UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

Relatório Final

Trabalho de Conclusão de Curso

Manipulação e Controle de Sistemas Dinâmicos através de Interface Visual

Autor: **Gabriel Daré**

Orientador: **Prof. Dr. André Ricardo Fioravanti**

Campinas, Novembro de 2014

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

Relatório Final
Trabalho de Conclusão de Curso

Manipulação e Controle de Sistemas Dinâmicos através de Interface Visual

Autor: **Gabriel Daré**
Orientador: **Prof. Dr. André Ricardo Fioravanti**

Curso: Engenharia de Controle e Automação

Trabalho de Conclusão de Curso, apresentado à Comissão de Graduação da Faculdade de Engenharia Mecânica, como requisito para a obtenção do título de Engenheiro de Automação e Controle.

Campinas, 2014
S.P. – Brasil

Agradecimentos

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas às quais presto minha homenagem:

Sr. Dr. André Ricardo Fioravanti pela excelente revisão da formatação e da gramática deste documento, além de sua grande ajuda técnica e de incentivo.

À meus familiares, amigos e, principalmente, namorada que me auxiliou e me deu todo o suporte para a conclusão deste trabalho.

Índice

Resumo

DARÉ, Gabriel, *Manipulação e Controle de Sistemas Dinâmicos através de Interface Visual*, Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Trabalho de Conclusão de Curso, (2014).

O objetivo deste Trabalho de Graduação I é fazer a integração completa entre um sistema de Interface Visual, como a Microsoft Kinect, e um sistema dinâmico. Desta forma, através do reconhecimento de posições e gestos, desejamos manipular ou mesmo controlar esta planta apenas utilizando informações provenientes de imagens.

Nesta primeira parte do projeto, iniciaremos os estudos deste problema. Como primeiro passo, devemos conhecer o sensor visual em si, ou seja, aprender a receber e interpretar os dados que a Microsoft Kinect é capaz de fornecer. Para isso, será necessário tanto o estudo da API quanto a instalação dos componentes e ambientes computacionais. Em seguida, estudaremos alguns algoritmos capazes de interpretar gestos humanos, de forma que seja possível enviar ao equipamento alguns comandos simples, como inicializar, encerrar ou mudar o ponto de operação.

Uma vez concluída esta etapa, que permitirá sobretudo uma melhor compreensão do funcionamento do sensor e a interligação dele com a planta física, pretendemos, na segunda etapa do projeto, controlar o equipamento utilizando apenas as informações fornecidas pelo equipamento visual. Duas metodologias serão exploradas. Na primeira, as informações dos *encoders* serão descartadas, e apenas os dados obtidos visualmente serão utilizados. Este é um problema extremamente complicado, envolvendo técnicas avançadas de visão computacional e controle com atrasos. Outra possibilidade, talvez mais simples, seja captar e interpretar a ação de um humano para controlar, virtualmente, o equipamento. A factibilidade deste problema depende, no entanto, em grande parte da precisão do equipamento, informação que só obteremos durante a execução da primeira parte do projeto.

Palavras Chave: Kinect, Interface Visual, Sensor Visual.

Lista de Figuras

Lista de Equações

Abreviações e Terminologia

API *Application programming interfaces*, são interfaces que permitem diferentes *softwares* se comunicarem;

C++ Linguagem de programação orientada a objeto usada para desenvolver os programas deste projeto;

IHM Interface Homem-Máquina é o *software* e/ou *hardware* que permite o usuário interagir com a máquina;

Matlab Linguagem de programação especializado em matrizes e matemática para o uso nas universidades e industrias para modelagem matemática;

RGB Referência ao tipo de sensor da câmera em *Red Green Blue* que é parte do sistema de sensor da Microsoft Kinect;

SDK *Software Development Kit* permite ao programador utilizar todas as bibliotecas, funções ou definições disponíveis para o Microsoft Kinect;

VGA *Video Graphics Array* que se refere à resolução de 640x480 pixels disponíveis na Câmera do Microsoft Kinect;

Capítulo 1

Introdução

Com o avanço na capacidade computacional de sistemas embarcados, o que permitiu a recente queda de preço e maior disseminação de sistemas complexos de interface visual, tornou-se possível a manipulação virtual de objetos 3D, como pode ser visto na tese de A. Ladzinski[1]. Uma área que pode se beneficiar com estas técnicas são as tele-operações robóticas, que possui uma vasta quantidade de aplicações interessantes, seja intervenções cirúrgicas à distância ou operações militares. Porém, apesar do grande interesse prático, existe ainda uma grande quantidade de complexos desafios acadêmicos para a disseminação do uso destes dispositivos, sobretudo na área de controle e automação.

Dessa forma, percebe-se que precisamos cada vez mais criar uma relação mais natural com o mundo da tecnologia. Essa necessidade se resume na finalidade do compartilhamento de informação com o espaço físico e, não apenas, em enviar, receber, armazenar e computar informações, como diz Felipe Santos[2].

Essa interação pode ser feita justamente através, por exemplo, do objeto de estudo desse Trabalho de Conclusão de Curso, o Microsoft Kinect, o qual iremos utilizar todas as especialidades que possui este equipamento para, em uma primeira opção como Trabalho de conclusão de curso II, utiliza-lo como um sensor que fecharia a malha de controle da planta, neste caso o pêndulo de Furuta. Uma segunda possibilidade, para este trabalho futuro, seria o controle de nossa planta através da interpretação visual dos movimentos de um ser humano.

Como dito anteriormente, o objeto de estudo deste Trabalho de Conclusão de Curso será os estudos preliminares para o estudo supracitado. Este, se dará da seguinte forma:

---

[1] A. Ladzinski, Development of 3D Image Manipulation Software Using the Microsoft Kinect, Murdoch University, 2013.

[2] F. Santos, Aplicação para reconhecimento de Gestos, baseada em OPENNI, Faculdade Anhanguera de Belo Horizonte, 2013.

um inicio focado em conhecer todos os recursos que o sensor visual pode fornecer e auxiliar. Uma vez compreendido, um segundo passo a ser dado é receber e interpretar os dados fornecidos pelo Kinect, o qual se faz necessário instalação e ambientação de softwares adequados, como o SDK, que fornecem APIs que ajudam nessas interpretações. Uma vez ambientado com estes softwares, linguagens e bibliotecas, passamos para uma terceira etapa que seria estudar, ou desenvolver, algoritmos que possam interpretar comandos gestuais e/ou sonoros, de forma que seja possível realizar algumas tarefas pré-determinadas em nossa planta.

Capítulo 2

Revisão Bibliográfica

Para este capitulo, será apresentado todos os recursos para a implementação do projeto como o Microsoft Kinect, as plataformas de desenvolvimento das programações SDK e Microsoft Visual Studio e a planta de trabalho, o pêndulo de furuta.

2.1 Microsoft Kinect

O sensor de movimento, Microsoft Kinect, é capaz é capaz de interpretar movimentos específicos com a finalidade de controlar dispositivos eletrônicos. Com o auxílio de um sensor infravermelho, chamado *Light Coding*, o Kinect é capaz de reconstruir o ambiente em 3D.

O sistema de *software* do Kinect foi desenvolvido por uma subsidiária da Microsoft chamada *Rare*. Já a tecnologia da câmera foi desenvolvida com a empresa israelita, Prime Sense, juntamente com o sensor infravermelho.

Como curiosidade, um dos pesquisadores responsáveis pelo desenvolvimento do sensor é o brasileiro Alex Kipman. Este, tem como cidade de nascimento Natal-RN que, devido a este fato, foi dado como primeiro nome ao sensor de "*Project Natal*" em 2010.

Figura 2.1. Microsoft Kinect. Fonte: Wikipédia

O módulo do Kinect é equipado com 5 recursos principais:

a) Câmera RGB;
b) Sensor Infravermelho que permite ao sensor medir a distancia dos objetos;

c) Microfone capazes de identificar mais de um usuário bem como a direção que o mesmo se encontra;
d) Processador e software;

Figura 2.2. Recursos do Microsoft Kinect. Fonte: http://www.winbeta.org

Com estes recursos, o sensor possui como características técnicas: campo de visão de 57° horizontalmente e 43° verticalmente; alcance especial 640x480 (VGA); alcance de profundidade de 0,8m a 3,5m.

2.2 SDK e Microsoft Visual Studio

Com a popularização do Kinect, em 2011 a Microsoft desenvolveu um *software* de desenvolvimento, chamado SDK, permitindo o sensor ser usado como um produto não comercial, principalmente para a aplicação de visão computacional.

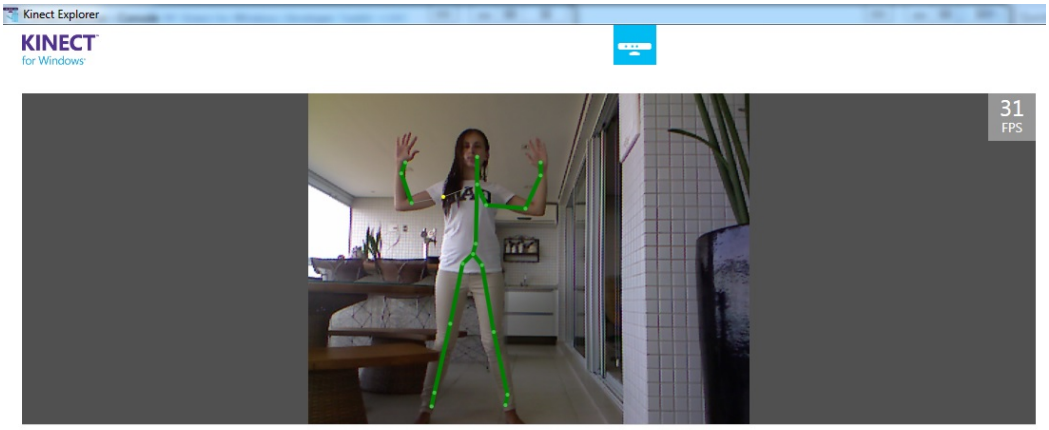

Figura 2.3. Plataforma SDK, com recurso Skeleton. Fonte: G. Daré, 2014

Existem hoje três *APIs* disponíveis para a programação do Kinect – Microsoft SDK, OpenNI e OpenKinect. A diferença entres os três vai desde as plataformas de trabalho, alguns podem ser trabalhados em Apple e Windows, outros apenas com Windows. Além disso, a precisão da distância varia entre eles tanto para perto como longe. Estes programas que permitem a visualização, teste e acesso aos recursos disponibilizados pelo sensor Kinect, como *skeleton*, figura 2.3, *depth image*, figura 2.4.

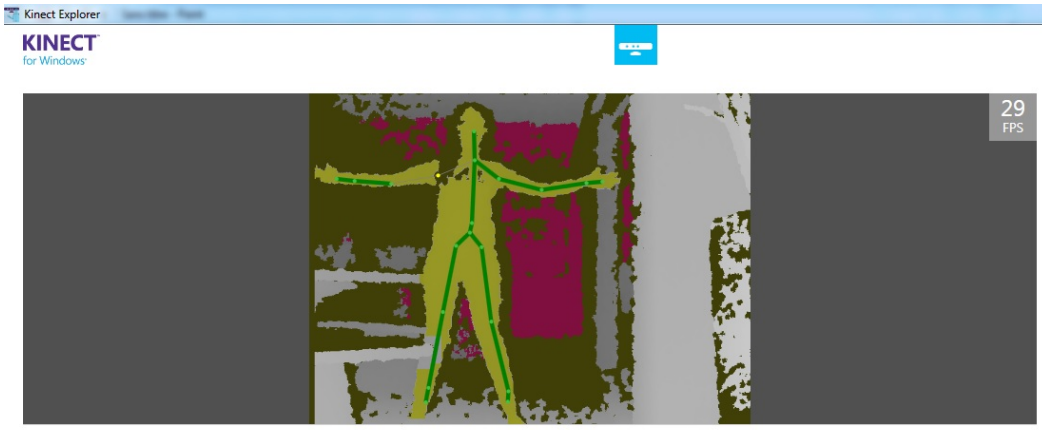

Figura 2.4. Plataforma SDK, com recurso *Depth Image*. Fonte: G. Daré, 2014

Para que você consiga criar programas com esses recursos como *Depth Image, Skeleton* entre outros, é utilizado o Visual Studio, figura 2.5, que com as API fazem a comunicação entre o lido pelo sensor, e o que se deseja com a programação no Visual Studio.

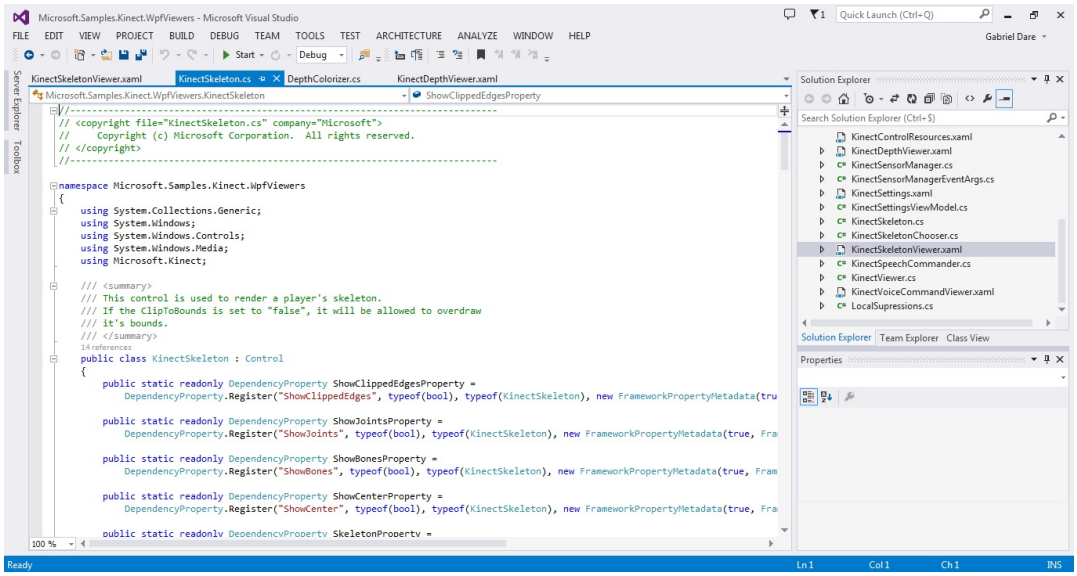

Figura 2.5. Plataforma Microsoft Visual Studio. Fonte: G. Daré, 2014

2.3 Pêndulo de Furuta

A planta a ser utilizada na segunda parte do nosso projeto, possivelmente no Trabalho de Conclusão de Curso II, será o Pêndulo de Furuta (Figura 2.3). Este pêndulo consiste em um braço conduzido um motor de rotação na horizontal. O objetivo final para o estudo será em utilizar o Microsoft Kinect no lugar dos *encoders* e fechar a malha de controle do pêndulo.

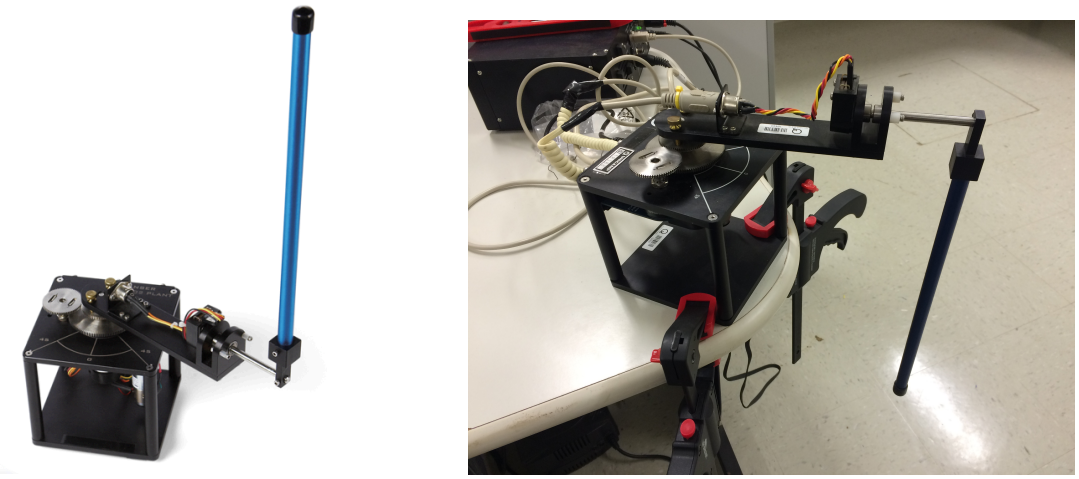

Figura 2.6. Pêndulo de Furuta. Fonte: Workbook of Quaser, 2011 e G. Daré, 2014

O Pêndulo de Furuta foi utilizado por muitos autores para demonstrar leis de controles linear e não lineares. Principalmente, no controle com o pêndulo invertido, não linear, será de extrema importância a utilização do sensor Kinect para fechar a malha.

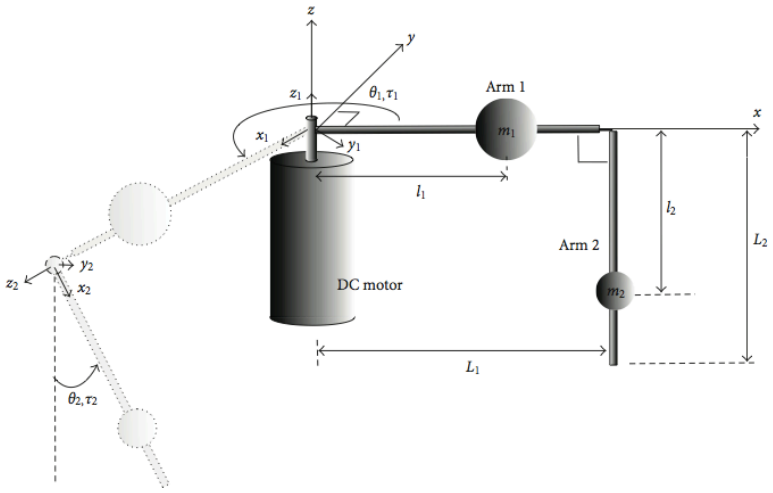

Figura 2.7. Esquemático do Pêndulo de Furuta. Fonte: B. Cazzolato & Z. Prime, On the Dynamics of the Furuta Pendulum, 2011.

B. Cazzolato & Z. Prime demonstraram em seu artigo de pesquisa toda a dinâmica não linear, parte do pêndulo invertido, utilizando dois métodos: Euler-Lagrange e Newton-Euler. Eles chegaram na seguinte equação:

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \dot{i} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ A_{31} & A_{32} & A_{33} & A_{34} & B_{31}K_m \\ A_{41} & A_{42} & A_{43} & A_{44} & B_{41}K_m \\ 0 & 0 & \frac{-K_m}{L_m} & 0 & \frac{-R_m}{L_m} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{L_m} \end{bmatrix} V + \begin{bmatrix} 0 \\ 0 \\ B_{32} \\ B_{42} \\ 0 \end{bmatrix} \tau_2.$$

Equação 2.1

Onde:

$$A_{31} = 0,$$

$$A_{32} = \frac{g m_2^2 l_2^2 L_1}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$A_{33} = \frac{-b_1 \hat{J}_2}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$A_{34} = \frac{-b_2 m_2 l_2 L_1}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$A_{41} = 0,$$

$$A_{42} = \frac{g m_2 l_2 \hat{J}_0}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$A_{43} = \frac{-b_1 m_2 l_2 L_1}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$A_{44} = \frac{-b_2 \hat{J}_0}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$B_{31} = \frac{\hat{J}_2}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$B_{41} = \frac{m_2 L_1 l_2}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$B_{32} = \frac{m_2 L_1 l_2}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)},$$

$$B_{42} = \frac{\hat{J}_0}{\left(\hat{J}_0 \hat{J}_2 - m_2^2 L_1^2 l_2^2\right)}.$$

Capítulo 3

Procedimento Experimental

Para este Trabalho de Conclusão de Curso I, teremos como objeto de trabalho a ambientação quanto aos *Softwares* de desenvolvimento, no caso o SDK e Visual Studio, bem como a implementação de fato do sensor Kinect com o reconhecimento de alguns movimentos definidos como: acionar e desligar a planta. Esta etapa está relacionada com uma terceira etapa que seria a comunicação do *software* criado juntamente com o sensor Kinect e a planta de trabalho. Uma vez que o sensor interpretando o movimento, vai atuar diretamente no sistema, pendulo de furuta.

3.1 Primeira Etapa

Nesta fase, segundo a explicação supracitada, foi concluída com a instalação dos *softwares* de desenvolvimento SDK e Visual Studio, juntamente com as API e bibliotecas já existentes para a programação do Kinect.

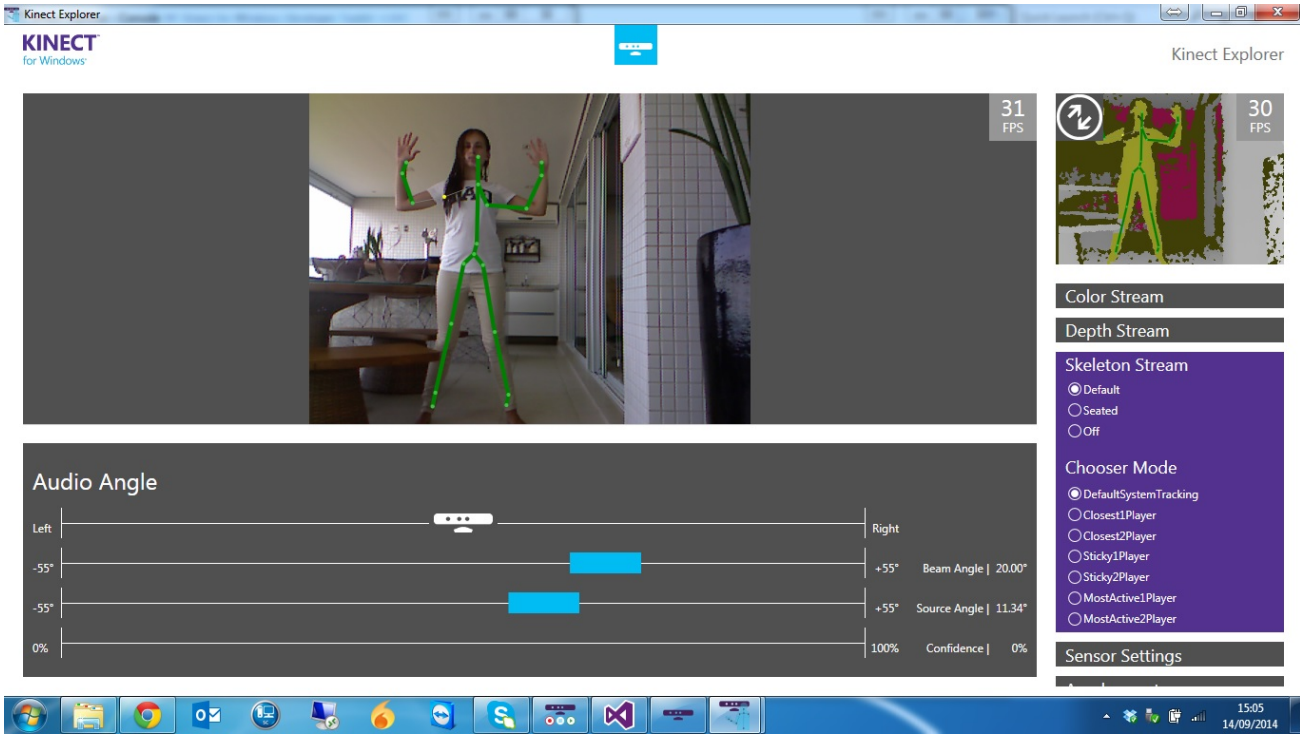

Figura 3.1. Kinect Explorer com reconhecimento de esqueleto. Fonte: G. Daré, 2014

Como forma de ambientação, foi possível observar sistemas básicos para o Kinect como o reconhecimento de esqueleto através da câmera, figura 2.8, bem como a

implementação da distancia através de cores, figura 2.9, com a implementação do infravermelho juntamente com a câmera. Estes são os primeiros passos para o reconhecimento de movimentos, uma vez que com a detecção das distâncias e do esqueleto de movimentação, basta agora criar um programa que interpreta os movimentos realizados e com isso executa alguma ação pré-estabelecida.

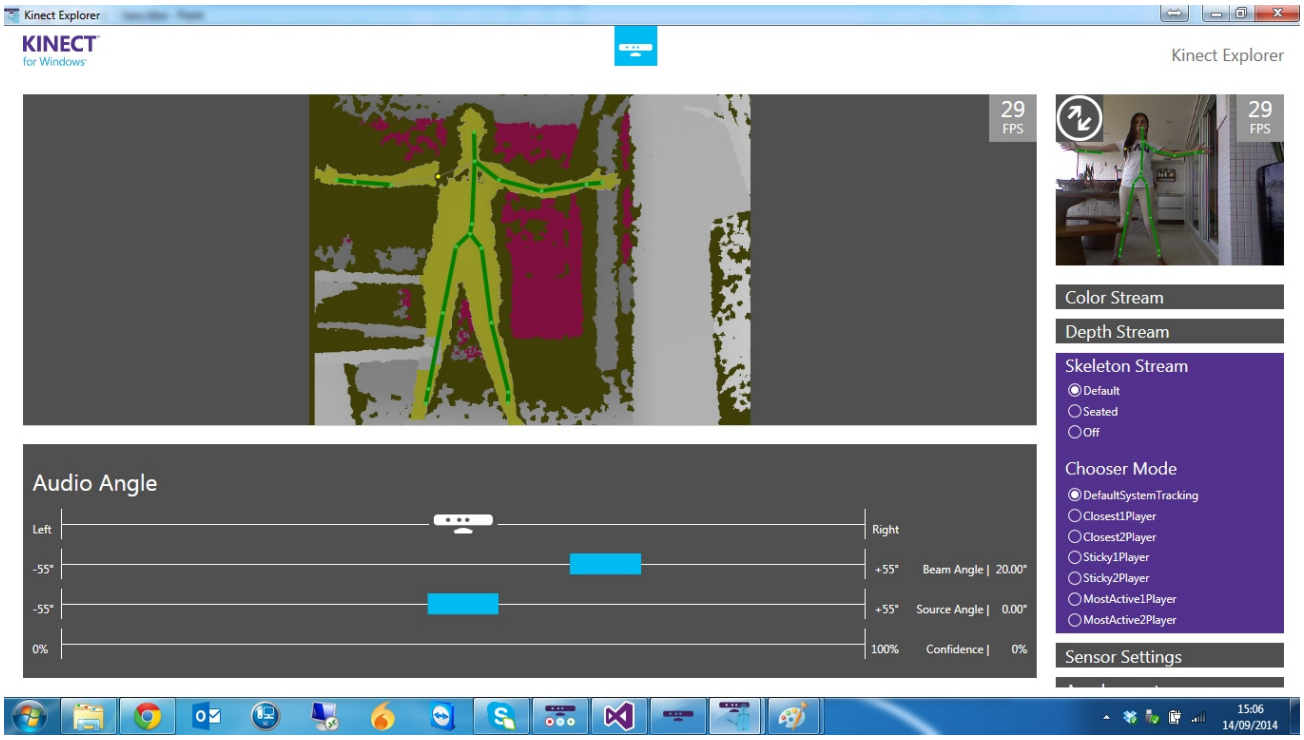

Figura 3.2. Kinect Explorer com reconhecimento de distancia com esqueleto. Fonte: G. Daré, 2014

3.2 Segunda Etapa

Para a segunda etapa, foi executada a implantação do reconhecimento de determinados movimentos para ligar e desligar a planta analisada.

Para o entendimento deste procedimento faz-se necessário o conhecimento de algumas particularidades do Kinect e sua programação através das *APIs*, são elas:

1) Fluxo de dados no Kinect;
2) As classes, métodos e eventos para cada Fluxo de dado.

A primeira particularidade pode ser facilmente entendida conforme o diagrama abaixo, Figura 3.3:

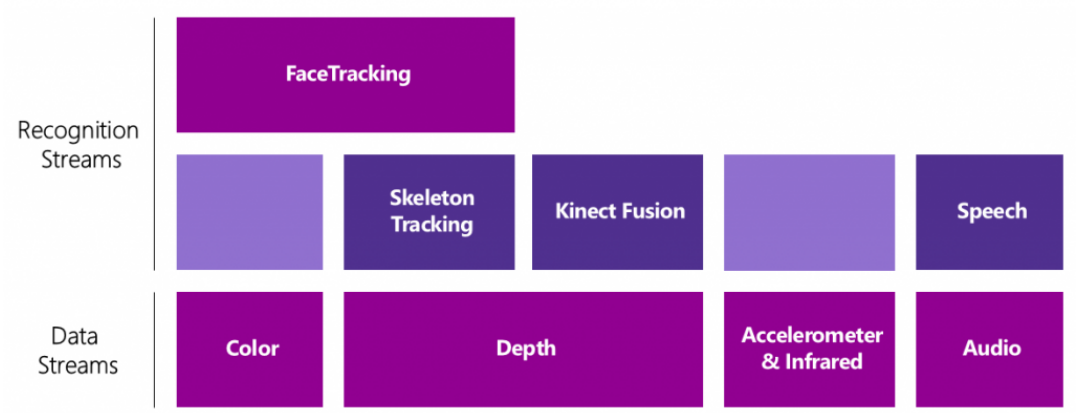

Figura 3.3 Fluxo de Dados do Kinect. Fonte: http://www.kinectingforwindows.com, 2014

Neste diagrama pode-se visualizar que para cada tipo de aplicação, seja ela rastrear uma face, uma pessoa, uma voz ou a fusão de alguns desses recursos, o Kinect pode fornecer algum fluxo de dado capaz de realiza-lo.

Vamos para o experimento aqui explanado: é necessário rastrear algum movimento que será interpretado e aplicado a uma planta específica. Logo, o fluxo de reconhecimento (*recognition stream*) a ser utilizado será o rastreamento do esqueleto (*skeleton tracking*), que nos será fornecido pelo Kinect como fluxo de dados (*Data streams*) a profundidade, ou mais simples de ser entendido, a posição de determinada parte do corpo no espaço.

Para tanto, algumas classes, métodos e eventos são fornecidos pela API afim de obter os dados ditos anteriormente. Estes podem ser visualizados de forma resumida na Figura 3.4 abaixo:

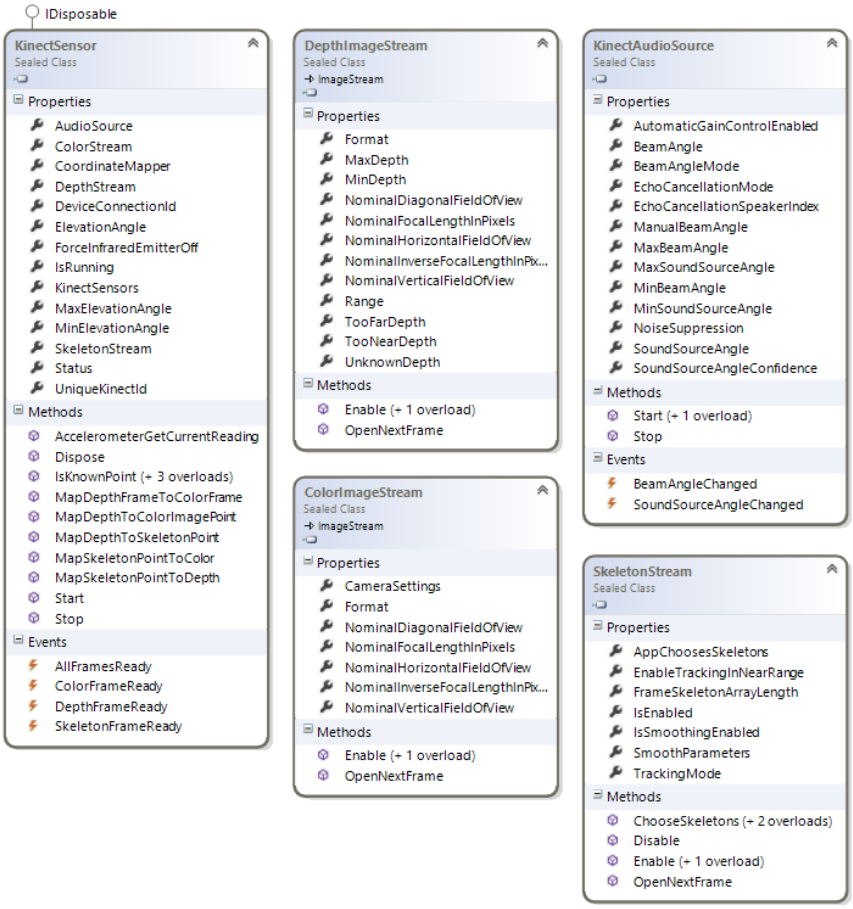

Figura 3.4 Classes principais do Kinect. Fonte: http://www.kinectingforwindows.com, 2014

Com essas classes foi possível executar as ações necessárias como inicializar o Kinect, rastrear as posições de membros pré-definidos como mão direita e esquerda, braços entre outros membros.

Como podemos ver abaixo estão as inclusões das bibliotecas das API para o Kinect.

```
#include <windows.h>
#include <objbase.h>
#include <ole2.h>
#include <oleauto.h>
#include <winsock2.h>
#include "NuiApi.h"
#include <iostream>
#include <ws2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <Shlobj.h>
```

Estas bibliotecas foram um dos principais problemas para a compilação do programa, uma vez que algumas bibliotecas depende de outras anteriormente. Exemplificando isso, ao utilizar a API “NuiApi.h”, para captar as posições da mão no espaço, era necessário ter a biblioteca “Windows.h”. Dessa forma, até descobrir quais

bibliotecas são necessárias para o bom funcionamento do seu programa demanda conhecimento das bibliotecas.

Abaixo é uma sequência de comandos afim de inicializar o Kinect da melhor forma. Esta deve ser realizada uma vez apenas.

```
NuiInitialize(NUI_INITIALIZE_FLAG_USES_SKELETON);
NUI_SKELETON_FRAME ourframe;
cout << "Kinect Iniciada" << endl;

// Socket de Comunicacao
WSADATA wsaData;
SOCKET ConnectSocket = INVALID_SOCKET;
struct addrinfo *result = NULL,
       *ptr = NULL,
       hints;
char *sendbuf = "this is a test";
char *on = "Planta Ligada";
char *desliga = "Planta OFF";
char recvbuf[DEFAULT_BUFLEN];
int iResult;
int recvbuflen = DEFAULT_BUFLEN;
```

Uma vez inicializado o Kinect, a cada obtenção de dados é necessário criar um Frame. Este, é como se fosse uma “foto” do momento em que quer analisar os dados necessários. Com esse Frame criado, consigo obter dados de posição da mão direita, esquerda e dos ombros direito e esquerdo. Desta forma, posso criar comando que no momento em que a mão direita ultrapassa a altura do ombro direito eu ligo a planta, bem como, se a mão esquerda fazer o mesmo eu desligo a mesma. Com esta simples comparação, não se faz necessário depender da distancia para o Kinect e a comparação é mais confiável. Este comando é feito como descrito:

```
while (1)
{
       NuiSkeletonGetNextFrame(0, &ourframe);
       cout << "Adquirindo dados" << endl;
       for (int i = 0; i < 6; i++)
       {
              if (ourframe.SkeletonData[i].eTrackingState == NUI_SKELETON_TRACKED)
              {
                     cout << "Mao Direita: x = " <<
ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_HAND_RIGHT].x << " y = " <<
ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_HAND_RIGHT].y << endl;
                     cout << "Mao Esquerda: x = " <<
ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_HAND_LEFT].x << " y = " <<
ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_HAND_LEFT].y << endl;

                     if
(ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_HAND_RIGHT].y >
ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_SHOULDER_RIGHT].y)
                     {
                            liga = 1;
                     }
                     if
```

```
(ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_HAND_LEFT].y >
ourframe.SkeletonData[i].SkeletonPositions[NUI_SKELETON_POSITION_SHOULDER_LEFT].y)
                        {
                                liga = 0;
                        }
                }
                if (liga_ant == 0 && liga == 1)
                {
                        cout << "Planta Ligada" << endl;
                        liga_ant = liga;
                }
                if (liga_ant == 1 && liga == 0)
                {
                        cout << "Planta Desligada" << endl;
                        liga_ant = liga;
                }
        }
}
```

Uma vez tendo o programa executando o que é necessário para esta aplicação, o último objetivo a ser executado é a comunicação entre o Kinect e o controlador da planta de furuta. Ou seja, foi feito um protocolo simples de comunicação entre o programa de interpretação do Kinect com o Matlab, programa de controle do pêndulo de furuta.

Para o envio das mensagens por parte do Kinect, é necessário a criação de um Socket de comunicação e inicializá-lo, conforme mostrado abaixo:

```
// Socket de Comunicacao
WSADATA wsaData;
SOCKET ConnectSocket = INVALID_SOCKET;
struct addrinfo *result = NULL,
        *ptr = NULL,
        hints;
char *sendbuf = "this is a test";
char *on = "Planta Ligada";
char *desliga = "Planta OFF";
char recvbuf[DEFAULT_BUFLEN];
int iResult;
int recvbuflen = DEFAULT_BUFLEN;

// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (iResult != 0) {
        printf("WSAStartup failed with error: %d\n", iResult);
        return 1;
}

ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;

// Resolve the server address and port
iResult = getaddrinfo("143.106.14.171", DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
        printf("getaddrinfo failed with error: %d\n", iResult);
        WSACleanup();
        return 1;
```

```
}

// Attempt to connect to an address until one succeeds
for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {

	// Create a SOCKET for connecting to server
	ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
		ptr->ai_protocol);
	if (ConnectSocket == INVALID_SOCKET) {
		printf("socket failed with error: %ld\n", WSAGetLastError());
		WSACleanup();
		return 1;
	}

	// Connect to server.
	iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
	if (iResult == SOCKET_ERROR) {
		closesocket(ConnectSocket);
		ConnectSocket = INVALID_SOCKET;
		continue;
	}
	break;
}

freeaddrinfo(result);

if (ConnectSocket == INVALID_SOCKET) {
	printf("Unable to connect to server!\n");
	WSACleanup();
	return 1;
}
```

Após a criação do socket de comunicação é necessário enviar as informações que necessita. Para isso, a cada identificação como anterior, mão direita com altura superior ao ombro e mão esquerda também, é necessário enviar os buffer definidos como liga e desliga a planta, com a seguinte função:

```
// Send a buffer
iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);
if (iResult == SOCKET_ERROR) {
	printf("send failed with error: %d\n", WSAGetLastError());
	closesocket(ConnectSocket);
	WSACleanup();
	return 1;
}

cout << "Mensagem Enviada" << endl;
```

Uma vez enviada a mensagem, o Matlab tem que receber essa mensagem e “jogar“ no sistema Simulink afim de ligar/desligar a planta. Até o reconhecimento do Matlab não houve problema de implementação, porém para alterar o sistema em tempo real foi onde houve muitos problemas. A solução encontrada é alteração de parâmetro direto no arquivo do Simulink, dessa forma podemos garantir a alteração em tempo real, uma vez que não

possuímos alguns pacotes no Matlab que facilitariam esse trabalho. O programa utilizado para criação do server e alteração do parâmetro em tempo real é o seguinte:

```
function server( output_port)

    import java.net.ServerSocket
    import java.io.*


    % wait for 1 second for client to connect server socket
    server_socket = ServerSocket(output_port);
    server_socket.setSoTimeout(100000);

    output_socket = server_socket.accept;

    fprintf(1, 'Client connected\n');

    input_stream   = output_socket.getInputStream;
    d_input_stream = DataInputStream(input_stream);

    while(1)
        % output the data over the DataOutputStream
        % Convert to stream of bytes
        pause(0.2);
        bytes_available = input_stream.available;

        message = zeros(1, bytes_available, 'uint8');
        for i = 1:bytes_available
            message(i) = d_input_stream.readByte;
        end

        message = char(message);
        t1 = strcmp(message,'Planta Ligada');
        t2 = strcmp(message,'Planta OFF');
        if t1
            set_param('system_quanser_kinect/Constant','Value','1');
        elseif t2
            set_param('system_quanser_kinect/Constant','Value','0');
        end

    end

    % clean up
    server_socket.close;
    input_socket.close;

end
```

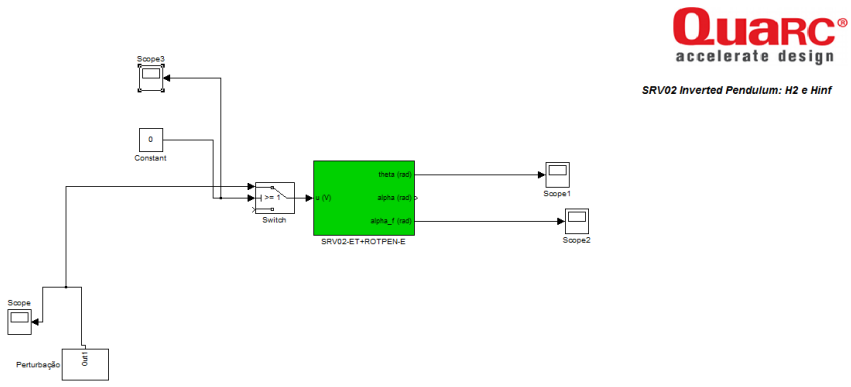

Figura 3.5 Simulink de Execução da Planta. Fonte: G. Daré, 2014

Uma vez todo sistema interligado e os programas executando, a ligação física da planta é feita de acordo com a Figura 3.5 abaixo.

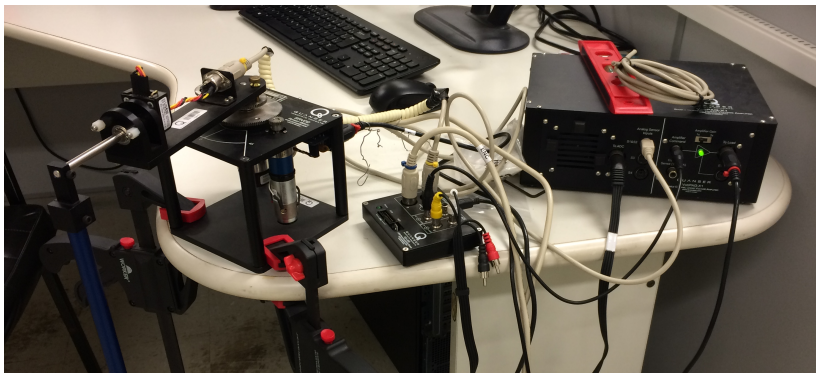

Figura 3.6 Ligação física da Planta. Fonte: G. Daré, 2014

Capítulo 4

Resultados e Discussões

Como resultado para esta primeira etapa, já é possível criar uma forma de interpretação do sensor com relação a movimentos já possuídos em suas bibliotecas de trabalho.

Existem diversos tutoriais para instalação das API e *softwar*es de desenvolvimento de programação para qualquer plataforma utilizada. Em geral, não houve nenhuma dificuldade na instalação e execução dos programas exemplos.

Ao ligar a Kinect, a primeira tarefa que o sistema irá fazer é encontrar os pontos de referencia: Mão direita e esquerda e ombro direito e esquerdo. Uma vez reconhecido os membros do corpo, o programa irá começar a rastrear através de um plano x e y, conforme mostra a Figura 4.1.

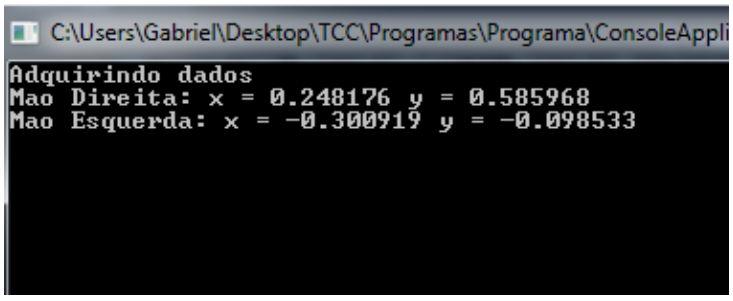

Figura 4.1 Posição x e y da mão direita e esquerda. Fonte: G. Daré, 2014

Como resultado do reconhecimento de liga e desliga, temo o seguinte: ao erguer o braço direito a uma altura acima do ombro direito, conforme figura 4.2. Já para desligar a planta o sistema reconhece quando a mão esquerda esta a uma altura acima do ombro esquerdo, conforme mostrado na figura 4.3.

Figura 4.2 Movimento de Ligar a Planta. Fonte: G. Daré, 2014

Figura 4.3 Movimento de Desligar a Planta. Fonte: G. Daré, 2014

Capítulo 5

Conclusões

Como conclusão, temos que embora tivesse algumas dificuldades com questão de bibliotecas (API) e com a atualização em tempo real do estado da planta com as configurações da Kinect. Este último nos da condição de dar prosseguimento para a segunda etapa que é uma atualização mais dinâmica da posição da mão em interação com a planta, o pêndulo de furuta.

Assim já temos todas as condições de reconhecimento dos membros do corpo em comunicação em tempo real com a planta que necessitarmos, nesse caso o pêndulo de furuta.

Referências Bibliográficas

[1] A. Ladzinski, Development of 3D Image Manipulation Software Using the Microsoft Kinect, Murdoch University, 2013. 66p.

[2] F. Santos, Aplicação para reconhecimento de Gestos, baseada em OPENNI, Faculdade Anhanguera de Belo Horizonte, 2013. 10p.

[3] B. Cazzolato & Z. Prime, On the Dynamics of the Furuta Pendulum, University of Adelaide, 2011. 9p.

[4] Workbook Rotary Pendulum, Instructior Version, Quanser Inc., 2011. 60p.