UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

LUCAS JOSÉ DOS SANTOS SOUZA

Controle de computadores utilizando interface natural

São Carlos
2014

LUCAS JOSÉ DOS SANTOS SOUZA

CONTROLE DE COMPUTADORES UTILIZANDO INTERFACE NATURAL

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

Curso de Engenharia de Computação
com ênfase em Sistemas Computacionais
Avançados

ORIENTADOR: Prof. Dr. Adilson Gonzaga

São Carlos
2014

Souza, Lucas José dos Santos

S764c Controle de computadores utilizando interface natural / Lucas José dos Santos Souza; orientador Adilson Gonzaga. São Carlos, 2014.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Interface natural. 2. Kinect. 3. Processamento de imagens. 4. Interação humano-computador. I. Título.

FOLHA DE APROVAÇÃO

Nome: Lucas José dos Santos Souza

Título: "Controle de computadores utilizando interface natural"

Trabalho de Conclusão de Curso defendido em 17 / 11 / 2014 .

Comissão Julgadora:	**Resultado:**
Prof. Associado Adilson Gonzaga (Orientador) - SEL/EESC/USP	APROVADO
Prof. Assistente Carlos Goldenberg SEL/EESC/USP	APROVADO
Mestre Raissa Tavares Vieira Doutoranda - SEL/EESC/USP	APROVADO

Coordenador do Curso Interunidades - Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Ainda que eu tenha o dom de profecia e saiba todos os mistérios e todo o conhecimento, e tenha uma fé capaz de mover montanhas, mas não tiver amor, nada serei.

Primeira carta de Paulo aos Coríntios, cap. 13, v. 2

DEDICATÓRIA

Dedico este trabalho aos meus pais, José Pereira e Luzia, pelo apoio e orientações que me deram ao longo de toda a vida. Sem eles, não teria chegado aonde estou.

Também a todos os meus amigos que me acompanharam ao longo desta carreira, me dando apoios e alegrias durante os anos de curso e durante os dias em que passei trabalhando neste projeto. Em especial, a Alice, que mais ansiosamente esperava por este trabalho.

A Alex Fernando Orlando, chefe e amigo, grande contribuição para minha carreira.

Ao professor que me deu as oportunidades de iniciação científica e trabalho de conclusão de curso e me orientou no desenvolvimento desses trabalhos, Prof. Dr. Adilson Gonzaga.

Aos professores que me orientaram ao longo do curso e me deram grandes oportunidades, em especial, Prof. Dr. Carlos Dias Maciel, que desde antes de eu estar na universidade me esperava e apoiava em meus projetos.

AGRADECIMENTOS

A Deus, por ter me guiado até mesmo através dos momentos mais difíceis da vida até aqui. Graças a Ele, venci esses obstáculos e não desisti.

A Caio César Viel, pela ajuda no desenvolvimento do projeto e pelas boas sugestões, que realmente me ajudaram.

Mais uma vez, a meus pais e a todos os meus amigos que me apoiaram e me ajudaram a chegar aonde estou na vida.

RESUMO

SOUZA, L. J. dos S. **Controle de computadores utilizando interface natural.** 2014. 73 p. Trabalho de conclusão de curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2014.

Quando se trata de computação, um dos principais objetivos para os desenvolvedores e usuários é tornar a interação entre o usuário e o sistema mais fácil e intuitiva. Ao longo dos anos, foram sendo desenvolvidos diversos tipos de interação. Uma das tecnologias com esse objetivo foi lançada em 2010 para jogos eletrônicos da empresa Microsoft, o Microsoft Kinect. A tecnologia, que se baseia no rastreamento das partes do corpo humano e a detecção de movimentos e gestos do usuário, gerou um grande interesse de empresas, institutos e desenvolvedores independentes. Vendo esse interesse, a Microsoft lançou um dispositivo Kinect voltado especialmente para computadores com o sistema operacional Microsoft Windows e também um *Software Development Kit* (SDK) para que pessoas de fora da Microsoft também possam desenvolver aplicações [1]. Neste projeto, foram utilizados o SDK e o Kinect para desenvolver uma forma de interação usuário-computador que possa substituir o mouse e o teclado, permitindo que o usuário controle o cursor e suas ações de clique com as mãos e também digite caracteres como com um teclado. Os resultados se mostraram satisfatórios e bem funcionais.

Palavras-chave: Interface natural, Kinect, processamento de imagens, interação humano-computador.

ABSTRACT

SOUZA, L. J. dos S. **Computers control using natural interface.** 2014. 73 p. Trabalho de conclusão de curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2014.

When it comes to computers, one of the main objectives for developers and users is to make the interaction between the user and the system easier and more intuitive. Over the years, many kinds of interactions were developed. One of the technologies with that objective was released in 2010 for Microsoft's video games, Microsoft Kinect. The technology, which is based on tracking human body parts and the detection of movements and gestures from the user, has aroused a great interest from companies, institutes and independent developers. Seeing that interest, Microsoft released a Kinect device made especially for computers with Microsoft Windows operating systems and also a Software Development Kit (SDK) so that people outside Microsoft could also develop applications [1]. In this project, the SDK and the Kinect were used to develop a way of user-computer interaction that can replace the mouse and the keyboard, allowing that users control the cursor and its click actions with their hands and type characters as with a keyboard. The results showed themselves to be satisfactory and well functional.

Key words: Natural interface, Kinect, image processing, human-computer interaction.

Sumário

Lista de Figuras

Lista de Siglas

SDK	*Software Development Kit* (*Kit* de desenvolvimento de *software*)
CLI	*Command-Line Interface* (Interface de linha de comando)
GUI	*Graphical User Interface* (Interface gráfica de usuário)
NUI	*Natural User Interface* (Interface natural de usuário)
RGB	*Red-Green-Blue* (Vermelho-Verde-Azul, referente às cores de imagens)
CLR	*Common Language Runtime*
API	*Application Programming Interface*

1 INTRODUÇÃO

1.1 Contextualização

Define-se interação humano-computador como “uma disciplina preocupada com o projeto, avaliação e implementação de sistemas computacionais interativos para uso humano e com o estudo dos principais fenômenos que os cercam” [2]. Isto é, o estudo que busca o desenvolvimento de formas de um ser humano participar da execução de um programa de computador através de dispositivos de entrada (como teclados, *mouse*, câmeras, microfones...) e de visualizar seus resultados através de dispositivos de saída (telas, caixas de som...).

Existem diversos tipos de interação, dentre as quais podem-se citar a interface de linha de comando (CLI), a interface gráfica (GUI) e a interface natural (NUI).

Na interface de linha de comando, o papel do usuário depende de instruções de texto que devem ser digitadas e processadas pelo sistema, que como resposta pode ou não retornar alguma informação que pode ser visualizada via texto. Esse tipo de interface foi muito utilizado até os anos 1980 como a principal forma de interação com o usuário no início do uso dos computadores. Seu uso exige certo conhecimento e experiência por parte do usuário, sendo preferido mais por usuários avançados para ter um controle maior das funções do sistema operacional. Para um usuário casual, isto é, alguém que não tem um profundo conhecimento do funcionamento dos sistemas operacionais e suas utilidades, em geral é preferido o uso de interfaces gráficas. [3]

Como uma alternativa à linha de comando, foi criada a interface gráfica de usuário, através dos trabalhos de diversos inovadores. Esse tipo de interface permite que o usuário se comunique com o computador através de símbolos, metáforas visuais e dispositivos de apontamento (*mouse*, por exemplo). [4] Isso tornou o uso dos computadores mais fácil e intuitivo mesmo por usuários casuais, como pode ser observado hoje na grande maioria dos sistemas operacionais e em seus usuários, sendo destaques de sua utilização os sistemas Microsoft Windows, Apple Mac OS X e diversas distribuições de Linux, como Ubuntu, Fedora e OpenSUSE, além de dispositivos móveis, como os *smartphones* com os sistemas iOS (Apple), Android e Microsoft Windows Phone.

É importante que a interação do usuário com o computador seja tão simples, intuitiva e poderosa quanto possível. A interface natural de usuário surge como uma potencial

evolução da interface gráfica [5], buscando ser capaz de fazer um usuário iniciante se tornar experiente em um curto período de tempo. Uma de suas possíveis aplicações se dá em gestos de um usuário sendo transmitidos ao sistema para, por exemplo, animar um objeto 3D. Esse tipo de aplicação é utilizado em consoles de *videogame* como o Nintendo Wii (através de um controle com sensor de movimentos) e Microsoft Xbox 360 (através do sensor Kinect).

1.2 Microsoft Kinect

Lançado em 2010, o sensor Kinect (Figura 1) foi desenvolvido inicialmente para o console de *videogame* Microsoft Xbox 360 como uma interface alternativa aos controles físicos. Através de diversos sensores como uma câmera RGB, um sensor de profundidade infravermelho e um arranjo de microfones [7], o Kinect permite detecção de áudio com supressão de ruído, detecção de jogadores e rastreamento de suas partes do corpo, além de informações como posições em três dimensões. O aparelho conta ainda com um motor de inclinação vertical, que pode ser ajustada via *software* conforme o necessário. Isso é útil, por exemplo, para encontrar a melhor posição para o rastreamento do corpo do jogador. Como exemplos de jogos que utilizam a tecnologia, podem-se citar “Kinect Adventures”, “Kinect Sports” e “Dance Central”.

Figura 1 – Microsoft Kinect [6]

A tecnologia despertou interesse não só dos jogadores como também de empresas, pesquisadores e desenvolvedores independentes, e muitos começaram a tentar descobrir como o sensor funciona através de engenharia reversa. A Microsoft inicialmente desaprovou essas ações, mas vendo o potencial que a tecnologia oferecia, permitiu que o

sensor fosse usado para outras funções além de entretenimento, desde que isso não caracterizasse *hack* do aparelho, isto é, modificação no dispositivo ou no console, e sim desenvolvimento baseado no USB e em seus recursos [8].

Foram criados *drivers* para o uso da câmera RGB e do sensor de profundidade para o sistema Linux e, mais tarde, foram desenvolvidas ferramentas para esses *drivers*, como o *middleware* NITE e o *framework* OpenNI [9]. Mais tarde, a Microsoft desenvolveu um *Software Development Kit* (SDK) e um aparelho voltados especificamente para computadores com o sistema operacional Microsoft Windows e o *framework* .NET, podendo ser utilizadas as linguagens C++, C# e Visual Basic [10, 11].

1.3 Objetivos

O projeto proposto teve como objetivo a implementação de uma interface natural utilizando o Kinect para Windows e o SDK apresentados e testar suas capacidades de uso em computadores, possivelmente como substituto das formas atuais de interação com o computador, como o *mouse* e o teclado.

Essa interface deve servir como um controlador do cursor do *mouse* no computador e possuir algum processamento para se simularem as teclas do teclado, de forma a ser uma camada intermediária entre as ações do usuário e a interface gráfica já existente.

Espera-se também, com esse projeto, aprender um pouco mais sobre interfaces naturais e se ter uma ideia melhor de quais são algumas possíveis aplicações do Kinect nessa área, bem como saber quais são suas capacidades e limitações.

1.4 Estrutura

O capítulo 2 descreve os conceitos teóricos já conhecidos e pesquisados para o desenvolvimento do trabalho. O capítulo 3 descreve a forma como o sistema foi planejado e implementado. O capítulo 4 apresenta e discute os resultados obtidos e se o sistema foi projetado e funciona de forma satisfatória. O capítulo 5, por fim, apresenta uma conclusão geral sobre o trabalho desenvolvido, o que foi aprendido e possíveis trabalhos futuros.

1.5 Observação sobre o projeto

Esse projeto deve servir como continuação ao de iniciação científica realizado pelo mesmo autor do presente trabalho de conclusão de curso entre agosto de 2013 e julho de 2014 pelo CNPq, sob orientação do professor Dr. Adilson Gonzaga [12].

2 FUNDAMENTAÇÃO TEÓRICA

O projeto desenvolvido envolvia dois principais conceitos teóricos: conhecimento da linguagem de programação C# com o *framework* .NET 4.5 e da programação com o Kinect SDK. Não havia conhecimentos prévios em nenhuma dessas tecnologias, portanto, alguns materiais tiveram que ser consultados.

2.1 C# .NET

Para o aprendizado da linguagem C#, foi utilizada principalmente a especificação da linguagem da Microsoft. [13] Devido ao paradigma orientado a objetos e à grande semelhança com a linguagem Java, com a qual já havia familiaridade e experiência, não foi difícil encontrar respostas às dúvidas que surgiam e se familiarizar com a linguagem e sua forma de criar interfaces gráficas e tratar seus eventos.

Quando se usa o *framework* .NET para aplicações que contêm janelas, o gerenciamento de seus eventos e do *loop* principal do sistema ficam sob responsabilidade da classe Application, que é gerenciada em *background* pelo *Common Language Runtime* (CLR), uma espécie de máquina virtual que também gerencia segurança, memória, *threads*, exceções e uma linguagem de *bytecode* comum para a plataforma .NET (criando códigos compatíveis entre as linguagens C#, Visual Basic entre outas). Com isso, cabe ao programador apenas descrever as janelas e classes de controle do sistema, contando com o auxílio do construtor de interfaces gráficas da IDE Microsoft Visual Studio.

Uma dificuldade encontrada foi descobrir como modificar a posição do cursor do *mouse* e executar os eventos de cliques através de *software*, pois isso necessitava de um acesso às APIs de baixo nível do sistema operacional. A resposta foi encontrada em um fórum de programação [14] que continha um exemplo de uma classe, implementada pelo usuário Keith, que executava essas operações através da importação de uma biblioteca (DLL) do sistema e então criava métodos de acesso a suas funções.

2.2 Kinect SDK

Por se tratar de uma tecnologia nova, não foram encontradas muitas referências que ensinassem a utilizar o Kinect SDK, sendo que no começo das pesquisas pelo

desenvolvedor do projeto sobre o assunto (final de 2012 e começo de 2013) os guias na Internet ainda utilizavam uma versão beta do SDK, e não as versões mais novas (à época, 1.5 e 1.6), que introduziram diferenças realmente significantes.

Outros exemplos, que utilizavam as versões mais recentes, se focavam mais no uso do sensor apenas dentro de uma janela ou de foco específico em alguma área (como exibir a imagem da câmera RGB na tela, gerar imagens cuja cor varia conforme a profundidade, entre outras), e não em um aplicativo para uso em todo o sistema do computador. Para tanto, foi utilizado um site [15] com um tutorial básico de utilização dos sensores do aparelho, o que permitiu o aprendizado de forma rápida e simples e permitindo que o projeto fosse implementado da forma como era desejado.

O SDK providencia um nível de abstração alto para o programador. Toda a parte de processamento de imagens fica por conta das bibliotecas oferecidas pela Microsoft, ficando disponíveis ao programador informações tais como: quantos e quais jogadores foram detectados pelo sensor, as localizações em três dimensões de cada parte do corpo do jogador, quais *streams* estão disponíveis, entre outras.

Com os sensores presentes no Kinect, é possível utilizar quatro *streams*: de cores, de profundidade, de esqueleto e de áudio.

O de cores é responsável por utilizar a câmera RGB do Kinect e obter seus dados assim que estiverem prontos na forma de *bytes*. Estes podem então ser convertidos em uma imagem Bitmap de 32 bits com um método disponível na classe BitmapImage.

O *stream* de profundidade permite que, para uma determinada resolução escolhida, seja obtido um *array* de dados do tipo *short*, contendo informações sobre a profundidade do ponto verificado em 13 *bits* e, caso pertença ao corpo de algum jogador, o número desse jogador nos 3 *bits* restantes. Esse *stream* não foi utilizado diretamente no projeto.

Já o *stream* de esqueleto provê uma importante abstração, que pode-se dizer que é a base do funcionamento do Kinect: através do processamento das imagens do sensor, ele consegue informar qual é o jogador que foi detectado e quais são as posições de suas *Joints*. As *Joints* são pontos do esqueleto que representam partes relevantes do corpo humano, tais como a cabeça, ombros, mãos, pés, entre outras. Um mapa completo das *Joints* detectadas pelo aparelho pode ser visto na Figura 2. O Kinect por padrão possui um *array* fixo de 6 esqueletos, podendo identificar todas as *Joints* de 2 deles e apenas inferir a posição dos outros 4, se existirem.

O *stream* de áudio provê métodos para captura dos dados dos microfones em *bytes*, semelhante ao funcionamento dos *streams* de cores e de profundidade. Esse *stream* também não foi utilizado no projeto.

Cada um desses *streams* possui métodos para ser iniciado, parado e lido. As informações podem ser obtidas via *poll*, mandando ser lido o próximo *frame* (conjunto de dados obtidos periodicamente) e checando se esses dados são válidos, ou via eventos, registrando uma função de *call-back* que será executada quando um novo *frame* estiver pronto.

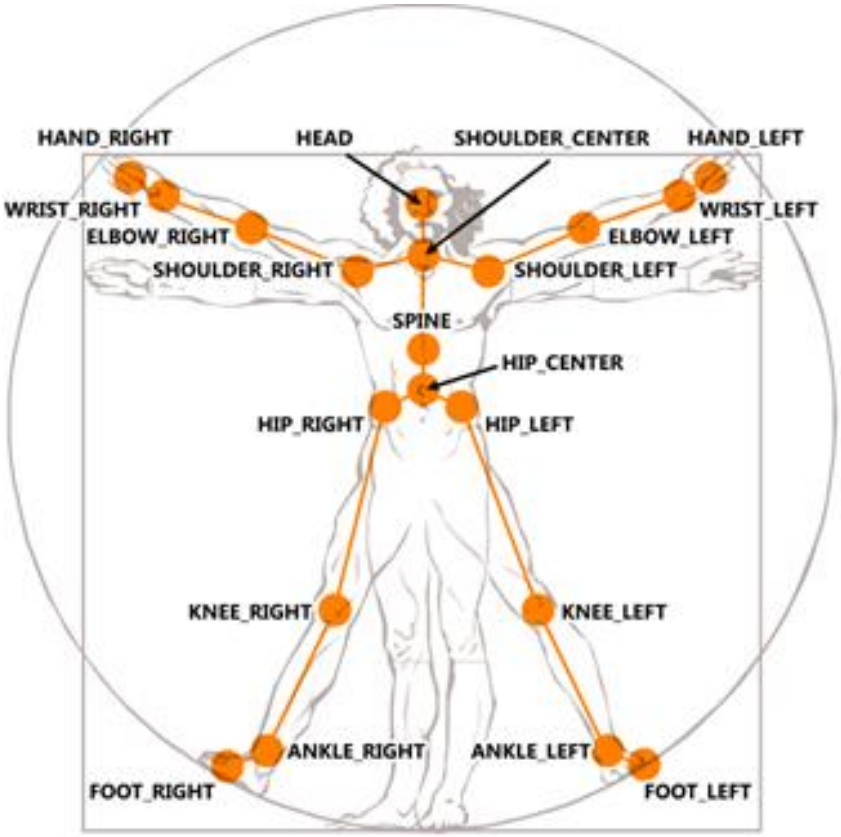

Figura 2 - Mapeamento das *Joints* do esqueleto [16]

O Kinect utiliza um sistema de coordenadas euclidiano em 3 dimensões para mapear o espaço que seus sensores “enxergam”. Seus eixos positivos são tais como vistos na Figura 3.

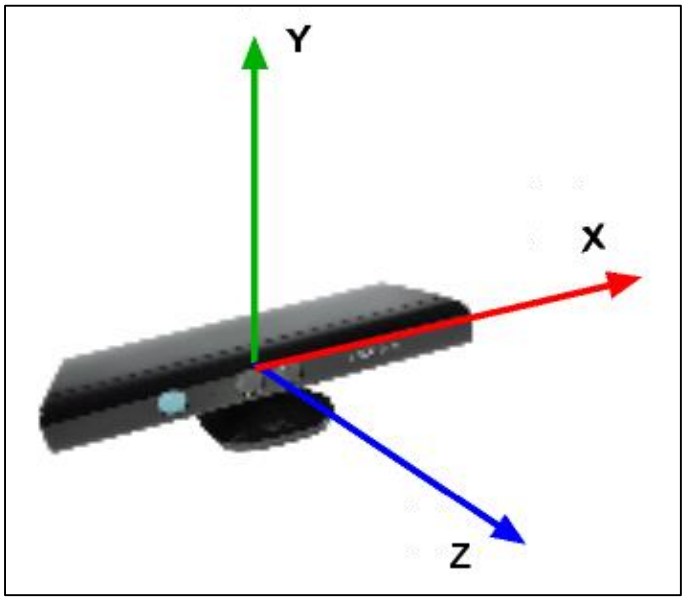

Figura 3 - Sistema de coordenadas euclidianas do Kinect [17]

2.3 Considerações finais

Conhecidos a linguagem e o SDK, restou apenas o planejamento e modelo do sistema para, enfim, fazer sua implementação. Para isso, foram utilizados conceitos básicos já conhecidos de programação orientada a objetos, como a separação entre a interface de usuário e os controladores e a especialização de cada componente do programa. Detalhes do desenvolvimento do projeto podem ser encontrados no capítulo 3.

3 MATERIAIS E MÉTODOS

3.1 Materiais

Para o desenvolvimento do projeto, foram utilizados:

- Aparelho Microsoft Kinect for Windows, já obtido anteriormente pelo orientador do projeto no laboratório
- Microsoft Kinect SDK versão 1.7, a mais recente no momento em que o projeto começou a ser implementado [18]
- Microsoft Visual Studio 2012 Ultimate, [19] com licença obtida pelo programa MSDNAA da Microsoft com o Instituto de Ciências Matemáticas e de Computação – ICMC, da USP São Carlos
- Notebook Acer Aspire 4736Z com o sistema operacional Microsoft Windows 7, 3 GB de memória RAM e processador Intel Pentium T4300 de 2,1 GHz, já pertencente ao aluno.

O projeto e as experiências foram realizados no Laboratório de Visão Computacional (LAVI) da Escola de Engenharia de São Carlos, na USP São Carlos. Uma foto dos materiais utilizados pode ser vista na Figura 4.

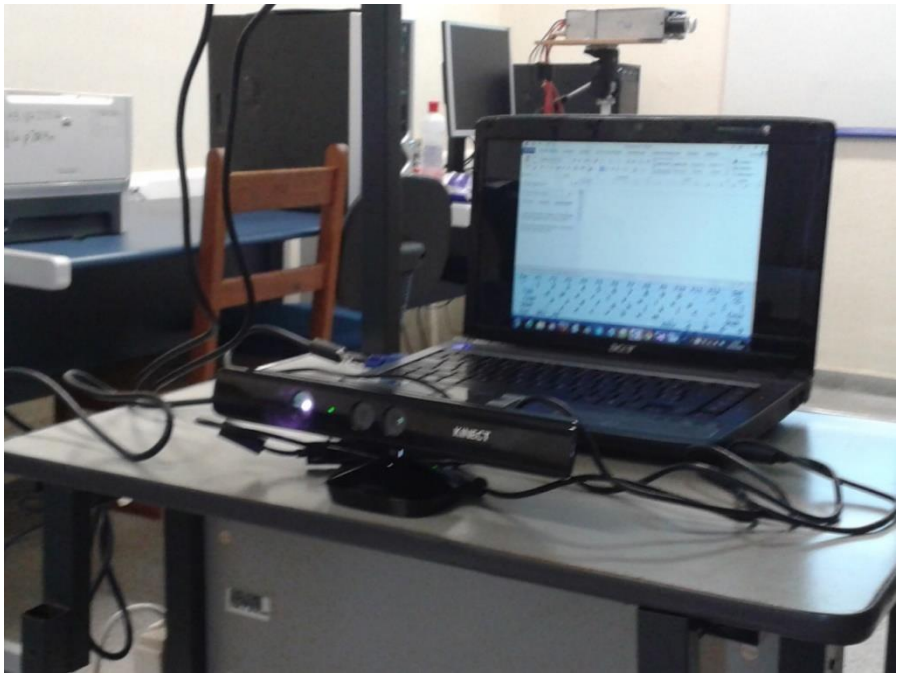

Figura 4 – Materiais utilizados no desenvolvimento do projeto

3.2 Metodologia

3.2.1 Treinamento

Antes do desenvolvimento do projeto propriamente dito, foram desenvolvidos alguns exemplos mais simples para treinar as tecnologias disponíveis na linguagem C# e no Kinect SDK. Não é necessário entrar em detalhes sobre esses exemplos, pois o aprendizado obtido com eles foi utilizado no desenvolvimento do projeto final.

Para a linguagem C#, esses exemplos consistiam basicamente em aprender como funciona a construção da janela e suas propriedades, como tamanho, localização e cores, e também de seus componentes, como botões, *checkboxes* e ícones.

Para o SDK, os exemplos consistiam em utilizar os *streams* de cor e de esqueleto, ver os tipos de informações que eram disponibilizadas, os erros que podiam ser gerados e como obter dados relevantes dessas informações, como as posições das *Joints*, o sistema de coordenadas utilizado pelo Kinect e como capturar os dados da câmera RGB para exibi-los ao usuário, tal como descrito na seção 2.2.

3.2.2 Implementação

Tendo o conhecimento necessário, foi planejado fazer uma aplicação que fosse executada em *background*, com a parte principal sem interfaces gráficas, apenas com o controle do sensor, contando com janelas apenas para funções auxiliares, como modificar opções ou exibir imagens da câmera. Porém, não foi encontrado um meio simples de fazer isso, pois o sistema ficava sem controle sobre quais tarefas deveriam ser executadas e quando deveria ser encerrado.

A solução encontrada para esse problema [20] foi criar uma aplicação do tipo Windows Forms Application contendo uma janela principal invisível, a partir da qual o programa era gerenciado através de um ícone de bandeja do sistema. Se o usuário desejasse encerrar o programa, deveria selecionar a opção de encerrar em um menu desse ícone, o que fazia que a janela principal fosse fechada e um evento do tipo FormClosed encerrasse os recursos alocados pelo sistema (ícone de bandeja, *streams* habilitados do Kinect, janelas entre outros).

Essa janela principal ficou sendo a classe responsável por gerenciar as instâncias de todos os outros componentes do sistema. Esses outros componentes incluem: uma janela de opções, uma classe responsável por gerenciar o Kinect e uma janela responsável por exibir a imagem da câmera RGB.

Quando o programa é iniciado, a classe Application instancia a janela principal, que por sua vez exibe uma mensagem de notificação ao usuário de que o sistema está sendo iniciado (Figura 5.a). Se todos os componentes puderem ser inicializados corretamente, outra mensagem de notificação é exibida informando que o sistema está pronto para ser utilizado (Figura 5.b). Também é criado um menu com quatro opções neste ícone para que o usuário interaja com o sistema (Figura 6).

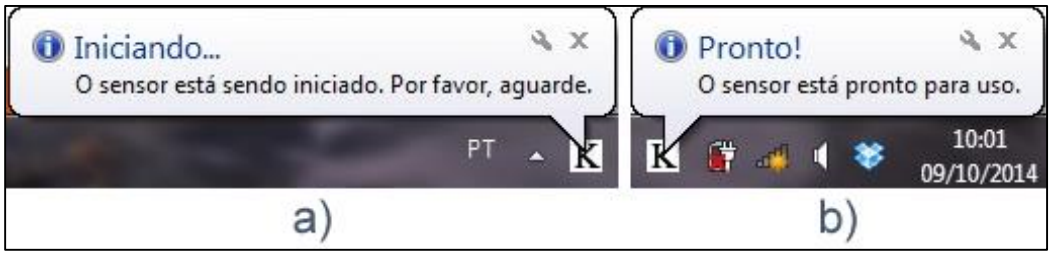

Figura 5 - Mensagens de notificação ao usuário. a) Mensagem de inicialização; b) Mensagem indicando que o sistema está pronto

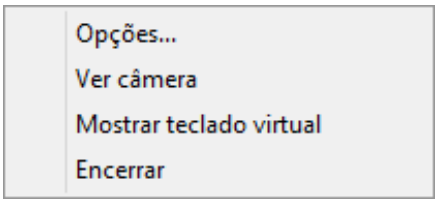

Figura 6 - Menu de opções

Quando os componentes do sistema ficam prontos, a classe KinectController recolhe informações como a largura e a altura da tela virtual e verifica se há algum Kinect conectado (pode haver mais de um). Caso pelo menos um seja detectado, a lista de sensores é percorrida até encontrar um sensor que possua o status “Connected” (conectado) ou “Initializing” (inicializando).

Um possível erro é que o sensor seja detectado mas não seja possível estabelecer a conexão devido a, por exemplo, pouca energia na porta USB. Nesse caso, uma mensagem de erro é exibida pedindo que o usuário verifique a conexão.

Se o sensor for detectado e a conexão estabelecida, o sistema cria um temporizador (*Timer*) com taxa de atualização de 15 quadros por segundo, habilita o *stream* de esqueleto e coloca o ângulo do Kinect em 10 graus.

Também são definidos alguns conceitos importantes no projeto: **mão padrão**, **mão secundária**, **ombro padrão** e **ombro secundário**. Os membros padrões representam inicialmente os que ficam do lado direito do corpo, enquanto que os secundários são os que ficam do lado esquerdo. Essa configuração pode ser mudada através do menu de opções, conforme a Figura 6.

Quando o temporizador conclui seu tempo, é disparado um evento que abre o próximo quadro de esqueleto, verificando se está disponível (portanto, as informações são obtidas via *poll*. Essa abordagem foi escolhida porque assim é possível mudar a taxa de atualização dos quadros). Em seguida, a lista de 6 esqueletos é percorrida até encontrar o primeiro esqueleto que esteja disponível. Se algum for encontrado, o sensor verifica as posições X e Y da mão principal, conforme descrito na Figura 3, salvando-as nas variáveis "rightX" e "rightY". Essas variáveis são multiplicadas por um valor de escala, obtendo os valores "nextX" e "nextY" conforme as equações (1) e (2).

$$nextX = \left(\frac{screenWidth}{2} + rightX * screenWidth * 2.0\right) \quad (1)$$

$$nextY = \left(\frac{screenHeight}{2} - rightY * screenHeight * 1.5\right) \quad (2)$$

Os valores "screenWidth" e "screenHeight" representam a largura e a altura da tela, respectivamente, valores obtidos na instanciação da classe KinectController. O cursor do *mouse* pode, então, ser atualizado para essas posições calculadas.

Depois de modificada a posição, o próximo passo consiste em verificar as ações de clique, que dependem das coordenadas da mão secundária. O planejamento inicial envolvia a utilização apenas da mão principal também para os cliques, apenas movendo-a para a frente, porém, foi encontrada uma grande dificuldade: era difícil manter a mão na mesma posição em X e Y enquanto ela se movia para a frente. Muitas vezes, o braço "girava" nesse movimento, fazendo com que o clique fosse executado em um local diferente do planejado. Para corrigir esse problema, foi adotada a mão secundária.

Para que um evento de botão de *mouse* pressionado seja enviado, é preciso que a mão secundária esteja a uma distância (no eixo Z) maior que um valor determinado do ombro secundário, enquanto que para que um evento de botão solto seja enviado, é preciso que a mão esteja a uma distância menor que esse valor.

Esse valor de distância, por sua vez, depende da altura (coordenada Y) da mão secundária em relação à altura do ombro secundário, conforme esquematizado na Figura 7. Se a mão estiver a mais de 20 cm acima do ombro, a distância em Z que a mão deve

estar é de 30 cm (Figura 7.a). Neste caso, considera-se que o usuário deseja utilizar o botão direito do *mouse*. Se a mão estiver entre 20 cm acima (Figura 7.b) e 10 cm abaixo (Figura 7.c) do ombro secundário, a distância em Z deve ser de 45 cm e considera-se que está sendo utilizado o botão do meio. Se a mão estiver abaixo de 10 cm do ombro, a distância deve ser também de 45 cm e considera-se que está sendo utilizado o botão esquerdo (Figura 7.d).

No menu associado ao ícone (Figura 6), é possível executar outros três módulos do sistema. O primeiro deles é uma janela de opções (Figura 8) onde o usuário pode modificar algumas configurações do projeto, como qual deve ser a mão padrão, a taxa de atualização do cursor (isto é, a frequência com que os quadros do esqueleto devem ser lidos), o ângulo de inclinação do Kinect (que pode variar entre -27° e +27°, sendo que ângulos positivos indicam que o aparelho aponta para cima e 0° indica que a linha de visão do aparelho está paralela ao chão) e o **modo de precisão**.

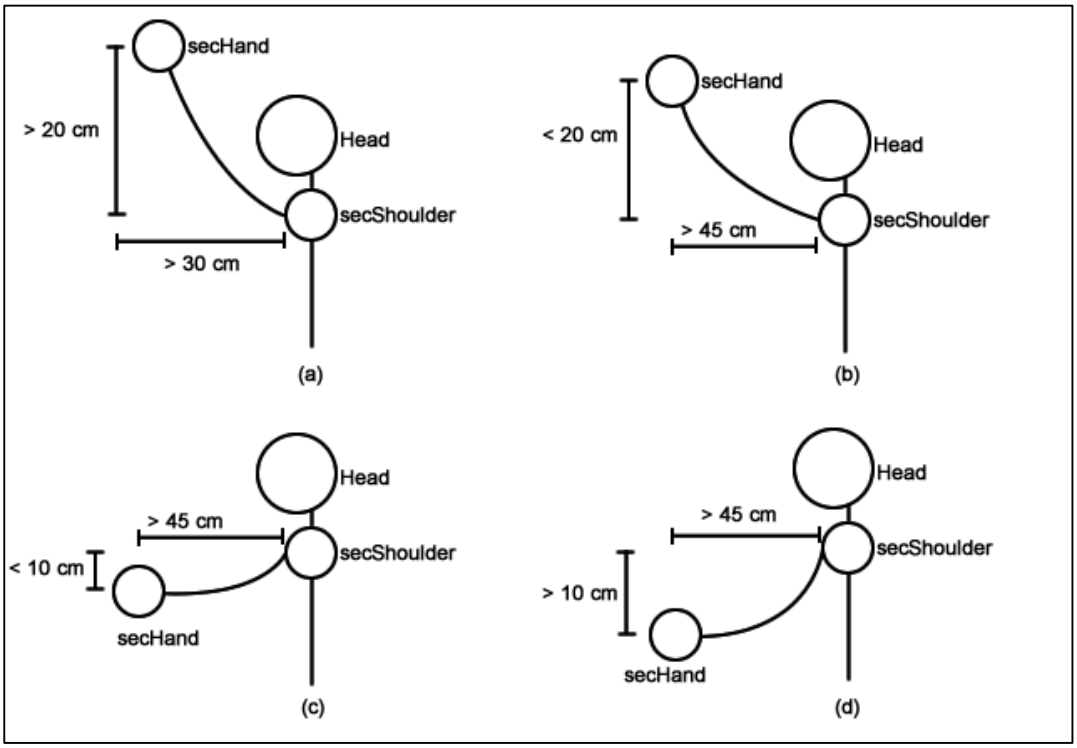

Figura 7 - Distâncias para eventos de clique. a) Clique com o botão direito; b) e c) Clique com o botão do meio; d) Clique com o botão esquerdo

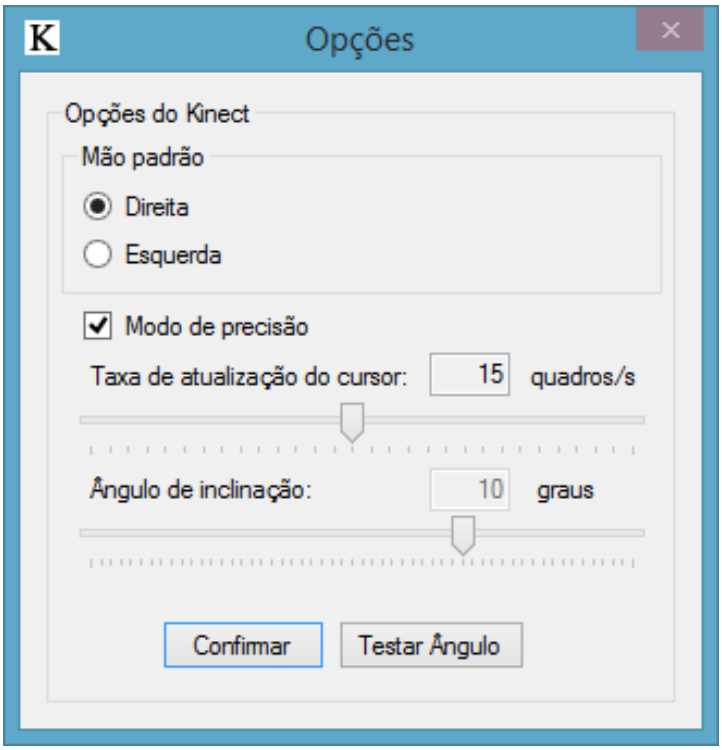

Figura 8 - Janela de opções do sistema

O modo de precisão é uma opção que faz com que o sistema só atualize a posição do cursor do *mouse* se a posição calculada para a tela tiver uma distância de mais de 20 pixels na horizontal ou vertical em relação à posição anterior. Caso esta opção não esteja selecionada, o cursor sempre será atualizado. Esse modo serve como uma forma de manter o cursor parado enquanto o usuário tenta executar um movimento de clique, pois, caso a mão principal se mexa, o que geralmente acontece, é executada uma ação de arrastar.

A opção seguinte do menu é a de exibir a janela de câmera (Figura 9). Ela consiste em uma janela com apenas uma área para a imagem capturada do Kinect e um botão para ser fechada. Quando essa janela é aberta, ela ativa o *stream* de cores do Kinect e, quando os dados do *stream* estiverem prontos, é disparado um evento. Esse evento abre o quadro do *stream*, copia os dados de seus pixels para um *array* de *bytes* e cria uma imagem Bitmap de 640x480 pixels, que é então exibida ao usuário.

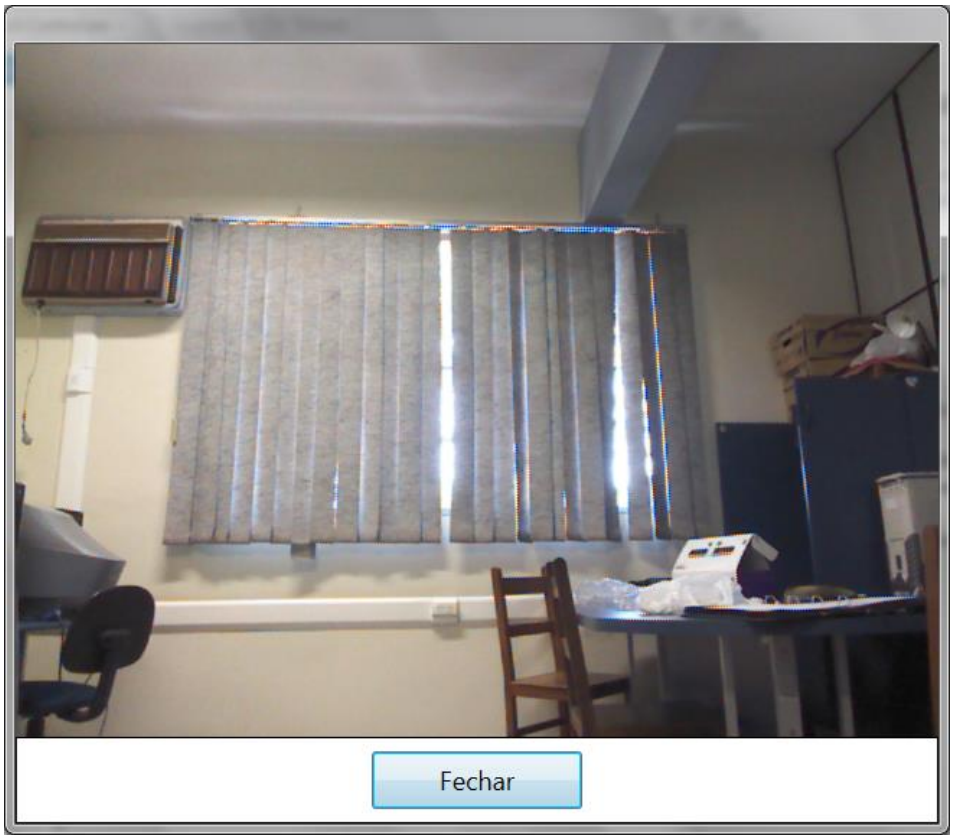

Figura 9 - Janela com a imagem da câmera do Kinect

A terceira opção, por sua vez, inicia um teclado virtual para que seja possível digitar textos com o projeto.

Inicialmente, foi considerado utilizar o teclado virtual nativo do Windows por ser simples e presente em todos os sistemas. Porém, quando isso foi testado, foi visto que o sensor do Kinect não funcionava quando a janela do teclado virtual obtinha o foco do sistema operacional, como se o programa ficasse travado.

Não se sabe ao certo por que isso ocorre; supõe-se que a implementação do teclado utiliza funções de baixo nível que, de alguma forma, impedem que os dados do esqueleto sejam lidos e processados. O mesmo efeito é observado, por exemplo, se o usuário pressiona as teclas Ctrl + Alt + Del enquanto usa o programa.

Para contornar esse problema, é utilizado um programa externo chamado Free Virtual Keyboard (Figura 10), [21] que também implementa um teclado virtual e não possui o problema descrito. Os desenvolvedores desse programa autorizaram seu uso no projeto (Figura 11). O fato de ser um programa externo não implica em problemas de compatibilidade; ele será executado sem problemas desde que seu executável, “FreeVK.exe”, seja mantido na mesma pasta do executável do projeto.

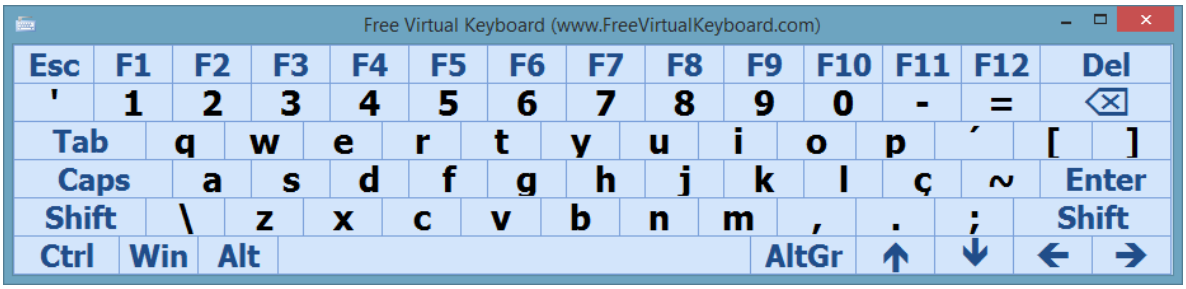

Figura 10 - Programa Free Virtual Keyboard

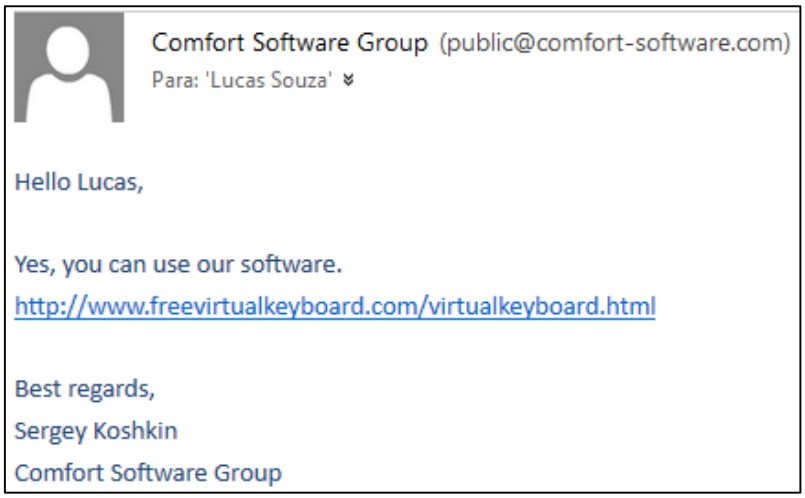

Figura 11 - Autorização de uso do programa Free Virtual Keyboard no projeto

A quarta opção do menu, por fim, encerra o programa, desabilitando os *streams* que estão ativos e liberando os recursos alocados pelo programa.

É importante citar que um problema semelhante ao que acontecia com o teclado virtual do Windows também acontecia a algumas janelas, como a de opções e a de câmera. Neste caso, o processamento de dados do esqueleto cessava de repente quando os botões de fechar, maximizar e minimizar a janela eram pressionados, executando um “clique pela metade” e travando o sistema. Isto é, o sistema entendia que o evento de botão de *mouse* pressionado foi executado, mas não conseguia executar o evento de botão de *mouse* solto. Uma solução para esse problema foi modificar a forma como as janelas eram criadas para impedir que esses botões ficassem disponíveis ao usuário, sendo possível fechar as janelas apenas através de botões dentro delas.

3.3 Mapa de código

Utilizando o próprio Visual Studio, foi criado um arquivo que gera um mapeamento gráfico do código e das dependências entre classes, como um diagrama de componentes de UML. Com esse diagrama, pode-se ter uma visualização mais intuitiva de quais são essas classes e como se relacionam. O diagrama pode ser visto na Figura 12. Detalhes dos códigos-fonte do projeto podem ser vistos no apêndice, no final do documento.

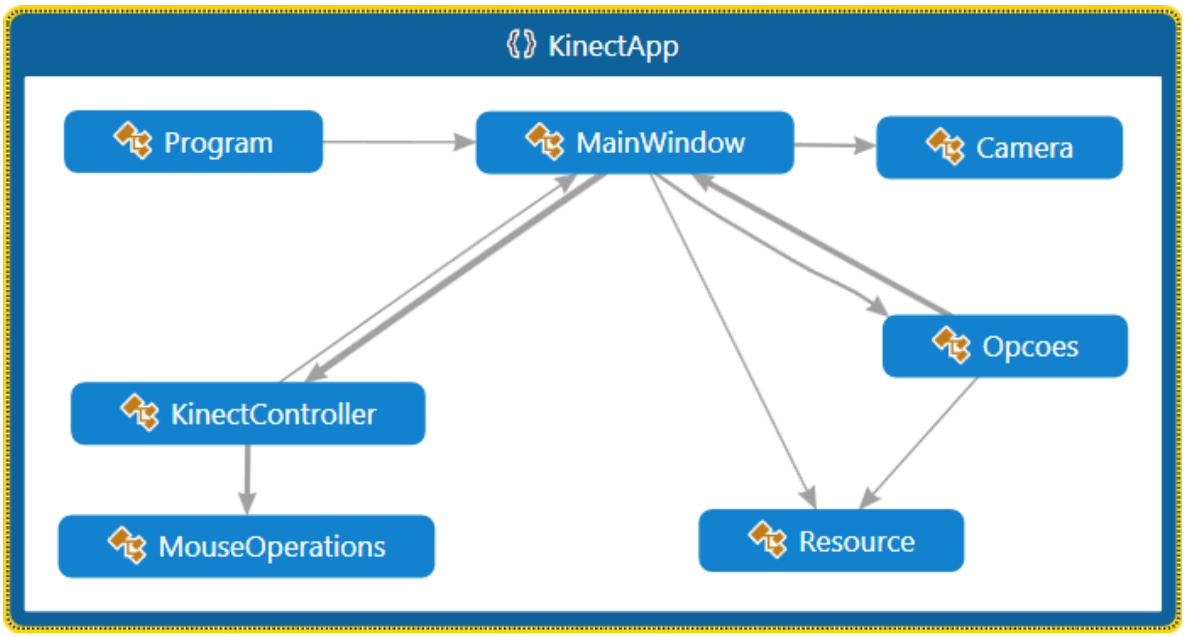

Figura 12 - Mapa de código do sistema

3.4 Considerações finais

O capítulo 3 descreveu as funcionalidades do projeto. Em resumo: é possível utilizar as mãos para simular os movimentos e ações de um *mouse* como com um dispositivo real, e, através do teclado virtual presente no sistema, simular o uso de um teclado, tornando disponíveis todas as funções básicas de uma interface gráfica de usuário através de uma interface natural, a do Kinect.

O capítulo 4 descreve os testes que foram realizados para avaliar se o projeto implementado funciona conforme o esperado e encontrar possíveis erros, discutindo esses resultados em seguida.

4 RESULTADOS E DISCUSSÃO

4.1 Testes de funcionamento

Com o projeto planejado e implementado, foi necessário primeiramente testar seu funcionamento para encontrar possíveis defeitos e realizar possíveis melhorias.

Esta fase de testes foi de certo modo realizada junto com a implementação (seção 3.2.2), pois a cada ideia era necessário testar os novos trechos de código para garantir que o projeto funcionaria corretamente.

Esses testes envolviam apenas o desenvolvedor do projeto e permitiram ter uma estimativa sobre, por exemplo, as melhores distâncias entre a mão secundária e o ombro secundário para as ações de clique, as melhores escalas para converter as informações do Kinect em coordenadas da tela (equações 1 e 2) e as possíveis exceções de *software* que poderiam ocorrer por falhas na implementação.

Para as distâncias entre a mão e o ombro, inicialmente considerava-se que, para os três tipos de clique, as distâncias deveriam ser as mesmas. Foi suposto que deveriam ser de 45 cm, comprimento aproximado do braço do desenvolvedor. As diferenças de altura entre a mão secundária e o ombro secundário eram as mesmas descritas na Figura 7, exceto para o clique com o botão direito, cuja altura deveria ser de 30 cm.

Para os eventos de clique com os botões esquerdo e do meio, essa distância se mostrou boa; para cliques com o botão direito, porém, era muito difícil conseguir manter a mão secundária a mais de 30 cm acima do ombro e a mais de 45 cm à frente. Portanto, a altura foi diminuída para 20 cm e a distância para 30 cm, o que facilitou bastante o uso dos três tipos de clique. Distâncias menores do que essas faziam os cliques serem executados com apenas movimentos simples, sem se ter a intenção.

Também foi determinado que, para um aparelho Kinect localizado sobre uma mesa de aproximadamente 75 cm de altura, o melhor ângulo para que os movimentos sejam bem detectados é de +10°. A altura da mesa e esse ângulo foram então mantidos fixos no desenvolvimento do projeto.

Outro fator relevante é a distância que o usuário deve ficar do sensor. O documento oficial [22] informa que o sensor de profundidade abrange entre 0,8 m e 4,0 m; os testes determinaram que uma boa distância (isto é, em que o sensor consegue determinar precisamente as posições das mãos) deve estar entre 1,0 m e 3,0 m. Distâncias fora desse

intervalo podem causar imprecisões, fazendo, por exemplo, com que o cursor do *mouse* apareça em posições muito diferentes das que deveria estar, problema causado por algum erro de processamento nas imagens capturadas e no cálculo das posições das *Joints*.

4.2 Testes com usuários

Com os testes anteriores, foi obtida uma boa estimativa sobre quais devem ser os valores das variáveis de controle do sistema, sendo possível implementar um sistema que funcione bem e seja de certa forma confortável de ser utilizado.

Porém, como o projeto se trata de uma forma de interação humano-computador, testes simples não são suficientes para se dar um resultado final, pois esse resultado depende de diversos fatores, como quem são os usuários que irão utilizar o projeto, quais são suas características físicas (como altura e comprimento dos braços), possíveis dificuldades motoras, preferências pessoais de uso (como usar ou não o modo de precisão e qual das mãos deve ser a principal), entre outros.

Foram então realizados testes com nove voluntários para avaliar a influência desses fatores e compará-los aos testes realizados na seção 4.1. Dessa forma, é possível ver o que ainda é necessário ser mudado no projeto e obter noções melhores sobre como torná-lo acessível e confortável ao maior número possível de usuários, o que é um dos principais objetivos de uma interface natural.

Para preservar a identidade dos voluntários, eles foram identificados apenas como “Usuário 1”, “Usuário 2” etc.

Os fatores considerados relevantes para esses testes foram: altura do usuário, distância mínima em relação ao Kinect, distância ideal, distância máxima, dificuldade de uso e observações.

Define-se a distância mínima como a menor distância (eixo Z) em que o usuário pode ficar em relação ao Kinect de forma que o sistema funcione corretamente, isto é, o cursor do *mouse* acompanhe corretamente a mão principal do usuário e seja possível executar as ações de cliques corretamente.

A distância ideal é a distância em que o usuário tem o melhor conforto para utilizar o sistema. O cursor acompanha fielmente as posições da mão do usuário, todas as regiões da tela podem ser alcançadas e o usuário possui um conforto e facilidade para as ações de cliques.

De forma análoga à mínima, a distância máxima é a maior distância em que o usuário deve ficar em relação ao Kinect de forma que o sistema funcione corretamente.

A dificuldade de uso são os problemas de usabilidade que o usuário encontrou durante os testes do projeto. Essas dificuldades podem tanto serem informadas pelo usuário quanto observadas pelo desenvolvedor durante os testes. Esses problemas podem ajudar a determinar formas de melhorar a interação do usuário com o sistema, e um constante *feedback* após a implementação do sistema pode ajudar a tornar o *software* cada vez melhor.

Se houvesse algum detalhe que fosse julgado relevante mas não se encaixasse nos fatores anteriores, este seria anotado como uma observação.

O *notebook* foi ligado a um projetor multimídia e sua imagem exibida no próprio monitor e em uma tela na parede. Foi também utilizada uma trena para medir as distâncias em relação ao Kinect, marcando no chão distâncias de 0,5 m utilizando fita adesiva.

As rotinas de testes consistiam em solicitar que os voluntários primeiramente se acostumassem com o sistema. Era explicado seu funcionamento e o que deveriam fazer para usá-lo.

Quando entendiam corretamente, era solicitado que ficassem a diversas distâncias do Kinect para determinar a distância mínima, utilizando as fitas adesivas como referências. O mesmo era feito para determinar as distâncias recomendada e máxima.

Em seguida, era solicitado que o voluntário ficasse à distância recomendada e testasse algumas funcionalidades básicas: mover o cursor por toda a tela, abrir e fechar alguns programas, executar os três tipos de cliques, digitar um texto utilizando o teclado virtual e o programa Microsoft Word e, por fim, conduzir uma apresentação de *slides,* que já estava pronta, executando as ações de clique com as mãos.

As dificuldades que o voluntário apresentava e que eram percebidas pelo desenvolvedor eram anotadas e, após os testes, o voluntário deveria dizer quais outras dificuldades encontrou e quais observações tinha a fazer sobre o projeto, como possíveis melhorias e como achava que outros usuários reagiriam ao sistema.

A Tabela 1 indica os resultados obtidos para as distâncias mínima, ideal e máxima de acordo com cada usuário e sua altura. A Tabela 2 indica as dificuldades que cada usuário teve e suas observações sobre os testes e o projeto.

Além desses resultados, foi encontrado um defeito que ainda não foi possível corrigir: em determinados momentos, o programa não detecta que um clique está sendo realizado e também não reconhece que o cursor ficou sobre algum ícone.

Tabela 1 - Resultados de distâncias nos testes do projeto

Usuário	Altura (m)	Distância mínima (m)	Distância ideal (m)	Distância máxima (m)
Usuário 1	1,65	1,2	1,5	2,0
Usuário 2	1,62	1,2	1,5	1,8
Usuário 3	1,53	1,0	1,5	2,0
Usuário 4	1,77	1,7	2,0	2,8
Usuário 5	1,86	1,5	2,0	3,0
Usuário 6	1,70	1,5	1,7	2,5
Usuário 7	1,89	1,5	2,0	3,0
Usuário 8	1,52	1,2	1,5	1,7
Usuário 9	1,75	1,5	2,0	3,0

Tabela 2 - Resultados de dificuldades e observações

Usuário	Dificuldade de uso	Observações
Usuário 1	Média; dificuldade para executar cliques com o botão direito	
Usuário 2	Alta para cliques; baixa para apresentação de slides	Preferiu não usar o modo de precisão
Usuário 3	Média para navegação *desktop*; baixa para apresentação de slides; alta para digitar	Dificuldade de ver o cursor do *mouse* e usá-lo
Usuário 4	Alta no começo, mas diminuiu com treino; nas bordas da tela, a dificuldade aumenta	Afirmou que um usuário precisa de um bom treino para usar bem o sistema
Usuário 5	Baixa; apresentou grande facilidade para começar, mover o cursor e digitar mesmo à distância mínima	Afirmou que interfaces como a desenvolvida exigem um tempo de adaptação, mas podem ser a evolução
Usuário 6	Média no começo, superada com treino	Achou usar o sistema cansativo; a mão principal tinha que ficar muito alta
Usuário 7	Média no começo; o lado esquerdo da tela apresentou mais defeitos no posicionamento do cursor; alta em digitação	
Usuário 8	Média, superada com treino; alta para encontrar as distâncias e usar o sistema de forma estável	O cursor do *mouse* tremia às vezes, mesmo na distância ideal; achou o tempo para treino cansativo
Usuário 9	Baixa em geral; dificuldade em manter o cursor parado durante um clique	

Uma observação importante a ser mencionada é que, devido à distância a que o usuário deve se manter do Kinect, a tela de um *notebook* (de 14 polegadas, como o utilizado pelo desenvolvedor) se torna muito pequena para um bom uso do projeto. Para um uso melhor, torna-se necessário o uso de um monitor grande (por exemplo, de 23 polegadas) ou de dispositivos de visualização maiores, como uma TV ou um projetor multimídia. Para os testes com voluntários, foi utilizado um projetor que estava disponível no laboratório.

4.3 Discussão

Inicialmente, havia-se suposto que os testes com voluntários (seção 4.2) teriam resultados semelhantes aos testes de funcionamento com o desenvolvedor (seção 4.1), com as mesmas distâncias e as mesmas facilidades e dificuldades encontradas.

Essa suposição se mostrou errada, pois, como pode ser visto nas tabelas 1 e 2, esses resultados variam bastante de acordo com o usuário. Os voluntários com altura maior que 1,70 m apresentaram uma maior facilidade de uso do sistema, em especial nas ações de clique, enquanto que os com altura menor que 1,70 m tiveram mais dificuldades.

As dificuldades envolvendo os cliques podem ser facilmente explicadas: dependendo da altura e porte físico do usuário, os comprimentos de seus braços podem ser muito diferentes. Como no projeto as distâncias usadas para os cliques eram fixas, essa provavelmente foi a maior causa dessas dificuldades.

Como melhoria futura para o projeto, uma solução que poderia diminuir essa dificuldade seria a de, antes da execução do projeto, utilizar-se o Kinect para analisar o corpo da pessoa e determinar medidas relevantes, como sua altura e comprimento dos braços, e utilizar essas medidas como variáveis para os cálculos das distâncias de cliques.

Vale também observar que as distâncias mínima, ideal e máxima também parecem ter uma relação com a altura do usuário, como se pode ver no gráfico da Figura 13.

É difícil determinar qual é exatamente essa relação, pois as variações de altura não são muito significativas. A facilidade de uso também depende de fatores subjetivos, como a experiência que o usuário tem com computadores e com interfaces alternativas à gráfica.

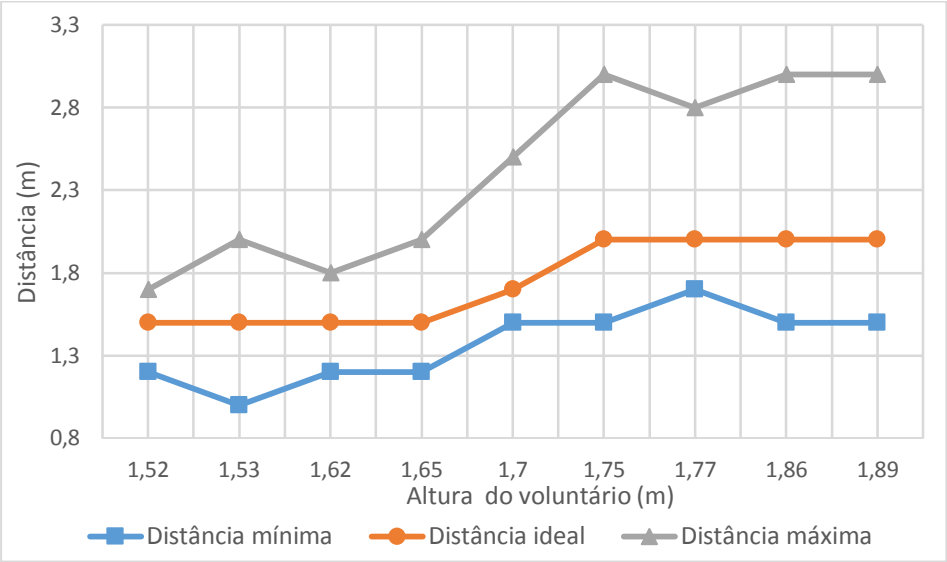

Figura 13 - Gráfico de distâncias (m) em relação à altura (m)

Uma possível resposta é que essa relação seja linear. Considerando que o Kinect possua um ângulo de visão vertical de 43° [23], seus raios de visão devem se expandir de forma linear conforme a Figura 14. Deve, portanto, haver uma equação semelhante a (3), sendo y a distância procurada (mínima, ideal ou máxima) e x a altura do usuário. Os coeficientes a e b podem ser calculados pelas equações (4) e (5). Para usá-las, porém, é recomendado que haja uma quantidade maior de voluntários de forma a tornar as diferenças subjetivas menos influentes e diminuir os erros das equações.

$$y = a.x + b \tag{3}$$

$$a = \frac{\sum_{i=1}^{n} x_i(y_i - \bar{y})}{\sum_{i=1}^{n} x_i(x_i - \bar{x})} \tag{4}$$

$$b = \bar{y} - a\bar{x} \tag{5}$$

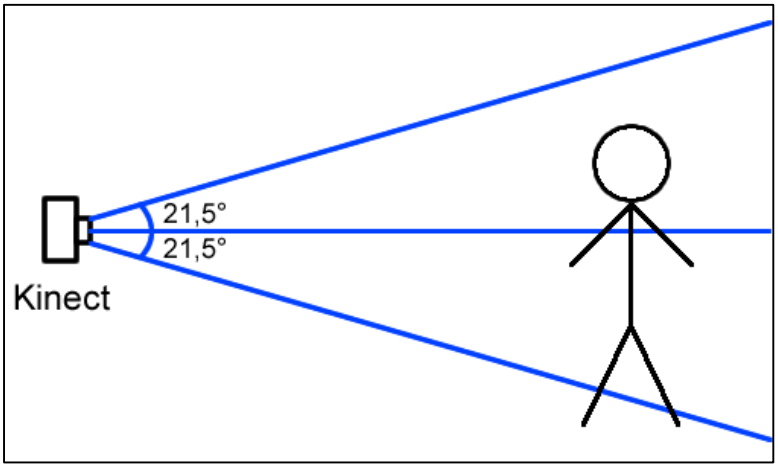

Figura 14 - Raios de visão do Kinect

Essas equações podem ser usadas para duas alternativas de melhoria para o sistema: uma seria informar ao usuário quais são as melhores distâncias a se ficar; a outra, alterar o ângulo de inclinação do Kinect para acompanhar as posições do usuário, por exemplo, com relação à sua cabeça.

Sobre o defeito descrito sobre o sistema não detectar cliques e que o cursor ficou sobre algum ícone, a causa mais provável é que algum clique tenha sido realizado, mas não foi detectado o evento de botão do *mouse* solto. Quando isso acontece, é necessário realizar um clique com o botão esquerdo ou direito de um *mouse* ou *touchpad* físico para que o sistema operacional reconheça esse evento e o programa possa voltar ao normal.

Os conhecimentos até então possuídos e adquiridos pelo desenvolvedor não permitiram fazer um programa focado na usabilidade do usuário. Para um maior conforto e praticidade, sugere-se que, em trabalhos futuros, sejam estudados mais conceitos de interfaces humano-computador e estes sejam aplicados ao projeto.

Como pôde ser observado, a maior parte das dificuldades pode ser superada com um certo treino que, nos testes, demorou apenas cerca de dez minutos. Com um maior tempo de treinamento e um melhor desenvolvimento da interface com o usuário, é possível que esse tipo de interface se torne tão confortável e utilizável quanto as interfaces gráficas utilizadas atualmente.

4.4 Considerações finais

O capítulo 4 descreveu os testes realizados para avaliação do projeto e das capacidades do Kinect para uso em computadores, bem como os resultados obtidos com esses testes e a discussão desses resultados.

No capítulo 5, será feito um breve resumo do trabalho desenvolvido e seus resultados. Serão discutidas também possíveis aplicações do Kinect em trabalhos futuros, além das implicações do projeto aqui apresentado para as pesquisas e trabalhos.

5 CONCLUSÃO

O presente trabalho apresentou o desenvolvimento de uma interface natural de usuário utilizando o dispositivo Microsoft Kinect para Windows.

Essa interface serve como uma camada intermediária entre o usuário e a interface gráfica do computador, baseando-se na movimentação das mãos do usuário para posicionar o cursor do *mouse*, executar ações de cliques e, através de um teclado virtual, emular as teclas como os dispositivos físicos.

No capítulo 3, foi descrita a implementação do sistema e de suas classes e métodos principais.

O capítulo 4 apresenta e discute os testes realizados e os resultados obtidos para o sistema implementado no presente trabalho. Apesar de, na atual fase de desenvolvimento, ainda haver algumas melhorias a serem feitas para um uso confortável do sistema e tratamento de seus erros, os testes com voluntários permitiram uma expectativa otimista de o que pode ser feito com o Kinect.

A maioria dos voluntários utilizados nos testes não apresentou grandes dificuldades para aprender a usá-lo, apenas no início dos testes, sendo essas dificuldades quase totalmente superadas em apenas alguns minutos. É possível, portanto, afirmar que a interface desenvolvida não é difícil de se aprender e, com as melhorias sugeridas na seção 4.3, poderia ser utilizada no lugar dos dispositivos tradicionais, como teclado e *mouse*.

Do ponto de vista do aluno, o projeto foi um grande aprendizado.

Não foi aprofundada a área de processamento de imagens, pois toda essa parte ficou como responsabilidade da API do Kinect.

Por outro lado, foi aprendida uma linguagem de programação nova e a como utilizar o dispositivo Kinect para as mais diversas finalidades, o que, associado à linguagem C#, de alto nível, facilita muito o trabalho de captura de imagens, determinação de profundidades, desenvolvimento das mais diversas estruturas de dados para armazenar essas informações e, sobretudo, utilização do corpo humano como forma de interação. Considerando o interesse e o uso do Kinect tanto na área de jogos como de pesquisas, esses conhecimentos podem ser de grande ajuda.

O dispositivo se mostrou uma ferramenta poderosa e prática, devido ao conjunto de sensores de diversos tipos reunidos em um único *hardware* e os diversos *frameworks* existentes para as mais diversas linguagens de programação.

Ainda assim, possui algumas limitações no que diz respeito à estabilidade na detecção das *Joints*. É necessário às vezes ficar a uma distância muito específica e mover os membros com cuidado para que a detecção fique estável, o que prejudica um pouco o conforto do usuário e torna o sistema cansativo. Uma solução para essas dificuldades deve, portanto, ser implementada por *software*.

Como trabalhos futuros, podem-se imaginar diversos tipos de aplicações.

Por exemplo, o uso do Kinect como uma ferramenta para criação e manipulação de modelos 3D.

Como o projeto apresentado neste trabalho atuava como uma camada intermediária entre o usuário e a interface gráfica já existente nos sistemas operacionais, um outro exemplo de trabalho futuro poderia ser criar algo novo, como iniciar aplicações específicas e interagir com elas através de gestos.

O usuário 5 afirmou que interfaces como a desenvolvida tendem a ser a evolução das interfaces gráficas atuais, o que também foi afirmado por August de los Reyes, gerente de experiência de usuário da Microsoft [5]. Como o Kinect ainda é um produto recente e novas versões dos dispositivos e do SDK ainda estão sendo lançadas, espera-se que o presente trabalho possa servir como base para diversos futuros trabalhos que possam vir, dando um ponto inicial às pesquisas da área de interfaces naturais envolvendo o Kinect.

Ao final deste trabalho, serão apresentados como apêndices os códigos-fonte para os principais componentes do sistema. A maior parte desses componentes é gerada automaticamente pelo Visual Studio, portanto, partes do código serão omitidas e serão escritas observações sobre como modificá-las.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] VELOSO, Thássius. *Microsoft lança SDK do Kinect para Windows 7* [*On-line*]. Disponível em: <https://tecnoblog.net/68297/kinect-sdk-windows-7/>. Acesso em 05/10/2014.

[2] HEWETT, Thomas T., et al. *ACM SIGCHI Curricula for Human-Computer Interaction* [*On-line*]. Disponível em: <http://old.sigchi.org/cdg/cdg2.html#2_1>. Acesso em 05/10/2014.

[3] *COMMAND-LINE interface*. In: WIKIPÉDIA: a enciclopédia livre. Disponível em: <http://en.wikipedia.org/wiki/Command-line_interface>. Acesso em 05/10/2014.

[4] LEVY Jr., Steven. Graphical user interface. In: Encyclopedia Britannica On-line. Disponível em: <http://global.britannica.com/EBchecked/topic/242033/graphical-user-interface-GUI#toc93007>. Acesso em 05/10/2014.

[5] RIEDER, David M. *From GUI to NUI: Microsoft's* Kinect *and the Politics of the (Body as) Interface* [*On-line*]. Disponível em: <http://www.presenttensejournal.org/volume-3/from-gui-to-nui-microsofts-kinect-and-the-politics-of-the-body-as-interface/>. Acesso em 07/10/2014.

[6] *KINECT*. In: WIKIPÉDIA: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Kinect>. Acesso em 10/10/2014.

[7] *KINECT fact sheet*. In: Microsoft. Disponível em: <www.microsoft.com/en-us/news/presskits/xbox/docs/KinectFS.docx>. Acesso em 07/10/2014.

[8] BISHOP, Todd. *Microsoft: Kinect wasn't hacked, USB port left open 'by design'* [*On-line*]. Disponível em: <http://www.bizjournals.com/seattle/blog/techflash/2010/11/microsoft-kinect-not-hacked-left.html>. Acesso em 07/10/2014.

[9] MITCHELL, Richard. *PrimeSense releases open source drivers, middleware that work with Kinect* [*On-line*]. Disponível em: <http://www.joystiq.com/2010/12/10/primesense-releases-open-source-drivers-middleware-for-kinect/>. Acesso em 07/10/2014.

[10] BOYD, E. B. *Microsoft legitimizes hacking of the Kinect* [*On-line*]. Disponível em: <http://www.fastcompany.com/1760493/microsoft-legitimizes-hacking-kinect>. Acesso em 07/10/2014.

[11] GREENE, Jay. *Microsoft debuts Kinect for Windows, commercial SDK* [*On-line*]. Disponível em: <http://www.cnet.com/news/microsoft-debuts-kinect-for-windows-commercial-sdk/>. Acesso em 07/10/2014.

[12] SOUZA, L. J. dos S. *Controle de computadores utilizando interface natural.* 2014. 9 f. Projeto de Iniciação Científica (Graduação em Engenharia de Computação) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos. 2014.

[13] C# Language Specification. In: Microsoft. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=7029>. Acesso em 09/10/2014.

[14] KEITH. *How to simulate Mouse Click in C#?* [*On-line*]. Disponível em: <http://stackoverflow.com/questions/2416748/how-to-simulate-mouse-click-in-c>. Acesso em 09/10/2014.

[15] CASTRO, André. *Kinect SDK 1.5 Parte 1 – Câmera RGB* [*On-line*]. Disponível em: < <http://www.100loop.com/destaque/kinect-sdk-1-5-parte-1-camera-rgb/>. Acesso em 09/10/2014.

[16] RAITEN, Shai. *Kinect – Getting Started – Become The Incredible Hulk* [*On-line*]. Disponível em: <http://blogs.microsoft.co.il/shair/2011/06/17/kinect-getting-started-become-the-incredible-hulk/>. Acesso em 10/10/2014.

[17] WILDMAN. *Programming for Kinect 4 – Kinect App with Skeleton Tracking* [*On-line*]. Disponível em: <http://blog.3dsense.org/programming/programming-for-kinect-4-kinect-app-with-skeleton-tracking/>. Acesso em 12/10/2014.

[18] *Kinect for Windows SDK v1.7*. In: Microsoft. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=36996>. Acesso em 30/10/2014.

[19] *Visual Studio.* In: Microsoft. Disponível em: <http://www.visualstudio.com/>. Acesso em 11/10/2014.

[20] NAIR, Ajith R. *Run a C# application in background* [*On-line*]. Disponível em: <https://social.msdn.microsoft.com/Forums/vstudio/en-US/f4ad839c-82b5-4349-b61f-d7bf86f9580d/run-a-c-application-in-background?forum=csharpgeneral>. Acesso em 12/10/2014.

[21] *Free Virtual Keyboard*. In: *Free Virtual Keyboard*. Disponível em: <http://freevirtualkeyboard.com/>. Acesso em 16/10/2014.

[22] *Kinect Sensor*. In: Microsoft. Disponível em: <http://msdn.microsoft.com/en-us/library/hh438998.aspx>. Acesso em 19/10/2014.

[23] *Kinect for Windows Sensor Components and Specifications.* In: Microsoft. Disponível em: <http://msdn.microsoft.com/en-us/library/jj131033.aspx>. Acesso em 28/10/2014.

[24] *ConvertICO*. In: *ConvertICO*. Disponível em: <http://www.convertico.com/>. Acesso em 29/10/2014.

[25] *Windows Kinect Control*. In: Github. Disponível em: <https://github.com/Hikarikun92/Windows-Kinect-Control>. Acesso em 19/11/2014.

APÊNDICE – IMPLEMENTANDO O PROJETO

Para reproduzir os códigos-fonte deste projeto, conta-se com a IDE Visual Studio para a criação das janelas e classes do sistema. O projeto completo pode ser acessado no Github [25]. É feita uma separação entre a interface gráfica, gerada automaticamente pelo *designer* da IDE, e a funcional, que deve ser especificada pelo usuário.

O foco deste apêndice será na parte funcional. Para as partes gráficas, será apresentado apenas um guia verbal sobre que componentes devem ser utilizados e alguma modificação que possa ser necessária.

Antes de começar a desenvolver a aplicação, é necessário instalar o Kinect for Windows SDK v1.7 [18] através de um instalador simples. Qualquer versão recente do Visual Studio deve funcionar; recomenda-se a versão 2012 ou 2013 com o *framework* .NET versão 4.5. A versão gratuita é a Express, que também deve funcionar.

Deve ser criado um novo projeto em C# do tipo Windows Forms Application, com nome “KinectApp” e nome de solução “Kinect”. Do lado direito da tela, será exibido o Solution Explorer. Na parte de References, o usuário deve clicar com o botão direito e selecionar “Add Reference...”. No menu Assemblies, deve selecionar a opção Extensions e marcar “Microsoft.Kinect”, que deve estar disponível após a instalação do SDK.

Uma janela com nome “Form1” será criada. Ela deve então ser renomeada para MainWindow. Do lado esquerdo, na Toolbox, deve ser arrastado para a janela um NotifyIcon, que deve ser renomeado para “Icone”. Nas propriedades da janela, no canto inferior direito, deve ser acessada a parte de eventos e registrar (duplo clique) eventos nas linhas FormClosed e Load, conforme a Figura 15.

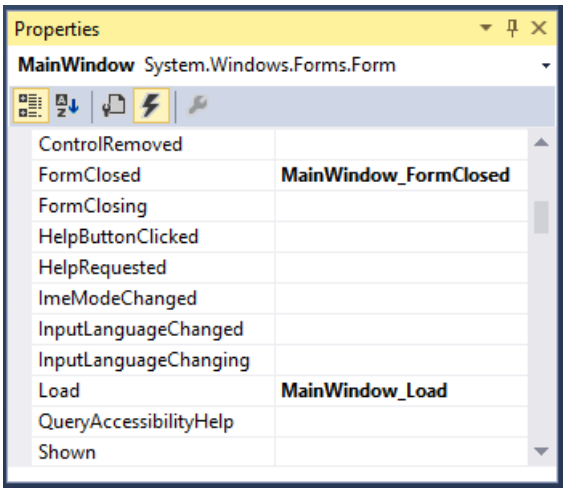

Figura 15 - Registro de eventos da classe MainWindow

O código para a parte funcional da classe MainWindow fica:

[MainWindow.cs]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace KinectApp
{
    public partial class MainWindow : Form
    {
        Opcoes op;
        ContextMenu menu;
        KinectController kc;
        Camera cam;

        public MainWindow()
        {
            InitializeComponent();
            Icone.Text = Resource.AppTitle;
            Icone.ShowBalloonTip(3000, "Iniciando...", "O sensor está sendo
iniciado. Por favor, aguarde.", ToolTipIcon.Info);

            op = new Opcoes(this);
            kc = new KinectController(this);
            cam = new Camera(kc.Sensor);

            menu = new ContextMenu();
            menu.MenuItems.Add(new MenuItem("Opções...", mi1_onClick));
            menu.MenuItems.Add(new MenuItem("Ver câmera", mi2_camera));
            menu.MenuItems.Add(new MenuItem("Mostrar teclado virtual",
mi3_teclado));
            menu.MenuItems.Add(new MenuItem("Encerrar", mi4_encerrar));
            Icone.ContextMenu = menu;

            Icone.ShowBalloonTip(3000, "Pronto!", "O sensor está pronto para uso.",
ToolTipIcon.Info);
        }

        public void mi1_onClick(object sender, EventArgs e)
        {
            op.Show();
        }

        public void mi2_camera(object sender, EventArgs e)
        {
            if (!cam.IsVisible)
            {
                cam.Show();
```

```
            }
        }

        public void mi3_teclado(object sender, EventArgs e)
        {
            try
            {
                Process.Start("FreeVK.exe");
            }
            catch (Win32Exception)
            {
                MessageBox.Show(
                    "Não foi possível encontrar o aplicativo FreeVK.exe. Certifique-
se de que o mesmo se encontra na pasta de instalação do Windows Kinect Control e
tente novamente.",
                    "Erro ao iniciar o teclado virtual"
                    );
            }
        }

        public void mi4_encerrar(object sender, EventArgs e)
        {
            Close();
        }

        public void setDefHand(int hand)
        {
            if (hand == 0)
            {
                kc.DefHand = Microsoft.Kinect.JointType.HandRight;
                kc.DefShoulder = Microsoft.Kinect.JointType.ShoulderRight;
                kc.SecondaryHand = Microsoft.Kinect.JointType.HandLeft;
                kc.SecondaryShoulder = Microsoft.Kinect.JointType.ShoulderLeft;
            }
            else
            {
                kc.DefHand = Microsoft.Kinect.JointType.HandLeft;
                kc.DefShoulder = Microsoft.Kinect.JointType.ShoulderLeft;
                kc.SecondaryHand = Microsoft.Kinect.JointType.HandRight;
                kc.SecondaryShoulder = Microsoft.Kinect.JointType.ShoulderRight;
            }
        }

        public void setPrecision(bool precision)
        {
            kc.Precision = precision;
        }

        public void setSkeletonRefreshRate(int rate)
        {
            kc.setSkeletonRefreshRate(rate);
        }

        public void setSensorAngle(int angle)
        {
            kc.Sensor.ElevationAngle = angle;
        }
```

```
        private void MainWindow_FormClosed(object sender, FormClosedEventArgs e)
        {
            Icone.Dispose();
            cam.Close();
            op.Dispose();
            if (kc != null) { kc.Dispose(); }
        }

        private void MainWindow_Load(object sender, EventArgs e)
        {
            BeginInvoke(new MethodInvoker(delegate
            {
                Hide();
            }));
        }
    }
}
```

Devem ser criadas em seguida duas outras classes: MouseOperations e KinectController. Seus códigos são de acordo com seus arquivos respectivos:

[MouseOperations.cs]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

namespace KinectApp
{
    /**
* http://webdeveloperswall.com/dot-net/simulate-mouse-click-in-csharp
**/
    public class MouseOperations
    {
        [Flags]
        public enum MouseEventFlags
        {
            LeftDown = 0x00000002,
            LeftUp = 0x00000004,
            MiddleDown = 0x00000020,
            MiddleUp = 0x00000040,
            Move = 0x00000001,
            Absolute = 0x00008000,
            RightDown = 0x00000008,
            RightUp = 0x00000010
        }
        [DllImport("user32.dll", EntryPoint = "SetCursorPos")]
        [return: MarshalAs(UnmanagedType.Bool)]
        private static extern bool SetCursorPos(int X, int Y);
        [DllImport("user32.dll")]
        [return: MarshalAs(UnmanagedType.Bool)]
        private static extern bool GetCursorPos(out MousePoint lpMousePoint);
```

```
        [DllImport("user32.dll")]
        private static extern void mouse_event(int dwFlags, int dx, int dy, int
dwData, int dwExtraInfo);

        public static void SetCursorPosition(int X, int Y)
        {
            SetCursorPos(X, Y);
        }
        public static MousePoint GetCursorPosition()
        {
            MousePoint currentMousePoint;
            var gotPoint = GetCursorPos(out currentMousePoint);
            if (!gotPoint) { currentMousePoint = new MousePoint(0, 0); }
            return currentMousePoint;
        }
        public static void MouseEvent(MouseEventFlags value)
        {
            MousePoint position = GetCursorPosition();

            mouse_event((int)value, position.X, position.Y, 0, 0);
        }
        public static void MouseClick()
        {
            MouseEvent(MouseEventFlags.LeftDown);
            MouseEvent(MouseEventFlags.LeftUp);
        }
        [StructLayout(LayoutKind.Sequential)]
        public struct MousePoint
        {
            public int X;
            public int Y;

            public MousePoint(int x, int y)
            {
                X = x;
                Y = x;
            }

        }
    }
}
```

[KinectController.cs]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit.Interaction;
using Microsoft.Kinect.Toolkit.Properties;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Windows.Shapes;
using System.Windows.Input;
```

```
namespace KinectApp
{
    class KinectController
    {
        KinectSensor sensor;
        int posX, posY, nextX, nextY;
        double screenWidth, screenHeight;
        JointType defHand, defShoulder, secondaryHand, secondaryShoulder;
        bool clicked, isMouseLDown, isMouseMDown, isMouseRDown, precision;
        System.Timers.Timer timer;
        int skeletonRefreshRate;

        public KinectSensor Sensor
        {
            get
            {
                return sensor;
            }
        }
        public JointType DefHand
        {
            set { defHand = value; }
        }
        public JointType DefShoulder
        {
            set { defShoulder = value; }
        }
        public JointType SecondaryHand
        {
            set { secondaryHand = value; }
        }
        public JointType SecondaryShoulder
        {
            set { secondaryShoulder = value; }
        }
        public bool Precision
        {
            set { precision = value; }
        }
        [DllImport("User32.dll")]
        private static extern bool SetCursorPos(int X, int Y);

        public KinectController(MainWindow mw)
        {
            bool sucesso = false;
            sensor=null;

            screenWidth = System.Windows.SystemParameters.VirtualScreenWidth;
            screenHeight = System.Windows.SystemParameters.VirtualScreenHeight;

            do
            {
                if (KinectSensor.KinectSensors.Count > 0)
                {
                    sensor = KinectSensor.KinectSensors.FirstOrDefault(s => s.Status
== KinectStatus.Connected || s.Status == KinectStatus.Initializing);
```

```
                    if (sensor != null)
                    {
                        if (sensor.Status == KinectStatus.Connected)
                        {
                            sucesso = true;
                            sensor.SkeletonStream.Enable();
                            defHand = JointType.HandRight;
                            secondaryHand = JointType.HandLeft;
                            defShoulder = JointType.ShoulderRight;
                            secondaryShoulder = JointType.ShoulderLeft;
                            clicked = false;
                            isMouseLDown = false;
                            isMouseMDown = false;
                            isMouseRDown = false;
                            precision = true;
                            posX = 0;
                            posY = 0;
                            timer = new System.Timers.Timer(1000.0 / 15.0);
                            timer.Elapsed += timer_Elapsed;
                            sensor.Start();
                            sensor.ElevationAngle = 10;
                            timer.Start();
                        }
                        else
                        {
                            MessageBox.Show("O sensor foi detectado, porém houve
algum problema ao estabelecer a comunicação. Por favor, tente reconectá-lo e inicie
o programa novamente.\nStatus do sensor: " + sensor.Status.ToString(), "Erro",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                            mw.Close();
                            Environment.Exit(-1);
                        }
                    }
                    else
                    {
                        DialogResult resultado = MessageBox.Show("Nenhum sensor
detectado. Tentar novamente?", "Erro", MessageBoxButtons.YesNo,
MessageBoxIcon.Error);
                        if (resultado == DialogResult.No)
                        {
                            mw.Close();
                            Environment.Exit(-1);
                        }
                    }
                }
                else
                {
                    DialogResult resultado = MessageBox.Show("Nenhum sensor
detectado. Tentar novamente?", "Erro", MessageBoxButtons.YesNo,
MessageBoxIcon.Error);
                    if (resultado == DialogResult.No)
                    {
                        mw.Close();
                        Environment.Exit(-1);
                    }
                }
```

```
            } while (!sucesso);
        }

        public void setSkeletonRefreshRate(int rate)
        {
            timer.Interval = 1000.0 / (double) rate;
        }

        void timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
        {
            SkeletonFrame skelframe = null;
            skelframe = sensor.SkeletonStream.OpenNextFrame((int) timer.Interval);

            try
            {
                if (skelframe != null)
                {
                    Skeleton[] skeletonGroup = new
Skeleton[skelframe.SkeletonArrayLength];
                    skelframe.CopySkeletonDataTo(skeletonGroup);
                    Skeleton sk = (from s in skeletonGroup where s.TrackingState ==
SkeletonTrackingState.Tracked select s).FirstOrDefault();

                    if (sk == null) return;
                    float rightX, rightY;

                    rightX = sk.Joints[defHand].Position.X;
                    rightY = sk.Joints[defHand].Position.Y;

                    nextX = (int)(screenWidth / 2 + rightX * screenWidth * 2.0);
                    nextY = (int)(screenHeight / 2 - rightY * screenHeight * 1.5);

                    if (precision)
                    {
                        if ((Math.Abs(nextX - posX) > 20) || Math.Abs(nextY - posY)
> 20)
                        {
                            posX = nextX;
                            posY = nextY;
                            SetCursorPos(posX, posY);
                        }
                    }
                    else
                    {
                        posX = nextX;
                        posY = nextY;
                        SetCursorPos(posX, posY);
                    }

                    Joint secHand = sk.Joints[secondaryHand];
                    Joint secShoulder = sk.Joints[secondaryShoulder];

                    if (clicked)
                    {
                        if (secShoulder.Position.Z - secHand.Position.Z < 0.3 &&
isMouseRDown)
                        {
```

```
MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.RightUp);
                            isMouseRDown = false;
                            clicked = false;
                        }
                        if (secShoulder.Position.Z - secHand.Position.Z < 0.45)
                        {

                            if (isMouseLDown)
                            {

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftUp);
                                isMouseLDown = false;
                            }
                            if (isMouseMDown)
                            {

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.MiddleUp);
                                isMouseMDown = false;
                            }

                            clicked = false;
                        }
                    }
                    else
                    {
                        if (secShoulder.Position.Z - secHand.Position.Z > 0.3 &&
secHand.Position.Y > secShoulder.Position.Y + 0.2)
                        {

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.RightDown);
                            isMouseRDown = true;
                            clicked = true;
                        }
                        else if (secShoulder.Position.Z - secHand.Position.Z > 0.45)
                        {
                            if (secHand.Position.Y < secShoulder.Position.Y - 0.1)
                            {

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.LeftDown);
                                isMouseLDown = true;
                            }
                            else if (secHand.Position.Y < secShoulder.Position.Y +
0.2)
                            {

MouseOperations.MouseEvent(MouseOperations.MouseEventFlags.MiddleDown);
                                isMouseMDown = true;
                            }
                            clicked = true;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
```

```
                Console.WriteLine(ex.StackTrace);
                Console.WriteLine();
            }
            finally
            {
                if (skelframe != null) skelframe.Dispose();
            }
        }

        public void Dispose()
        {
            if (sensor != null)
            {
                timer.Stop();
                sensor.SkeletonStream.Disable();
                sensor.Stop();
                sensor.Dispose();
            }
        }
    }
}
```

Para a janela de câmera, deve ser adicionado um “User Control (WPF)” com nome Camera. São gerados dois arquivos principais: Camera.xaml e Camera.xaml.cs, cujos códigos se encontram nos arquivos respectivos abaixo.

[Camera.xaml]

```
<Window x:Class="KinectApp.Camera"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d" Width="648.955" ResizeMode="NoResize"
Closing="OnClosing"
             Loaded="OnLoad" Activated="Window_Activated" Height="577.015"
             WindowStartupLocation="CenterScreen" SizeToContent="WidthAndHeight">
    <Grid>
        <Image Name="imgKinect" HorizontalAlignment="Left" Height="480"
VerticalAlignment="Top" Width="640"/>
        <Border BorderBrush="Black" BorderThickness="1" HorizontalAlignment="Left"
Height="60" Margin="0,480,0,0" VerticalAlignment="Top" Width="640">
            <Button x:Name="bFechar" Content="Fechar" Width="146" Margin="246,9"
FontSize="18" Click="Button_Click" IsDefault="True"/>
        </Border>
    </Grid>
</Window>
```

[Camera.xaml.cs]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Threading;
using Microsoft.Kinect;
using System.Runtime.InteropServices;
using System.Windows.Interop;

namespace KinectApp
{
    public partial class Camera : Window
    {
        KinectSensor sensor;

        public Camera(KinectSensor sensor)
        {
            InitializeComponent();
            this.sensor = sensor;
        }

        //Tira os botões de minimizar, maximizar e fechar da janela
        private 63ons tint GWL_STYLE = -16;
        private 63ons tint WS_SYSMENU = 0x80000;
        [DllImport("user32.dll", SetLastError = true)]
        private static extern int GetWindowLong(IntPtr hWnd, int nIndex);
        [DllImport("user32.dll")]
        private static extern int SetWindowLong(IntPtr hWnd, int nIndex, int
dwNewLong);

        private void OnClosing(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            sensor.ColorStream.Disable();
            //Do not close application
            e.Cancel = true;
            Visibility = Visibility.Hidden;
        }

        private void OnLoad(object sender, RoutedEventArgs e)
        {
            sensor.ColorFrameReady += sensor_ColorFrameReady;
            var hwnd = new WindowInteropHelper(this).Handle;
            SetWindowLong(hwnd, GWL_STYLE, GetWindowLong(hwnd, GWL_STYLE) &
~WS_SYSMENU);
        }
```

```
            //Gera uma imagem 640x480 com os dados do stream de cores
        void sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
        {
            using (ColorImageFrame cif = e.OpenColorImageFrame())
            {
                if (cif == null) return;

                byte[] cbytes = new byte[cif.PixelDataLength];
                cif.CopyPixelDataTo(cbytes);

                int stride = cif.Width * 4;

                imgKinect.Source = BitmapImage.Create(640, 480, 96, 96,
PixelFormats.Bgr32, null, cbytes, stride);
            }
        }

        private void Window_Activated(object sender, EventArgs e)
        {
            sensor.ColorStream.Enable();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            sensor.ColorStream.Disable();
            Hide();
        }

    }
}
```

Deve agora ser adicionado um Windows Form de nome Opcoes. Será exibida a interface de criação de janelas. Clicando com o botão direito, há a opção “View Code”, que exibe o código da parte funcional da janela. Este deve ser como no arquivo Opcoes.cs. Em seguida, deve ser aberto o arquivo Opcoes.Designer.cs e seu código de acordo com o respectivo arquivo abaixo.

[Opcoes.cs]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace KinectApp
{
```

```
public partial class Opcoes : Form
{
    MainWindow mw;
    int previousAngle;

    public Opcoes(MainWindow mw)
    {
        InitializeComponent();
        this.mw = mw;
        this.Icon = Resource.Icone;
        this.trackAngle.Value = 10;
        previousAngle = 10;
    }

    private void Opcoes_FormClosing(object sender, FormClosingEventArgs e)
    {
        Hide();
        e.Cancel = true;
    }

    private void buttonConfirm_Click(object sender, EventArgs e)
    {
        if (radioDir.Checked)
        {
            mw.setDefHand(0);
        }
        else
        {
            mw.setDefHand(1);
        }
        Hide();
    }

    private void cbPrecisao_CheckedChanged(object sender, EventArgs e)
    {
        mw.setPrecision(cbPrecisao.Checked);
    }

    private void trackRate_Scroll(object sender, EventArgs e)
    {
        textRate.Text = trackRate.Value.ToString();
        mw.setSkeletonRefreshRate(trackRate.Value);
    }

    private void bAngle_Click(object sender, EventArgs e)
    {
        int angle = trackAngle.Value;
        if (angle != previousAngle)
        {
            previousAngle = angle;
            mw.setSensorAngle(angle);
        }
    }

    private void trackAngle_Scroll(object sender, EventArgs e)
    {
        textAngle.Text = trackAngle.Value.ToString();
```

```
        }
    }
}
```

[Opcoes.Designer.cs]

```
using System.Windows.Forms;

namespace KinectApp
{
    partial class Opcoes
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.label4 = new System.Windows.Forms.Label();
            this.textAngle = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.bAngle = new System.Windows.Forms.Button();
            this.trackAngle = new System.Windows.Forms.TrackBar();
            this.label2 = new System.Windows.Forms.Label();
            this.textRate = new System.Windows.Forms.TextBox();
            this.trackRate = new System.Windows.Forms.TrackBar();
            this.label1 = new System.Windows.Forms.Label();
            this.cbPrecisao = new System.Windows.Forms.CheckBox();
            this.buttonConfirm = new System.Windows.Forms.Button();
            this.chooseDefHand = new System.Windows.Forms.GroupBox();
            this.radioEsq = new System.Windows.Forms.RadioButton();
            this.radioDir = new System.Windows.Forms.RadioButton();
            this.groupBox1.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.trackAngle)).BeginInit();
```

```
((System.ComponentModel.ISupportInitialize)(this.trackRate)).BeginInit();
            this.chooseDefHand.SuspendLayout();
            this.SuspendLayout();
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.label4);
            this.groupBox1.Controls.Add(this.textAngle);
            this.groupBox1.Controls.Add(this.label3);
            this.groupBox1.Controls.Add(this.bAngle);
            this.groupBox1.Controls.Add(this.trackAngle);
            this.groupBox1.Controls.Add(this.label2);
            this.groupBox1.Controls.Add(this.textRate);
            this.groupBox1.Controls.Add(this.trackRate);
            this.groupBox1.Controls.Add(this.label1);
            this.groupBox1.Controls.Add(this.cbPrecisao);
            this.groupBox1.Controls.Add(this.buttonConfirm);
            this.groupBox1.Controls.Add(this.chooseDefHand);
            this.groupBox1.Location = new System.Drawing.Point(12, 12);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(289, 278);
            this.groupBox1.TabIndex = 0;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Opções do Kinect";
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new System.Drawing.Point(223, 173);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(33, 13);
            this.label4.TabIndex = 11;
            this.label4.Text = "graus";
            //
            // textAngle
            //
            this.textAngle.Enabled = false;
            this.textAngle.Location = new System.Drawing.Point(176, 170);
            this.textAngle.Name = "textAngle";
            this.textAngle.ReadOnly = true;
            this.textAngle.Size = new System.Drawing.Size(37, 20);
            this.textAngle.TabIndex = 10;
            this.textAngle.Text = "10";
            this.textAngle.TextAlign =
System.Windows.Forms.HorizontalAlignment.Right;
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(17, 173);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(109, 13);
            this.label3.TabIndex = 9;
            this.label3.Text = "Ângulo de inclinação:";
            //
            // bAngle
```

```
            //
            this.bAngle.Location = new System.Drawing.Point(134, 240);
            this.bAngle.Name = "bAngle";
            this.bAngle.Size = new System.Drawing.Size(86, 23);
            this.bAngle.TabIndex = 8;
            this.bAngle.Text = "Testar Ângulo";
            this.bAngle.UseVisualStyleBackColor = true;
            this.bAngle.Click += new System.EventHandler(this.bAngle_Click);
            //
            // trackAngle
            //
            this.trackAngle.Location = new System.Drawing.Point(7, 190);
            this.trackAngle.Maximum = 27;
            this.trackAngle.Minimum = -27;
            this.trackAngle.Name = "trackAngle";
            this.trackAngle.Size = new System.Drawing.Size(276, 45);
            this.trackAngle.TabIndex = 7;
            this.trackAngle.Scroll += new
System.EventHandler(this.trackAngle_Scroll);
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(220, 121);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(55, 13);
            this.label2.TabIndex = 6;
            this.label2.Text = "quadros/s";
            //
            // textRate
            //
            this.textRate.Location = new System.Drawing.Point(176, 117);
            this.textRate.MaxLength = 2;
            this.textRate.Name = "textRate";
            this.textRate.ReadOnly = true;
            this.textRate.Size = new System.Drawing.Size(37, 20);
            this.textRate.TabIndex = 5;
            this.textRate.Text = "15";
            this.textRate.TextAlign =
System.Windows.Forms.HorizontalAlignment.Right;
            //
            // trackRate
            //
            this.trackRate.Location = new System.Drawing.Point(7, 138);
            this.trackRate.Maximum = 30;
            this.trackRate.Minimum = 1;
            this.trackRate.Name = "trackRate";
            this.trackRate.Size = new System.Drawing.Size(276, 45);
            this.trackRate.TabIndex = 4;
            this.trackRate.Value = 15;
            this.trackRate.Scroll += new System.EventHandler(this.trackRate_Scroll);
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(17, 121);
            this.label1.Name = "label1";
```

```
            this.label1.Size = new System.Drawing.Size(153, 13);
            this.label1.TabIndex = 3;
            this.label1.Text = "Taxa de atualização do cursor:";
            //
            // cbPrecisao
            //
            this.cbPrecisao.AutoSize = true;
            this.cbPrecisao.Checked = true;
            this.cbPrecisao.CheckState = System.Windows.Forms.CheckState.Checked;
            this.cbPrecisao.Location = new System.Drawing.Point(17, 97);
            this.cbPrecisao.Name = "cbPrecisao";
            this.cbPrecisao.Size = new System.Drawing.Size(111, 17);
            this.cbPrecisao.TabIndex = 2;
            this.cbPrecisao.Text = "Modo de precisão";
            this.cbPrecisao.UseVisualStyleBackColor = true;
            this.cbPrecisao.CheckedChanged += new
System.EventHandler(this.cbPrecisao_CheckedChanged);
            //
            // buttonConfirm
            //
            this.buttonConfirm.Location = new System.Drawing.Point(53, 240);
            this.buttonConfirm.Name = "buttonConfirm";
            this.buttonConfirm.Size = new System.Drawing.Size(75, 23);
            this.buttonConfirm.TabIndex = 1;
            this.buttonConfirm.Text = "Confirmar";
            this.buttonConfirm.UseVisualStyleBackColor = true;
            this.buttonConfirm.Click += new
System.EventHandler(this.buttonConfirm_Click);
            //
            // chooseDefHand
            //
            this.chooseDefHand.Controls.Add(this.radioEsq);
            this.chooseDefHand.Controls.Add(this.radioDir);
            this.chooseDefHand.Location = new System.Drawing.Point(7, 20);
            this.chooseDefHand.Name = "chooseDefHand";
            this.chooseDefHand.Size = new System.Drawing.Size(276, 70);
            this.chooseDefHand.TabIndex = 0;
            this.chooseDefHand.TabStop = false;
            this.chooseDefHand.Text = "Mão padrão";
            //
            // radioEsq
            //
            this.radioEsq.AutoSize = true;
            this.radioEsq.Location = new System.Drawing.Point(10, 43);
            this.radioEsq.Name = "radioEsq";
            this.radioEsq.Size = new System.Drawing.Size(70, 17);
            this.radioEsq.TabIndex = 1;
            this.radioEsq.TabStop = true;
            this.radioEsq.Text = "Esquerda";
            this.radioEsq.UseVisualStyleBackColor = true;
            //
            // radioDir
            //
            this.radioDir.AutoSize = true;
            this.radioDir.Location = new System.Drawing.Point(10, 21);
            this.radioDir.Name = "radioDir";
            this.radioDir.Size = new System.Drawing.Size(55, 17);
```

```
            this.radioDir.TabIndex = 0;
            this.radioDir.TabStop = true;
            this.radioDir.Text = "Direita";
            this.radioDir.UseVisualStyleBackColor = true;
            //
            // Opcoes
            //
            this.AcceptButton = this.buttonConfirm;
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(313, 302);
            this.Controls.Add(this.groupBox1);
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "Opcoes";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Opções";
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.Opcoes_FormClosing);
            this.groupBox1.ResumeLayout(false);
            this.groupBox1.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.trackAngle)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackRate)).EndInit();
            this.chooseDefHand.ResumeLayout(false);
            this.chooseDefHand.PerformLayout();
            this.ResumeLayout(false);

        }

        protected override CreateParams CreateParams
        {
            get
            {
                CreateParams myCp = base.CreateParams;
                myCp.ClassStyle = myCp.ClassStyle | 0x200;
                return myCp;
            }
        }

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox chooseDefHand;
        private System.Windows.Forms.RadioButton radioEsq;
        private System.Windows.Forms.RadioButton radioDir;
        private System.Windows.Forms.Button buttonConfirm;
        private System.Windows.Forms.CheckBox cbPrecisao;
        private Label label2;
        private TextBox textRate;
        private TrackBar trackRate;
        private Label label1;
        private Label label4;
        private TextBox textAngle;
        private Label label3;
        private Button bAngle;
        private TrackBar trackAngle;

        #endregion
```

```
    }
}
```

Após isto, é necessária a criação do arquivo de recursos. Deve ser adicionado um item do tipo “Resources File” com o nome Resource.

Na pasta do projeto do Visual Studio (pelo Windows Explorer), deve ser criada uma pasta chamada Resources e dentro dela deve haver um arquivo de ícone, chamado Icone.ico. Nas versões Express do Visual Studio, não há como criar esse tipo de arquivo. Como alternativa, pode ser criada uma imagem 16x16 com extensão png e esta pode ser convertida para o formato ico utilizando o site ConvertICO [24].

Criado o arquivo, ele deve ser aberto pelo Windows Explorer utilizando algum editor de texto (como o Bloco de Notas ou o Notepad++) e seu conteúdo substituído pelo abaixo: [Resource.resx]

```
<?xml version="1.0" encoding="utf-8"?>
<root>
      <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
```

```
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <data name="AppTitle" xml:space="preserve">
    <value>Windows Kinect Control</value>
  </data>
  <assembly alias="System.Windows.Forms" name="System.Windows.Forms,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <data name="Icone" type="System.Resources.ResXFileRef, System.Windows.Forms">
    <value>Resources\Icone.ico;System.Drawing.Icon, System.Drawing, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a</value>
  </data>
</root>
```

No *designer* da janela MainWindow, o NotifyIcon previamente criado deve ser editado com a opção “Choose Icon...”, o qual deve abrir o arquivo Icone.ico da pasta Resources.

No fim do arquivo Resource.Designer.cs, após a declaração do campo AppTitle, deve ser colocado o seguinte trecho de código:

[Resource.Designer.cs - trecho]

```
        /// <summary>
        ///   Looks up a localized resource of type System.Drawing.Icon similar to
(Icon).
        /// </summary>
        internal static System.Drawing.Icon Icone
        {
            get
            {
                object obj = ResourceManager.GetObject("Icone", resourceCulture);
                return ((System.Drawing.Icon)(obj));
            }
       }
```

Por fim, deve ser baixado o arquivo executável FreeVK.exe do site do Free Virtual Keyboard [21]. Esse arquivo deve ficar presente na mesma pasta onde o executável do projeto será gerado. Essa pasta é a “bin”, que possui as subpastas Debug e Release, relacionados aos modos de compilação do Visual Studio. O FreeVK.exe pode ser colocado nessas duas pastas.

Com isso, todos os arquivos necessários para o sistema devem estar criados. Se não houver nenhum problema, o projeto pode ser compilado e executado.