 & L_{a1s} & L_{b1s} & 0 & 0 & 0 \\ M_u & M_v & 0 & 0 & 0 & 0 & M_{a1s} & M_{b1s} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \frac{-1}{\tau_f} & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & B_{a1s} & \frac{-1}{\tau_f} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_{a1s} & Z_{b1s} & Z_w & Z_r & 0 \\ 0 & 0 & N_p & 0 & 0 & 0 & 0 & 0 & N_w & N_r & MN_{ped} \\ 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & K_r & MK_{rfb} \end{bmatrix} \quad (2.5)$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ A_{lat} & A_{lon} & 0 & 0 \\ B_{lat} & B_{lon} & 0 & 0 \\ 0 & 0 & 0 & Z_{col} \\ 0 & 0 & N_{ped} & N_{col} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.6}$$

$$\mathbf{x} = \begin{bmatrix} v_x & v_y & \omega_x & \omega_y & \phi & \theta & a_{1s} & b_{1s} & v_z & \omega_z & r_{zfb} \end{bmatrix}^T \tag{2.7}$$

$$\mathrm{u} = \begin{bmatrix} \delta_{lat} & \delta_{lon} & \delta_{ped} & \delta_{col} \end{bmatrix}^T \tag{2.8}$$

No vetor de estados **x** (2.7), v_x representa a velocidade longitudinal, v_y a velocidade lateral, ω_x e ω_y, representam, respectivamente, a velocidade angular sobre o eixo x e y, ϕ e θ representam, respectivamente, o ângulo sobre o eixo x e y, a_{1s} e b_{1s} representam, respectivamente, os ângulos de *flapping* longitudinal e lateral, v_z a velocidade vertical, ω_z a velocidade angular sobre o eixo z e r_{zfb} representa o coeficiente de amortecimento para a velocidade angular sobre o eixo z, o sistema de coordenadas utilizado é referênte ao centro de massa do helicóptero.

No vetor das variáveis de entrada **u** (2.8), δ_{lat} representa o comando cíclico lateral, δ_{lon} o comando cíclico longitudinal, δ_{ped} o comando de leme (guinada) e δ_{col} o comando de arfagem.

A partir deste modelo, pode-se identificar os parâmetros para o helicóptero utilizado em laboratório, com a ajuda da *toolbox ident* do *software MATLAB*, para isso é necessário a obtenção dos sinais de entrada do modelo e dos sinais de saída. A aquisição destes sinais poderá ser executada pelo sistema de câmera da *Vicon*, a aquisição poderá ser vista nos capítulos seguintes.

Parte II

Resultados

3.0 Aquisição dos Dados das Câmeras

Para possibilitar a identificação do modelo de espaço de estados descrito anteriormente, deve-se desenvolver métodos de aquisição dos dados das câmeras, também é necessário adquirir os sinais de entrada no sistemas que é feito através do rádio controle *Spektrum* do modelo *DX7s* (Figura 10). Tanto a aquisição dos dados de saída de entrada são feitos utilizando as câmeras. Para isso é necessário desenvolver os códigos de aquisição de dados, utilizando o *software MATLAB*.

Junto com o treinamento obtido junto aos representantes da empresa *Vicon*, foram fornecidos a documentação necessária para a utilização das câmeras e do *Software Tracker* (Figura 11), os códigos podem ser desenvolvidos em algumas linguagens como *C++, script MATLAB* e *Microsoft .NET* (VICON, 2013).

3.1 Aquisição dos dados utilizando *MATLAB*

Como existe uma familiaridade com o uso do *software MATLAB* nos estudos de controle, principalmente pela facilidade de operações com matrizes e também pela existência de *toolboxes* especificamente para tratar dados de controle, optou-se pelo desenvolvimento do código de aquisição dos dados das câmeras com o *software MATLAB*.

Para a interpretação do *script* desenvolvido é necessário que os compiladores *Microsoft Visual C++ 2010* e o *Microsoft Windows SDK 7.1*, com estes compiladores é o *script* pode ser interpretado pelo software e os dados podem ser adquiridos. Alguns problemas podem acontecer quando utiliza-se o sistema operacional de *64 bits*, recomenda-se desinstalar completamente os compiladores *Microsoft Visual C++ 2010* de *32* e *64 bits*, antes da instalação do compilador *Microsoft Windows SDK 7.1*. Após a instalação deste compilador ser concluída com sucesso deve-se executar o comando `mex -setup` e selecionar a opção do compilador instalado, no caso o *Microsoft Windows SDK 7.1*. Em sistemas operacionais de *32 bits* não foi verificado este tipo de problema.

Como é possível a diferenciação dos objetos detectados pelas câmeras, possibilitando a detecção simultânea de diversos objetos, assim o *script* desenvolvido separa cada objeto em uma lista que contém as suas coordenadas cartesianas e ângulos de interesse, além da posição no espaço de cada marcador do objeto. O código pode ser observado no Anexo A.

Após desenvolver este código, uma interface (Figura 12) para visualização e gravação dos dados foi feita, está interface agrega todos os códigos de aquisição, tratamento, gravação e visualização. Futuramente poderão ser adicionados os códigos de identificação e controle do sistema. Também neste programa foi desenvolvido uma janela para a visualização do movimento

Figura 10 – Rádio *Spektrum* modelo *DX7s*, com os marcadores de identificação das câmeras.

do helicóptero (Figura 13) adquirido pelo sistema de detecção de movimento.

3.2 Aquisição dos comando de entrada

A aquisição dos comandos de entrada poderia ser feita de três formas diferentes, inicialmente foi proposto utilizar dois rádio e um circuito de armazenamento de dados, assim poderias utilizar um rádio como passivo que se comunicaria com o sistema de armazenamento de dados, e o sistema após ler os dados recebidos, os enviava para o outro rádio para que esse se comunicasse com o helicóptero. O problema desse sistema era que não conseguiríamos uma sincronia com a aquisição dos sinais de saída.

Outra proposta seria utilizar um circuito receptor do sinais de 2,4 GHz enviado do rádio para

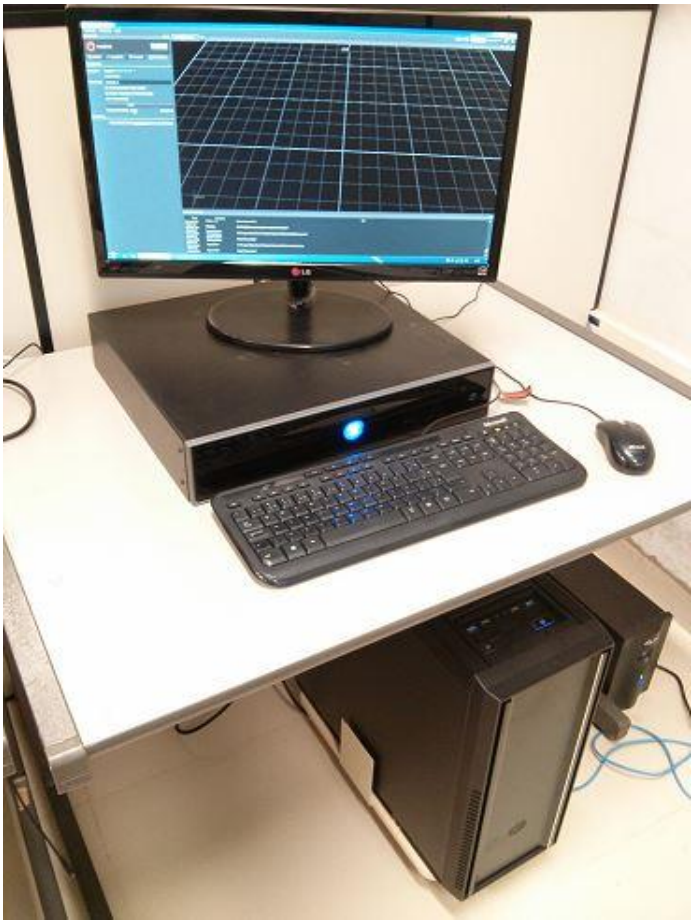

Figura 11 – Computador das câmeras e o *software Tracker*.

o helicóptero, o nome usual desse tipo de circuito é farejador(*sniffer*), observamos dois problemas para esse tipo de abordagem, além do problema de sincronização, igual ao da proposta anterior, ele teria o problema de descobrir e decodificar em qual canal o helicóptero o rádio controle estaria se comunicando.

A aquisição então foi feita utilizando o próprio sistema de detecção de movimento, assim resolvendo o problema de sincronia, pois os sinais de entrada e de saída seriam amostrados concomitantemente, para que isso fosse possível foi desenvolvido um conjunto de hastes de alumínio onde os marcadores foram fixados em cada comando do rádio, este conjunto pode ser visto na Figura 10, e em detalhes cada conjunto de marcadores pode ser visto na Figura 14.

Foram necessários 12 marcadores ao todo, apenas para a aquisição dos sinais de entrada, pois quatro foram fixados no rádio para servirem de referência para os outros conjuntos, cada conjunto fixados nos comandos do rádio continham quatro marcadores cada. Na teoria seriam necessários apenas 3 marcadores, por comando, para que a adquisição dos dados de entrada fosse possível, mas notamos que a redundância de ter um marcador a mais, faz com que o sistema de aquisição fique muito mais estável, pois caso algum marcador saia do campo de visão das câmeras, mesmo assim o sistema continua identificando o objeto.

Após o tratamento dos dados adquiridos podemos observar os gráficos gerados para o comando de rotação sobre o eixo de guinada (Figura 15), coletivo (Figura 16) e lateral (Figura 17) do

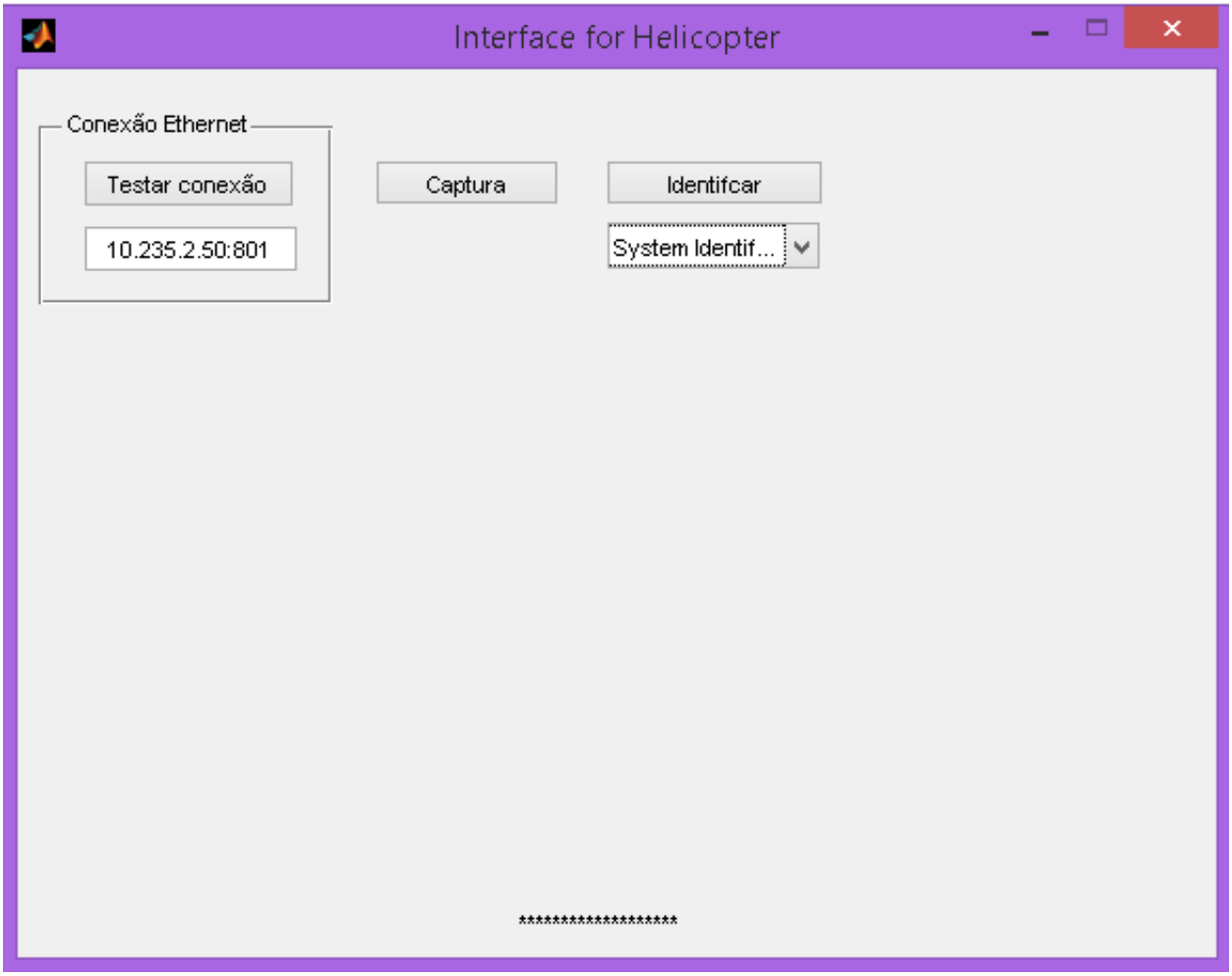

Figura 12 – Interface de integração dos códigos de aquisição, tratamento, gravação e visualização.

helicóptero.

No gráfico do comando e coletivo do sistema, pode-se observar um pico de sinal perto dos 650 ms, este pico é devido ao erro de leitura da câmera durante a aquisição dos dados, este erro ocorreu pois dois marcadores foram cobertos, pelo operador, por um instante, fazendo com que a câmera perdesse a referência do objeto. Com os gráficos obtidos pode-se notar que é válido a aquisição dos dados de entrada com este método.

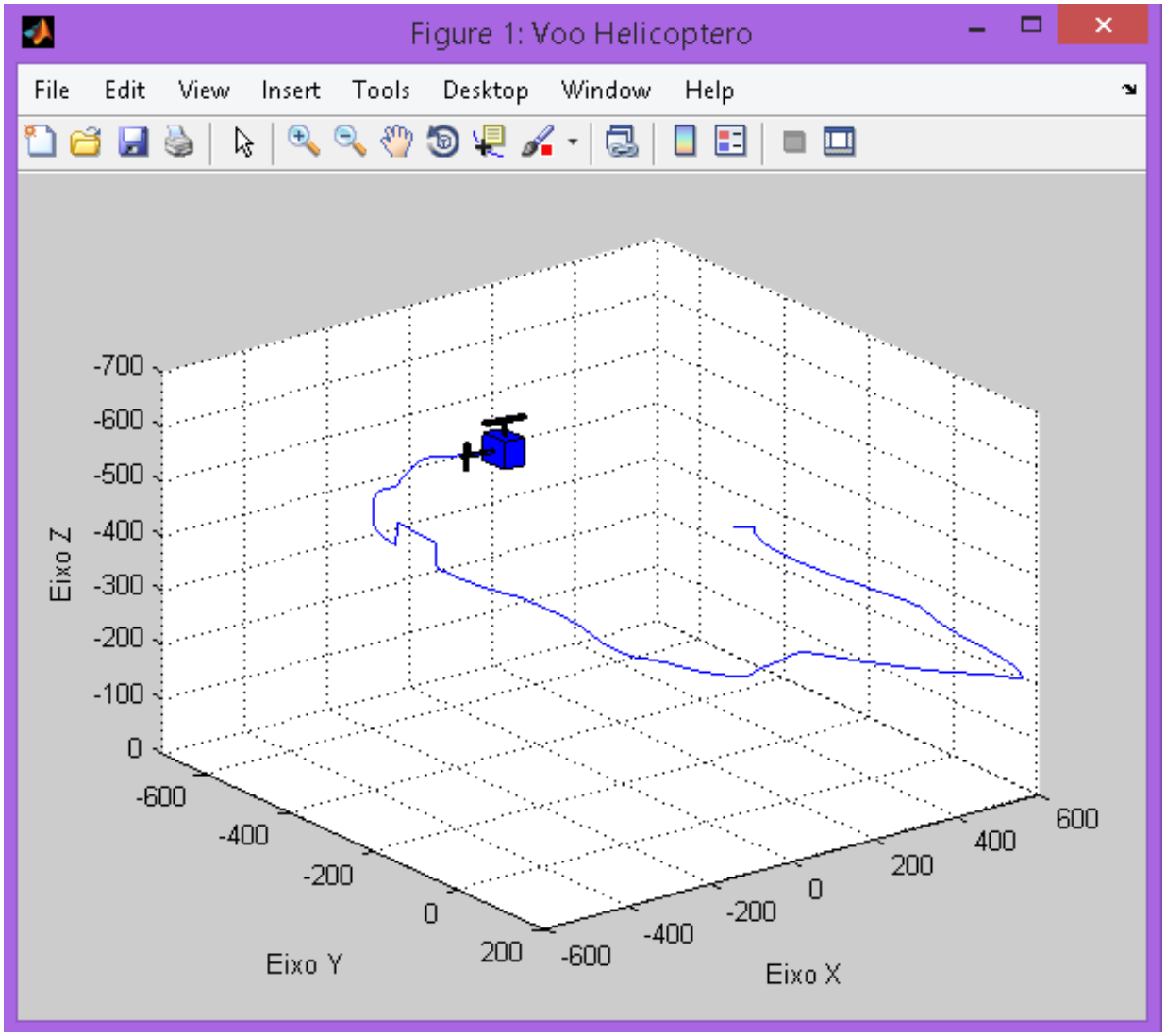

Figura 13 – Interface de visualização do movimento do helicóptero.

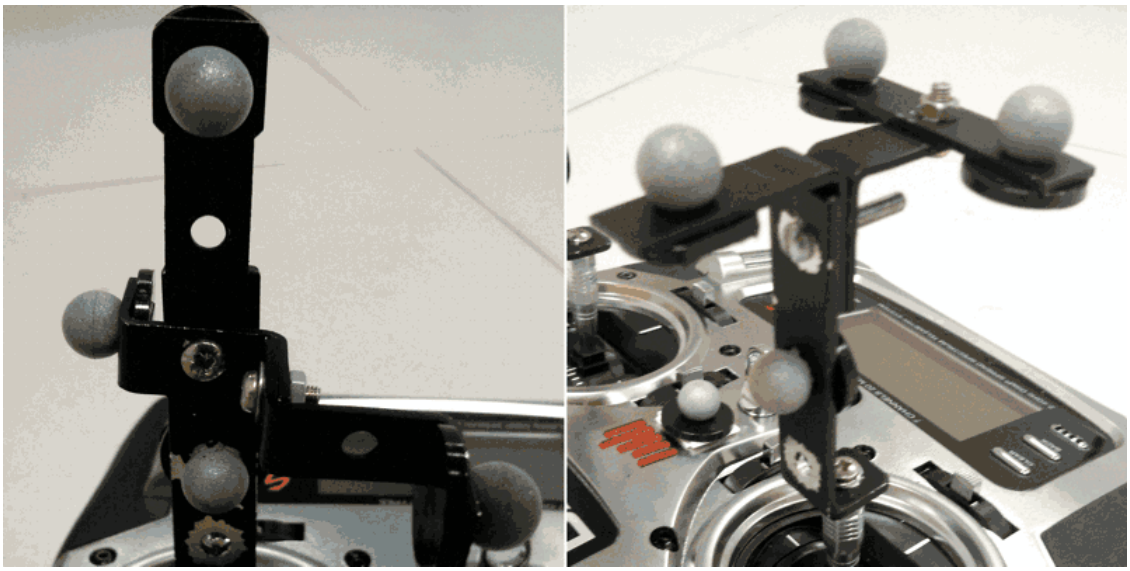

Figura 14 – Detalhe das hastes de alumínio para a fixação dos marcadores nos comando do rádio controle.

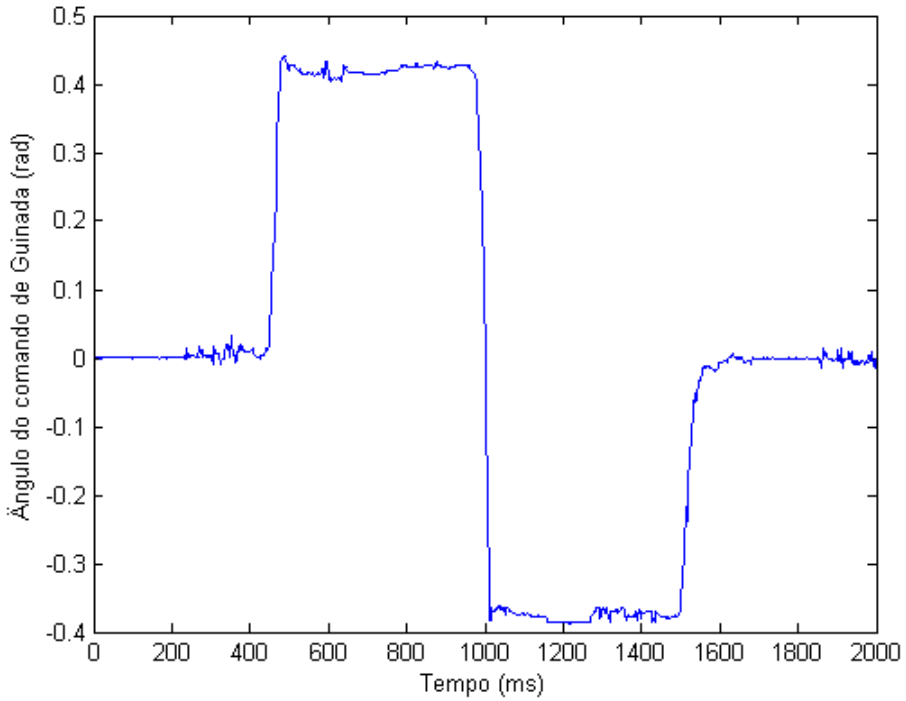

Figura 15 – Ângulo do comando de guinada adquirido pelo sistema de detecção de movimento.

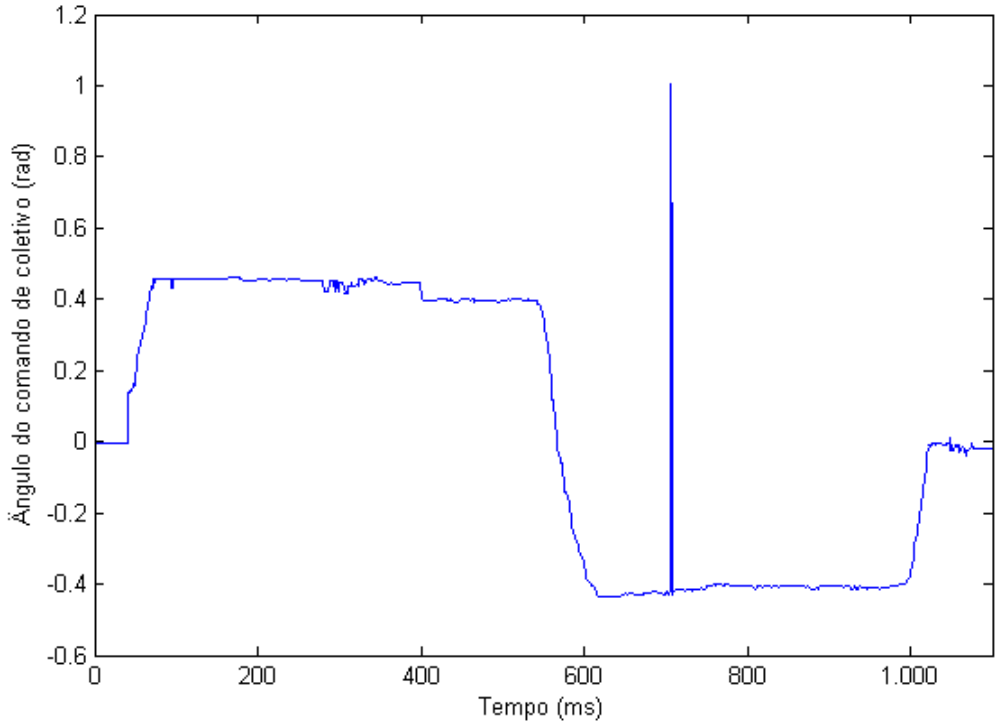

Figura 16 – Ângulo do comando de coletivo adquirido pelo sistema de detecção de movimento.

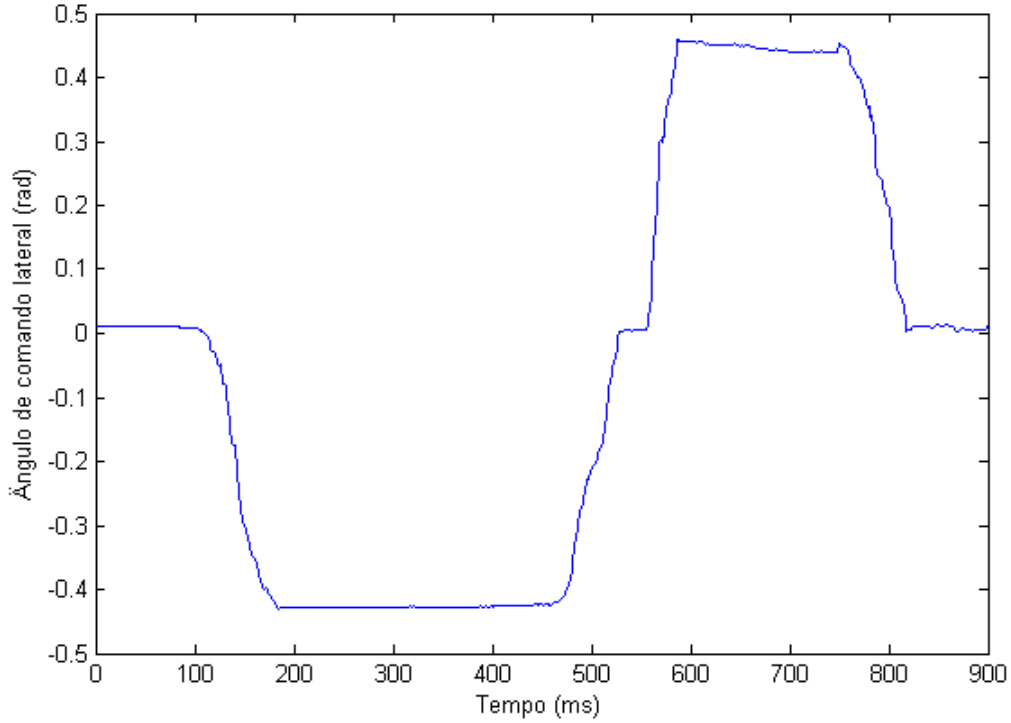

Figura 17 – Ângulo do comando lateral adquirido pelo sistema de detecção de movimento.

4.0 Comunicação do Computador com o Rádio Controle

O controle futuro do helicóptero precisará de um meio de comunicação entre o computador e os servomotores, no modelo de radio controle atual a comunicação é feita através do protocolo digital DSMX. Essa comunicação é sem fio na frequência de 2.4 GHz, no entanto esses radio controle possuem uma configuração de treino que possibilita que ao ser acoplado com outro rádio controle compatível, pode-se utilizar apenas o módulo de comunicação do rádio principal e deixar com que o rádio *slave* dê todos os comandos para o helicóptero.

Com isso pode-se emular um circuito que ao ser conectado ao computador faz com que o rádio controle principal veja o computador como um radio controle, possibilitando o computador controlar o helicóptero.

4.1 Circuito PCTX

Este circuito conversor é chamado de PCTX (Figura 18), o qual liga-se ao rádio principal por meio de um cabo com conector do tipo P2 mono, e liga-se ao computador através da interface USB. No circuito existe um microcontrolador do modelo *PIC18F4550* que converte o sinal enviado pelo computador em um sinal serial que é enviado ao rádio com as informações de cada canal.

A confecção deste circuito foi executada em laboratório (Figura 19), a lista de componentes pode ser visto na Tabela 3.

4.2 Programa PCTX

A comunicação entre a placa PCTX e o computador é feita através de um programa escrito em *Microsoft Visual C++*. Após a identificação do controlador HID pelo computador, ele envia a cada *50 ms* uma mensagem de 18 bites contendo, respectivamente, o resto da divisão por 256 do valor contido no canal, e a parte inteira da divisão por 256 do valor do canal. Valor que deve ser entre 0 e 1023.

O código do programa desenvolvido para a comunicação entre o PCTX e o computador pode ser visto no Anexo B. A partir do código desenvolvido, da identificação do modelo e desenvolvimento do controlador, o controle do helicóptero pode ser feito como indica a Figura 20.

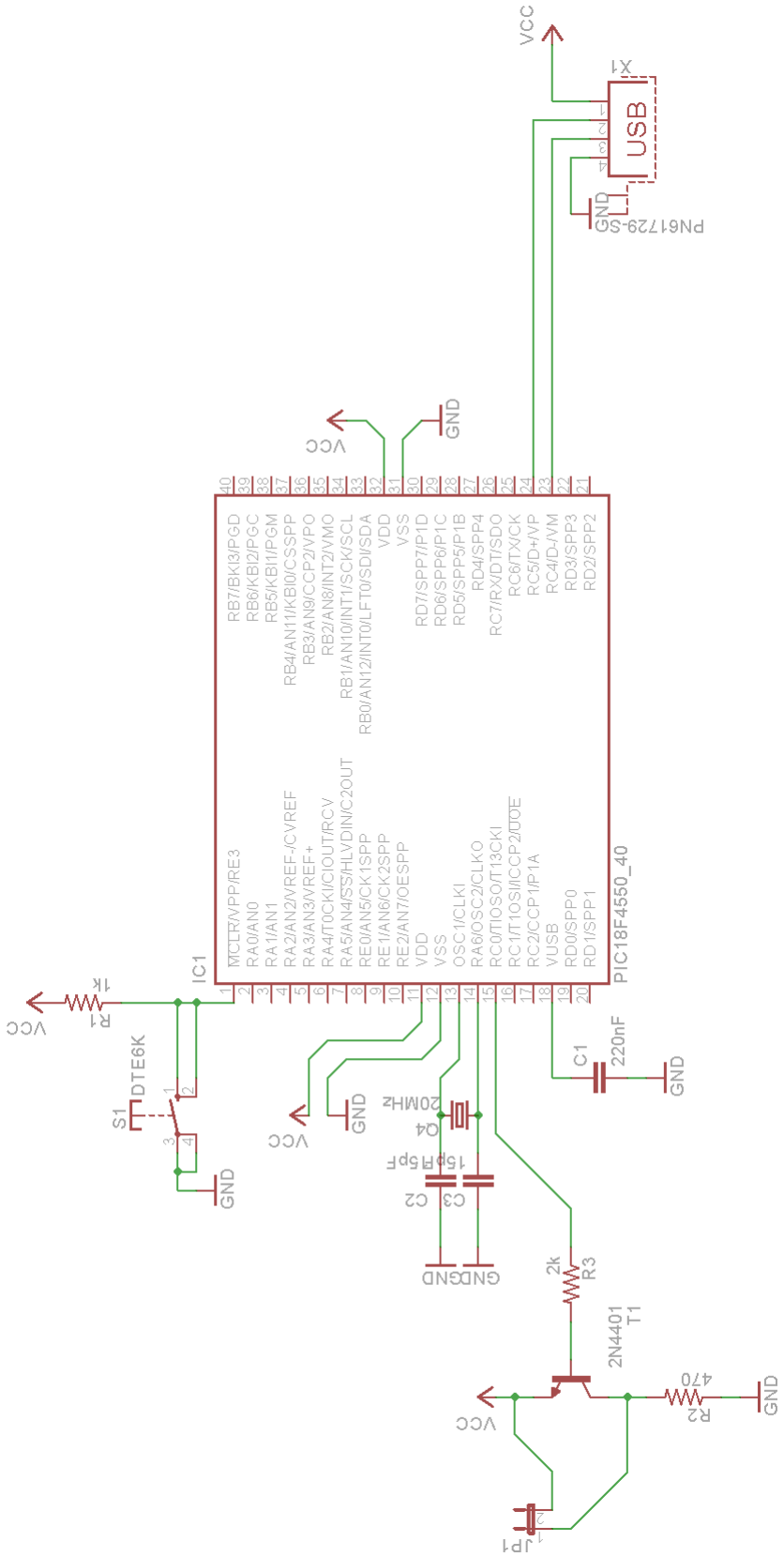

Figura 18 – Circuito do PCTX.

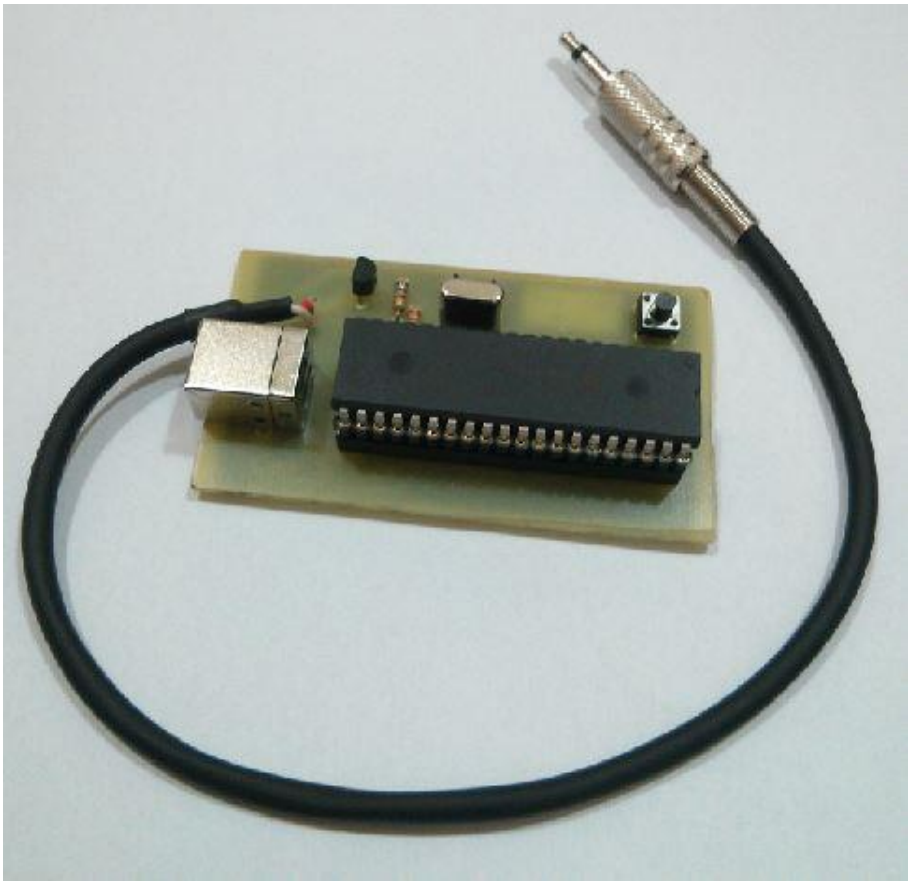

Figura 19 – Placa do circuito PCTX confeccionada no LASI.

Tabela 3 – Componentes do circuito PCTX

Componente)	**Quantidade**
Resistor 1 kΩ	1
Resistor 2,2 kΩ	1
Resistor 560 Ω	1
Capacitor 15 pF	2
Capacitor 220 nF	1
Transistor Bipolar NPN 2N4401	1
Cristal 20 kHz	1
Microprocessador *Microchip PIC18F4550*	1
Conector P2 macho	1
Conector USB fêmea	1
Chave Tact Switch	1

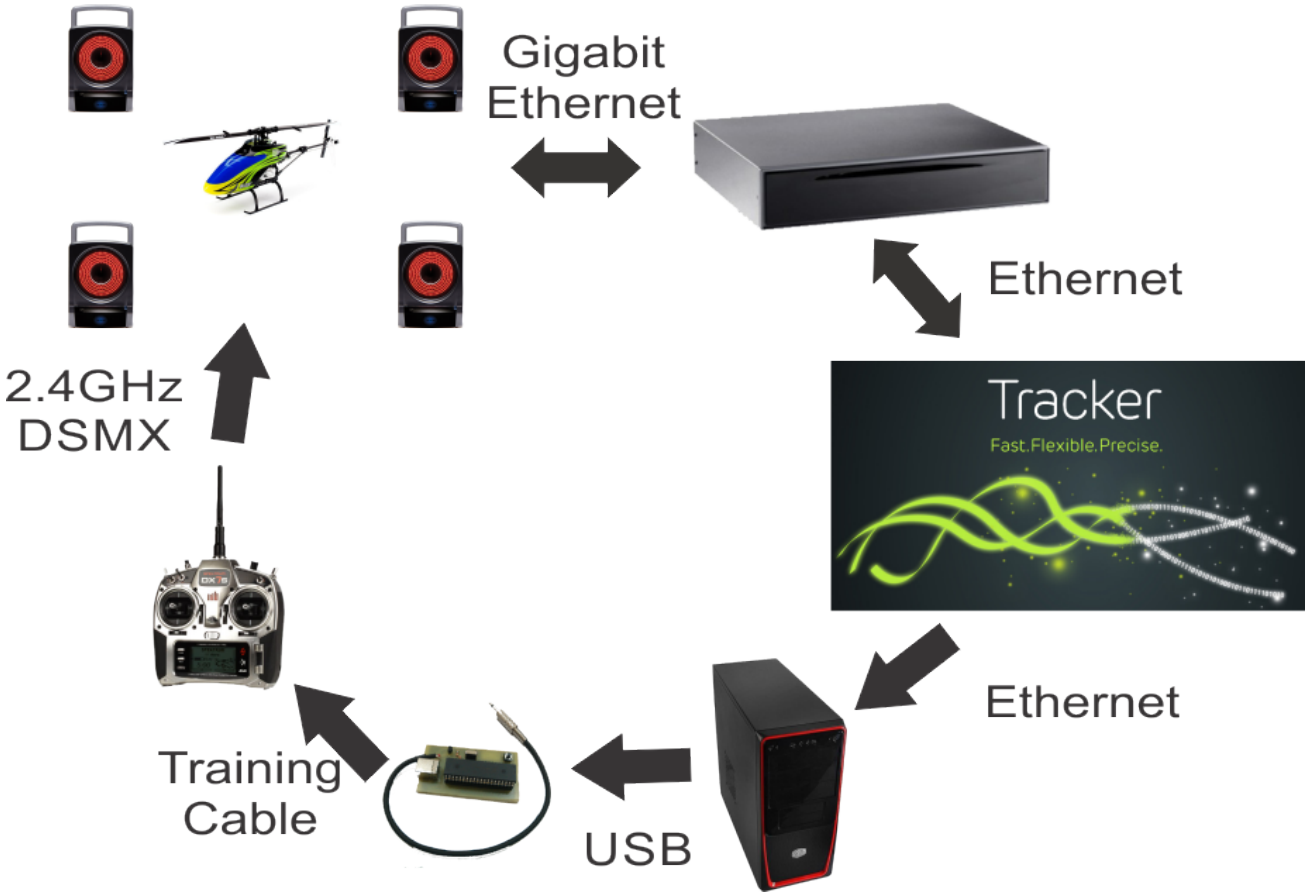

Figura 20 – Diagrama de controle utilizando o sistema de detecção de movimento.

Conclusão

Embora existiram algumas mudanças ao decorrer da execução do projeto, principalmente com a não utilização da base 3D e com a aquisição e utilização das câmeras. O trabalho consistiu em uma instrumentação tanto de *software* quanto de *hardware* para possibilitar a identificação e, futuramente, o controle de um ou mais modelos de helicóptero utilizando o conjunto de câmeras e computadores.

Além da familiarização com a utilização das câmeras, que embora tenham uma autonomia, algumas configurações como de foco e de utilização do *Tracker* são necessárias para que a utilização do conjuto seja o mais efetivo e simples possível. Parte do trabalho desenvolvido foi importante para o entendimento do funcionamento destes equipamentos, fazendo com que futuramente os códigos desenvolvidos aqui sejam base para linhas de pesquisa que utilizem o sistema de detecção de movimento utilizado neste trabalho.

A modelagem já citada neste trabalho visa a identificação do modelo de espaço de estados do helicóptero utilizado através dos dados adquiridos com o trabalho desenvolvido neste projeto, com as técnicas de aquisição e tratamento de dados desenvolvidas neste trabalho é possível em um futuro próximo a identificação dos parâmetros de modelagem para o desenvolvimento de um controlador para um VANT helicóptero.

Para complementar esse trabalho, e concluir a parte de aquisição dos sinais de entrada, a aquisição do comandos de entrada deverá ser tratada com as curvas de compensação configurada no próprio rádio controle, esta curva depende de como o rádio será configurado, assim para cada trabalho subsequente está curva deverá ser anotada e a transformação dos ângulos de entrada captados e os de entrada no sistema serão únicos para cada configuração.

O resultado deste trabalho contribui diretamente para o desenvolvimento de trabalhos de pesquisa de mestrado e doutorado de outros alunos do Laboratório de Sistemas Inteligentes (LASI).

Referências

FOGH M., M. T. H. M. E. P. R. S. J. M. *Autonomous Helicopter*. [S.l.], 2004. Citado 2 vezes nas páginas 22 e 23.

HEHN, R. D. M. Quadrocopter trajectory generation and control. In: *18th World Congress The International Federation of Automatic Control Milano (Italy)*. [S.l.: s.n.], 2011. Citado na página 21.

INOUE, E. S. Controle robusto descentralizado de movimentos coordenados de robôs heterogêneos. *Tese (Doutorado). Escola de Engenharia de São Carlos da Universidade de São Paulo*, 2011. Citado na página 13.

MELLINGER, D. Trajectory generation and control for quadrotors. *A Dissertation in Mechanical Engineering and Applied Mechanics*, 2012. Citado na página 21.

METTLER B., T. M. B. K. T. System identification of small-size unmanned helicopter dynamics. In: *American Helicopter Society, 55th Forum, Montreal, Quebec, Canada*. [S.l.: s.n.], 1999. Citado na página 23.

NAIDOO R. STOPFORTH, G. B. Y. Quad-rotor unmanned aerial vehicle helicopter modelling & control. *Int J Adv Robotic Sy, 2011, Vol. 8, No. 4, 139-149*, 2011. Citado na página 21.

RICH, M. Master of science thesis, iowa state university. *Int J Adv Robotic Sy, 2011, Vol. 8, No. 4, 139-149*, 2012. Citado na página 21.

VICON. *Vicon DataStrem SDK Developer's Manual*. [S.l.], 2013. Citado na página 29.

YI, J. K.; SASTRY, S. S. Output tracking control design of a helicopter model based on approximate linearization. In: *IEEE Conference on Decision and Control, Tampo, FL, USA*. [S.l.: s.n.], 1998. Citado na página 22.

Anexos

A.0 *Script* desenvolvido para o *software MATLAB* para aquisição dos dados das câmeras

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
%                                                                       %
% Jose Renato Bulhoes Vicente                                           %
% Departamento de Engenharia Eletrica e de Computacao                   %
% Escola de Engenharia de Sao Carlos                                    %
% Universidade de Sao Paulo                                             %
% Abril de 2014                                                         %
%                                                                       %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
clc
% Program options
TransmitMulticast = false;

% A dialog to stop the loop
MessageBox = msgbox( 'Stop DataStream Client', 'Vicon DataStream SDK' );

% Load the SDK
fprintf( 'Loading SDK...' );
Client.LoadViconDataStreamSDK();
fprintf( 'done\n' );

% Program options
HostName = '127.0.0.1:801';

% Make a new client
MyClient = Client();

% Connect to a server
fprintf( 'Connecting to %s ...', HostName );
while ~MyClient.IsConnected().Connected
  % Direct connection
  MyClient.Connect( HostName );


  fprintf( '.' );
end
fprintf( '\n' );

% Enable some different data types
MyClient.EnableSegmentData();
```

```
MyClient.EnableMarkerData();
MyClient.EnableUnlabeledMarkerData();
MyClient.EnableDeviceData();

fprintf( 'Segment Data Enabled: %s\n',          AdaptBool(
    MyClient.IsSegmentDataEnabled().Enabled ) );
fprintf( 'Marker Data Enabled: %s\n',           AdaptBool(
    MyClient.IsMarkerDataEnabled().Enabled ) );
fprintf( 'Unlabeled Marker Data Enabled: %s\n', AdaptBool(
    MyClient.IsUnlabeledMarkerDataEnabled().Enabled ) );
fprintf( 'Device Data Enabled: %s\n',           AdaptBool(
    MyClient.IsDeviceDataEnabled().Enabled ) );

% Set the streaming mode
MyClient.SetStreamMode( StreamMode.ClientPull );

% Set the global up axis
MyClient.SetAxisMapping( Direction.Forward, ...
                         Direction.Left,    ...
                         Direction.Up );    % Z-up

Output_GetAxisMapping = MyClient.GetAxisMapping();
fprintf( 'Axis Mapping: X-%s Y-%s Z-%s\n', Output_GetAxisMapping.XAxis.ToString(),
    ...
                                    Output_GetAxisMapping.YAxis.ToString(), ...
                                    Output_GetAxisMapping.ZAxis.ToString() );


% Discover the version number
Output_GetVersion = MyClient.GetVersion();
fprintf( 'Version: %d.%d.%d\n', Output_GetVersion.Major, ...
                                Output_GetVersion.Minor, ...
                                Output_GetVersion.Point );

if TransmitMulticast
  MyClient.StartTransmittingMulticast( 'localhost', '224.0.0.0' );
end

% New frame structure
   Frame_2past = struct('number', {}, 'rate', {}, 'timecode', zeros(9), 'latency',
       {});

% New Subject

SubjectCount = MyClient.GetSubjectCount().SubjectCount;
Subject = struct([1:SubjectCount]);
for i=1:SubjectCount;
    Subject(i)  = struct('number', {}, 'name', {'nome'}, 'trans', zeros(3), 'helical
        ', zeros(3), 'matrix', zeros(9), 'quater', zeros(4), 'euler', zeros(3), '
        markers', {});

end
```

```
% Loop until the message box is dismissed
fprintf( '\n Running...' );
while ishandle( MessageBox )
  drawnow;

  % Get a frame
%   fprintf( '\n Waiting for new frame...' );
  while MyClient.GetFrame().Result.Value ~= Result.Success
    fprintf( '.' );
  end% while

  % Get the frame number
  Output_GetFrameNumber = MyClient.GetFrameNumber();
  Frame_2past(1).number = Output_GetFrameNumber.FrameNumber;

  % Get the frame rate
  Output_GetFrameRate = MyClient.GetFrameRate();
  Frame_2past(1).rate = Output_GetFrameRate.FrameRateHz;

  % Get the timecode
  Output_GetTimecode = MyClient.GetTimecode();
  Frame_2past(1).timecode(1) = Output_GetTimecode.Hours;
  Frame_2past(1).timecode(2) = Output_GetTimecode.Minutes;
  Frame_2past(1).timecode(3) = Output_GetTimecode.Seconds;
  Frame_2past(1).timecode(4) = Output_GetTimecode.Frames;
  Frame_2past(1).timecode(5) = Output_GetTimecode.SubFrame;
  Frame_2past(1).timecode(7) = Output_GetTimecode.Standard.Value;
  Frame_2past(1).timecode(8) = Output_GetTimecode.SubFramesPerFrame;
  Frame_2past(1).timecode(9) = Output_GetTimecode.UserBits;


  % Get the latency
  Frame_2past(1).latency = MyClient.GetLatencyTotal().Total;


  % Count the number of subjects
  SubjectCount = MyClient.GetSubjectCount().SubjectCount;

  for SubjectIndex = 1:SubjectCount
    Subject(SubjectIndex).number = SubjectIndex - 1;

    % Get the subject name
    SubjectName = MyClient.GetSubjectName( SubjectIndex ).SubjectName;
    Subject(SubjectIndex).name = SubjectName;

    % Get the root segment
    RootSegment = MyClient.GetSubjectRootSegmentName( SubjectName ).SegmentName;

    % Count the number of segments
    SegmentCount = MyClient.GetSegmentCount( SubjectName ).SegmentCount;
    for SegmentIndex = 1:SegmentCount
```

```
% Get the segment name
SegmentName = MyClient.GetSegmentName( SubjectName, SegmentIndex ).SegmentName
    ;

% Get the segment parent
SegmentParentName = MyClient.GetSegmentParentName( SubjectName, SegmentName )
    .SegmentName;

% Get the segment's children
ChildCount = MyClient.GetSegmentChildCount( SubjectName, SegmentName )
    .SegmentCount;

% Get the global segment translation
Output_GetSegmentGlobalTranslation = MyClient.GetSegmentGlobalTranslation(
    SubjectName, SegmentName );
for i=1:3
  Subject(SubjectIndex).trans(i) =
      Output_GetSegmentGlobalTranslation.Translation( i );
end

% Get the global segment rotation in helical co-ordinates
Output_GetSegmentGlobalRotationHelical =
    MyClient.GetSegmentGlobalRotationHelical( SubjectName, SegmentName );
for i=1:3
  Subject(SubjectIndex).helical(i) =
      Output_GetSegmentGlobalRotationHelical.Rotation( i );
end



% Get the global segment rotation as a matrix
Output_GetSegmentGlobalRotationMatrix =
    MyClient.GetSegmentGlobalRotationMatrix( SubjectName, SegmentName );
for i=1:9
  Subject(SubjectIndex).matrix(i) =
      Output_GetSegmentGlobalRotationMatrix.Rotation( i );
end



% Get the global segment rotation in quaternion co-ordinates
Output_GetSegmentGlobalRotationQuaternion =
    MyClient.GetSegmentGlobalRotationQuaternion( SubjectName, SegmentName );
for i=1:4
  Subject(SubjectIndex).quater(i) =
      Output_GetSegmentGlobalRotationQuaternion.Rotation( i );
end


% Get the global segment rotation in EulerXYZ co-ordinates
Output_GetSegmentGlobalRotationEulerXYZ =
    MyClient.GetSegmentGlobalRotationEulerXYZ( SubjectName, SegmentName );
```

```
      for i=1:3
        Subject(SubjectIndex).euler(i) =
            Output_GetSegmentGlobalRotationEulerXYZ.Rotation( i );
      end


    end% SegmentIndex

    % Count the number of markers
    MarkerCount = MyClient.GetMarkerCount( SubjectName ).MarkerCount;
    Subject(SubjectIndex).markers_num = MarkerCount;
    Subject(SubjectIndex).markers = struct('name', {'', '', '', ''}, 'values', zeros
        (3));

    for MarkerIndex = 1:MarkerCount
      % Get the marker name
      MarkerName = MyClient.GetMarkerName( SubjectName, MarkerIndex ).MarkerName;

      % Get the marker parent
      MarkerParentName = MyClient.GetMarkerParentName( SubjectName, MarkerName )
          .SegmentName;

      Subject(SubjectIndex).markers(MarkerIndex).name = MarkerName;

      % Get the global marker translation
      Output_GetMarkerGlobalTranslation = MyClient.GetMarkerGlobalTranslation(
          SubjectName, MarkerName );
      for i=1:3
        Subject(SubjectIndex).markers(MarkerIndex).values(i) =
            Output_GetMarkerGlobalTranslation.Translation( i );
      end

     end% MarkerIndex

  end% SubjectIndex


end% while true


if TransmitMulticast
  MyClient.StopTransmittingMulticast();
end

% Disconnect and dispose
MyClient.Disconnect();

% Unload the SDK
fprintf( 'Unloading SDK...' );
Client.UnloadViconDataStreamSDK();
fprintf( 'done\n' );
clearvars -except Subject Frame_2past
```

B.0 Código desenvolvido em *Visual C++ 2012* para o conversor PCTX

```
#pragma once
#include "controller.h"
#undef GetObject

namespace controllerGUI
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    public __gc class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
        }

    protected:
        void Dispose(Boolean disposing)
        {
            if (disposing && components)
            {
                components->Dispose();
            }
            __super::Dispose(disposing);
        }

    private: System::ComponentModel::IContainer *  components;
    private: System::Windows::Forms::Label *  label20;
    private: System::Windows::Forms::Label *  label21;
    private: System::Windows::Forms::Label *  label22;
    private: System::Windows::Forms::Timer *  timer1;
    private: int canal1;
    private: int canal2;
    private: int canal3;
    private: int canal4;
```

```
    private: int canal5;
    private: int canal6;
    private: int canal7;
    private: int canal8;
    private: int canal9;

    private: controller *c;

        void InitializeComponent(void)
        {
            this->components = (new System::ComponentModel::Container());
            System::ComponentModel::ComponentResourceManager*  resources = (new
                System::ComponentModel::ComponentResourceManager(__typeof(Form1)));
            this->timer1 = (new System::Windows::Forms::Timer(this->components));
            this->label20 = (new System::Windows::Forms::Label());
            this->label21 = (new System::Windows::Forms::Label());
            this->label22 = (new System::Windows::Forms::Label());
            this->SuspendLayout();

            this->timer1->Enabled = true;
            this->timer1->Interval = 50;
            this->timer1->Tick += new System::EventHandler(this, &Form1::timer1_Tick
                );

            this->label20->Image = (__try_cast<System::Drawing::Image*  >(resources
                ->GetObject(S"label20.Image")));
            this->label20->Location = System::Drawing::Point(199, 240);
            this->label20->Name = S"label20";
            this->label20->Size = System::Drawing::Size(16, 16);
            this->label20->TabIndex = 35;

            this->label21->Image = (__try_cast<System::Drawing::Image*  >(resources
                ->GetObject(S"label21.Image")));
            this->label21->Location = System::Drawing::Point(199, 240);
            this->label21->Name = S"label21";
            this->label21->Size = System::Drawing::Size(16, 16);
            this->label21->TabIndex = 36;

            this->label22->Location = System::Drawing::Point(231, 242);
            this->label22->Name = S"label22";
            this->label22->Size = System::Drawing::Size(128, 13);
            this->label22->TabIndex = 37;
            this->label22->Text = S"Not Connected";

            this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
            this->ClientSize = System::Drawing::Size(424, 267);
            this->Controls->Add(this->label22);
            this->Controls->Add(this->label21);
            this->Controls->Add(this->label20);
            this->Name = S"Form1";
            this->Text = S"Control PCTX Spektrum DX7S";
            this->Load += new System::EventHandler(this, &Form1::Form1_Load);
```

```
            this->ResumeLayout(false);

        }

    private: System::Void Form1_Load(System::Object *  sender, System::EventArgs *
        e)
             {
                 c = new controller();

                 if(c->connected == true) {
                    label22->Text = "Connected";


                    canal1 = 100;
                    canal2 = 200;
                    canal3 = 300;
                    canal4 = 400;
                    canal5 = 500;
                    canal6 = 600;
                    canal7 = 700;
                    canal8 = 800;
                    canal9 = 900;

                    this->timer1->Enabled = true;
                }
             }

private: bool update()
         {
             //Code for control

             return c->send(canal1%256,canal1/256,canal2%256,canal2/256,canal3%256,
                 canal3/256,canal4%256,canal4/256,canal5%256,canal5/256,canal6%256,
                 canal6/256,canal7%256,canal7/256,canal8%256,canal8/256,canal9%256,
                 canal9/256);
         }



private: System::Void timer1_Tick(System::Object *  sender, System::EventArgs *  e)
         {
                if(update()){
                    label21->Visible = true;
                    label22->Text = "Connected";
                }
                else {
                    if(c->connect()){
                        label21->Visible = true;
                        label22->Text = "Connected";
                    }
                    else {
                        label21->Visible = false;
```

```
                        label22->Text = "Not Connected";
                    }
                }
            }
};
}
```