HENRIQUE BORSATTI LISBOA
LUIZ ARTHUR RAMOS DE OLIVEIRA SANTOS

MOVIMENTAÇÃO DE ROBÔ VIRTUAL POR MEIO DE SENSORES DE SOM E MOVIMENTO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Engenheiro Mecatrônico

São Paulo
2012

HENRIQUE BORSATTI LISBOA
LUIZ ARTHUR RAMOS DE OLIVEIRA SANTOS

MOVIMENTAÇÃO DE ROBÔ VIRTUAL POR MEIO DE SENSORES DE SOM E MOVIMENTO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Engenheiro Mecatrônico

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Fabrício Junqueira

São Paulo
2012

FICHA CATALOGRÁFICA

Santos, Luiz Arthur Ramos de Oliveira
Movimentação de robô virtual por meio de sensores de som e movimento / L.A.R.O. Santos, H.B. Lisboa. -- São Paulo, 2012.
77 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Realidade virtual 2.Manufatura 3.Robôs industriais 4.Robôs I.Lisboa, Henrique Borsatti II.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III.t.

Aos meus pais Gilmar e Rosa e às minhas irmãs Paula e Leonarda.

Henrique Borsatti Lisboa

Aos meus pais Joaquim Eugênio e Maria Christina e às minhas irmãs Ana Carolina e Ana Christina.

Luiz Arthur R. O. Santos

AGRADECIMENTOS

Primeiramente agradecemos a nossas famílias pelo auxílio e apoio incondicionais, dedicados em todos os momentos de nossas vidas. Por sempre nos incentivarem a continuar a lutar por nossos ideais e a nunca desistir, independente da situação.

Ao Prof. Dr. Fabrício Junqueira, pelo incentivo, dedicação e orientação despendidos neste trabalho. Ao Sr. Marcosiris que sempre procurou nos auxiliar.

Agradecemos a todos os professores, colegas e amigos que conhecemos ao longo dessa jornada. A todos que, de alguma forma, colaboraram para que chegássemos aqui.

Por fim, agradecemos a Deus.

RESUMO

Para tornar mais eficaz e aproveitável a experiência da Realidade Virtual, buscam-se novas técnicas de modo a tornar a mesma mais interativa (maior influência nos objetos do meio) e intuitiva (facilidade de interação). Sendo assim, a tendência da indústria de Realidade Virtual é a de eliminar ou restringir o uso de dispositivos de entradas mais convencionais como *mouses*, teclados e *joysticks*. A proposta neste trabalho é controlar um robô industrial virtual por meio de um dispositivo de captação de som e movimento. Foi utilizado o periférico Microsoft Kinect, constituído de uma câmera RGB, um sensor de profundidade, quatro microfones e um processador próprio. Como foco do controle foi utilizado e aperfeiçoado um robô virtual previamente desenvolvido. Utilizou-se o movimento do braço do operador para controlar o robô virtual e a câmera e comandos de voz para diversas opções do programa. O movimento gerado pode ser gravado e executado em um simulador desenvolvido. Para tal foi feito uso da ferramenta Microsoft XNA, empregando o C# no ambiente de desenvolvimento Microsoft Visual Studio em conjunto com o Kinect SDK.

Palavras-chave: Realidade Virtual, Manufatura Virtual, Simulação, Robótica, Microsoft Kinect.

ABSTRACT

To make a more effective and profitable experience of Virtual Reality, society is looking up new techniques to make it more interactive (greater influence on the objects in the middle) and intuitive (ease of interaction). Thus, the Virtual Reality industry trend to eliminate or restrict the use of more conventional input devices such as mouse, keyboards and joysticks. The proposal of this work is to control a virtual industrial robot via a sound and movement pickup device. The device used is the Microsoft Kinect peripheral, which consists of a RGB camera, a depth sensor, four microphones and its own processor. As control object was used and perfected a virtual robot previously developed. It was used the arm movement of the operator to control the robot and camera and voice commands for various program functions. The movement detected can be record and executed in the simulator developed. It was developed with Microsoft XNA and C # in the Microsoft Visual Studio environment in conjunction with the Kinect SDK.

Keywords: Virtual Reality, Virtual Manufacturing, Simulation, Robotics, Microsoft Kinect.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

A Realidade Virtual é uma tecnologia de interface avançada entre homem e máquina, com o objetivo de recriar a sensação de realidade para o usuário (GUTIÉRREZ; VEXO; THALMANN, 2008). Sua aplicação na área da manufatura é recente. Pode-se controlar e verificar os mais diversos processos de fabricação, de modo virtual, reduzindo o custo e tempo necessários quando comparado às tecnologias tradicionalmente utilizadas (BANERJEE; ZETU, 2001). Além disso, também podem ser utilizados em ambientes de aprendizado (CHENG; WANG, 2008).

Dentre todas as suas áreas, a realidade virtual não imersiva (com uso de monitores convencionais) é a mais utilizada, devido ao seu baixo custo. Para tornar mais eficaz e aproveitável a experiência, buscam-se novas técnicas de modo a tornar a mesma mais interativa (maior influência nos objetos do meio) e intuitiva (facilidade de interação). Sendo assim, a tendência da indústria de Realidade Virtual é a de eliminar ou restringir o uso de dispositivos de entradas mais convencionais como *mouses*, teclados e *joysticks* (GUTIÉRREZ; VEXO; THALMANN, 2008).

A proposta neste trabalho é controlar um robô industrial virtual por meio de um dispositivo de captação de som e movimento. Foi utilizado o periférico Microsoft Kinect, constituído de uma câmera RGB, um sensor de profundidade, quatro microfones e um processador próprio (MICROSOFT, 2012a).

Como foco do controle foi utilizado e aperfeiçoado um robô virtual previamente desenvolvido em (MIYASHIRO; SUGAWARA, 2011). Utilizou-se o movimento do braço do operador para controlar o robô virtual e a câmera, aliados a comandos de voz para outros procedimentos. O movimento pode ser gravado em forma de código KRL, próprio dos robôs Kuka, e testado no simulador desenvolvido.

Para tal foi utilizado a *framework* Microsoft XNA utilizando o C# no ambiente de desenvolvimento Microsoft Visual Studio em conjunto com o Kinect SDK.

A aplicação do que foi desenvolvido não se limita a ambientes de manufatura virtuais, como pode ser expandida para ambientes reais com o controle humano de forma natural e intuitiva das máquinas no chão de fábrica, quando necessário, ou em missões não tripuladas e em ambientes inóspitos, onde um controle simplificado pode ser de grande valor. Outro uso é na programação por aprendizagem, em

oposição à programação textual tradicional, onde a movimentação do operador pode ser armazenada em forma de código. Também há um grande campo de exploração na área educacional.

2. REVISÃO BIBLIOGRÁFICA

2.1. REALIDADE VIRTUAL

Uma das primeiras menções ao termo Realidade Virtual data da década de 60 nos Estados Unidos, e se trata do Sensorama Simulator (Figura 1). Criado por Morton Heilig, numa tentativa de expandir a experiência cinematográfica de sua época. Ele era constituído de um sistema de vídeo tridimensional, som estéreo, assento vibrante e reclinável, dispersor de aromas (comidas, flores etc.) e ventiladores. Era possível simular uma viagem de bicicleta pelo bairro americano do Brooklyn, sentindo o vento nos cabelos e o trepidar da estrada (RHEINGOLD, 1992).

Figura 1. Pôster de propaganda do Sensorama (PAYATAGOOL, 2008).

Desde essa época os ambientes de realidade virtual evoluíram consideravelmente (KARASEITANIDIS et al., 2006). A Realidade Virtual (RV) é descrita como o conjunto de técnicas de simulação computacional que permitem simular a presença física em ambientes reais e imaginários por meio do uso em conjunto de *hardware*, *software* e sensores (JIANG; FENG, 2006). A RV permite a

navegação e a operação em ambientes tridimensionais, que podem ser semelhantes a ambientes reais e proporcionam experiências comparáveis às reais (GUTIÉRREZ; VEXO; THALMANN, 2008).

As aplicações clássicas de tais conceitos podem ser aplicadas às mais diferentes áreas do conhecimento, com destaque às áreas da educação e treinamento (Ambientes de Aprendizagem Virtual), além do entretenimento. Exemplos são os treinamentos militares e aeronáuticos. Com o rápido desenvolvimento das tecnologias computacionais, tais aplicações se estenderam também à manufatura, medicina (treinamento e tratamento), arte, entre outras (JIANG; FENG, 2006).

Segundo (BURDEA; COIFFET, 2003) as três principais características da Realidade Virtual (RV), conhecidos como os “Três Is” são:

- **Interação** – interação em tempo real. O mundo virtual é capaz de detectar as ações do usuário e promover as modificações necessárias em ambientes e objetos quase que instantaneamente. Uma ramificação importante desse conceito é a interação com outros usuários também em tempo real.
- **Imersão** – a própria interação com o ambiente provoca a sensação de imersão. Ela está relacionada à sensação de estar presente no ambiente e de ser parte da experiência que nele ocorre. Sendo assim está intimamente ligada aos sentidos do corpo humano.
- **Imaginação** – a realidade virtual também é um meio de solução de problemas de engenharia, medicina etc. A eficácia nesse âmbito está intimamente relacionada à imaginação e poder de imaginar coisas não existentes, mas necessárias à formulação e solução do problema proposto.

A interface homem-computador na Realidade Virtual ocorre muitas vezes por meio dos periféricos tradicionais do computador, como *mouse*, teclado e monitor. Para tornar a experiência mais aproveitável ou próxima da real, diversos outros dispositivos podem ser utilizados, de modo a estimular os cinco sentidos humanos.

- **Visão** – historicamente foi um dos primeiros sentido humanos a ser explorado pela realidade virtual. Não se pode afirmar ser o mais relevante sentido humano, mas é normalmente reconhecido como a fonte primária de informação. Muitas vezes é usada para confirmar os outros sentidos (GUTIÉRREZ; VEXO; THALMANN, 2008). Pode ser explorada por monitores

CRT (*Cathode-Ray Tube*), LCD (*Liquid Crystal Display*), Plasma (*Plasma Display*) entre outros. O HDM (*Head Mounted Display*) (Figura 2) trata-se de um sistema mais imersivo e pode utilizar-se das tecnologias de monitores citadas anteriormente. É composto de uma ou duas lentes (para visão estereoscópica) e sensores de posicionamento e inclinação, permitindo ao computador gerar em tempo real as imagens dependendo da posição da cabeça do usuário (GUTIÉRREZ; VEXO; THALMANN, 2008) (BURDEA; COIFFET, 2003). São montadas em capacetes ou óculos especiais e muitas vezes possuem fones de ouvido.

- **Audição** – os sons são de extrema importância para produzir ambientes realísticos. Principalmente o som espacial, que pode ser percebido como proveniente de diferentes localizações (GUTIÉRREZ; VEXO; THALMANN, 2008). Um dos pontos chaves está na gravação e reprodução dos sons. Podem ser utilizados fones de ouvido ou caixas acústicas. Fones de ouvido podem ser caros e incômodos ao usuário, também tornando a experiência individual e difícil de ser compartilhada. Com as caixas acústicas, como o sinal que chega a cada ouvido não pode ser controlado separadamente, é mais difícil controlar a informação sonora espacial que chega ao usuário. Sistemas com múltiplas caixas acústicas são utilizados para simular esses efeitos (GUTIÉRREZ; VEXO; THALMANN, 2008) (BURDEA; COIFFET, 2003).
- **Tato** – são raras as aplicações de RV que exploram a geração de forças para causar a sensação de toque. Sistemas que produzem saídas simplesmente sonoras e visuais são bem menos especializados e baratos (GUTIÉRREZ; VEXO; THALMANN, 2008). Os sistemas podem ser divididos entre aqueles que excitam a pele e aqueles que excitam músculos, tendões e juntas. O *mouse*, teclado, luvas virtuais, detecção de movimento, entre outros não estão relacionados com a sensação de tato, e sim com o conceito de imersão ao tornar a experiência mais ou menos imersiva do ponto de vista psicológico. Contudo, muitas vezes, nesses equipamentos são incorporadas as tecnologias para estimular o tato. Sistemas vibrotáteis constituídos de atuadores podem ser aplicados a luvas, roupas, assentos, etc. Também há sistemas táteis: eletrotátil (produz sensações com pequenas correntes elétricas), térmico (estimuladores térmicos com controle de temperatura) e mecânico (produz deformações na

pele de modo a simular o contato). Além disso, é comum o uso de exoesqueletos (GUTIÉRREZ; VEXO; THALMANN, 2008).

- **Olfato e Paladar** – o desenvolvimento da exploração desses sentidos na RV ainda é muito incipiente. Estudos mostram que a memória e o olfato estão intimamente ligados. Para o olfato são construídos sistemas odorizadores com dispersores de fragrâncias, ventiladores e sistemas de controle. Quanto aos mecanismos simuladores de paladar, poucos exemplos podem ser encontrados, como simuladores de comida e bebida que muitas vezes encontram críticas devido a fatores higiênicos (GUTIÉRREZ; VEXO; THALMANN, 2008).

Figura 2. Diferentes tipos de HDM (GUTIÉRREZ; VEXO; THALMANN, 2008).

A importância relativa de cada sentido varia de aplicação para aplicação. Exemplos serão vistos posteriormente onde esse ponto ficará claro. A arquitetura utilizada normalmente é monolítica, ou seja, otimizada para uma aplicação específica. Sistemas mais gerais e adaptáveis, devido às várias áreas de aplicação da RV, tendem a produzir experiências mais pobres (menos imersivas e iterativas), pois os custos e a complexidade altos são um empecilho nessa arquitetura (GUTIÉRREZ; VEXO; THALMANN, 2008).

A classificação de um sistema de RV é feita levando em consideração as características anteriormente detalhadas. Pode ser feita, por exemplo, uma categorização dos dispositivos visuais em relação ao fator da imersividade. Há sistemas imersivos (HDM) e não-imersivos (monitores). Os sistemas não imersivos, por uso de monitores, são muito populares devido ao baixo custo e facilidade de implementação.

Exemplos de aplicação são vistos nas mais diversas frentes, sendo áreas de grande interesse atual. Destaca-se:

- Construção de mundos virtuais que proporcionam tratamento clínico por meio da comunicação em terapias de grupo. Grupos como esse já estão em funcionamento no ambiente virtual social *Second Life*. Neles o terapeuta pode monitorar remotamente as respostas psicológicas e emocionais dos pacientes e intervir de modo apropriado (GORINI; GAGGIOLI; RIVA, 2007).
- Na área de aprendizado e treinamento vários exemplos podem ser citados. Devido a diversas características da RV (tangibilidade, visibilidade, manipulabilidade) a mesma se torna muito popular nessa área (CHENG; WANG, 2011). Exemplos: sistema de desenvolvimento de habilidades de visualização espacial para estudantes de engenharia (MARTÍN-GUTIÉRREZ et al., 2010), treinamento individual e não centralizado de tripulantes de missões espaciais (ROCHE; KURT; AHRENS, 2010), telepresença (KYBARTAITE; NOUSIAINEN; MALMIVUO, 2010).
- A metodologia tradicional para o desenvolvimento de habilidades técnicas na medicina, como as habilidades cirúrgicas, baseia-se na observação do trabalho de um especialista e na absorção de seus conhecimentos. Contudo, frente aos erros técnicos recorrentes presenciados durante cirurgias, discute-se a eficiência dessa metodologia (GRANTCHAROV, 2008). Um exemplo de Ambiente de Aprendizado Virtual é o de treinamento para cirurgia de substituição de prótese de quadril (PETROLO et al., 2010).
- Em busca de uma melhor competitividade no mercado as empresas devem introduzir novos produtos, mais inovadores e personalizados, de forma cada vez mais rápida. (MAHDJOUB et al., 2010). A Manufatura Virtual permite a prototipagem de novos produtos com custos muito reduzidos, de forma rápida, sem gastos de material, sem parar as máquinas do chão de fábrica e sem

colocar em risco a segurança (KADIR; XU; HAMMERLE, 2011). Além disso, pode-se utilizar os conceitos de RV para ajudar a promover projetos colaborativos entre times de engenheiros (MAHDJOUB et al., 2010) ou na captura computacional dos conhecimentos humanos intrínsecos associados a processos para posterior automação de forma não intrusiva e rápida (SUNG et al., 2009). Devido à importância desse tema ao projeto proposto, uma seção própria foi dedicada ao assunto.

2.2. MANUFATURA VIRTUAL

Para uma empresa de manufatura ter competitividade no mercado internacional não basta o cumprimento dos princípios da manufatura enxuta. A mesma deve ter a capacidade de fornecer soluções completas e com ciclo de vida com custos compatíveis a seus clientes. Há uma grande pressão para aumentar a eficiência no desenvolvimento, operação e uso de recursos de forma aberta e transparente. Com prazos cada vez menores é necessário atender a demanda de clientes de todo o mundo, com um fluxo de informações confiável ente clientes, projetistas, fornecedores e chão de fábrica, sem paralisar o mesmo. Esses são os objetivos principais da manufatura virtual (KOÇ et al., 2005).

Basicamente um Sistema de Manufatura Virtual é constituído de ferramentas de áudio e vídeo que simulam um ambiente de manufatura real. Dessa forma, a RV promove a integração dos diversos processos de manufatura tendo como resultado final a manufatura dentro do próprio computador, que tenta obedecer ao máximo os comportamentos físicos e lógicos da manufatura real. Pode ser a modelagem e simulação da fabricação e montagem de um produto ou da logística associada (contabilidade, compras etc.) entre outros. Ou de todos esses processos integrados (PORTO et al., 2002).

De acordo com (PORTO et al., 2002), a Manufatura Virtual engloba três paradigmas clássicos:

- **Manufatura Virtual Orientada ao Projeto:** está relacionado às diversas alternativas de manufatura e a criação de protótipos virtuais. Busca a otimização do projeto do produto das variáveis principais de projeto (massa, qualidade superficial, rigidez etc.).

- **Manufatura Orientada a Produção:** está relacionada à simulação dos processos pelos quais se dá a manufatura permitindo a escolha entre as diversas alternativas que possam existir. Busca a otimização do processo de manufatura e de suas variáveis de projeto escolhidas (tempo, economia etc.).
- **Manufatura Virtual Orientada ao Controle:** está relacionada à simulação dos processos de controle e otimização dos mesmos, antes de aplicá-los no sistema real.

Podem-se dividir os principais subsistemas da Manufatura Virtual em (PORTO et al., 2002):

- **Planejamento de Produtos e Processos** – foca nas necessidades geométricas e não geométricas do produto, sequência de operações, tempos de processos, etc.
- **Gerenciamento de Recursos** – foca em elaborar *layout* de fábrica, gerenciar máquinas, solicitar serviços, etc.
- **Programação de controle numérico** – foca na programação de centros de usinagem, robôs (montagem, pintura, solda, corte), medição, etc. Pode ser utilizada principalmente na programação por aprendizagem (ensino por demonstração) onde o operador movimenta o braço do robô e as trajetórias são armazenadas (utilizada em técnicas de pintura, soldagem e paletização). Também pode ser utilizada na programação por ensino de pontos, sendo as vantagens do método empregado, entretanto, menores.
- **Validações** – foca na validação de aceitação, de ergonomia, de projeto, de campo de visão, de programação, de processos, etc.
- **Simulação** – foca na simulação de sistemas de manufatura, de processos de manufatura, de elementos finitos, de sistemas mecânicos, treinamento de pessoas, implementação de novas ideias, etc.

Os principais aplicações e benefícios provenientes dessas tecnologias também podem ser enumerados, a saber:

- Concepção de novos produtos, onde é reduzindo o número de testes e interrupção da linha de produção, diminuindo os custos e reduzindo o tempo antes necessário (KADIR; XU; HAMMERLE, 2011). Testar protótipos e localizar

possíveis erros ou ineficiências. Planejar processos de modo a organizá-los logicamente e eficazmente (BANERJEE; ZETU, 2001) (SOUZA; SACCO; PORTO, 2006).

- Realização de processos considerados de risco humano ou patrimonial com menor ou nenhuma preocupação com segurança. Situações podem ser simuladas e possíveis situações de risco podem ser identificadas e evitadas no mundo real (POULIQUEN et al., 2007).
- Possibilidade de projetos colaborativos e flexíveis, com a execução de atividades em paralelo por diferentes grupos (MAHDJOUB et al., 2010). A telepresença e o contato direto com fornecedores, outros times e clientes, em tempo real, de modo a atender suas necessidades (CHENG; BATEMAN, 2008).
- Treinamento de times, testes de ergonomia, posição do operador, campo de visão, segurança (Figura 3) (MAHDJOUB et al., 2010), e captação computacional do conhecimento de processos, armazenamento e análise das ações individuais do projetista ou operador de modo a poder automatizá-las (SUNG et al., 2009).

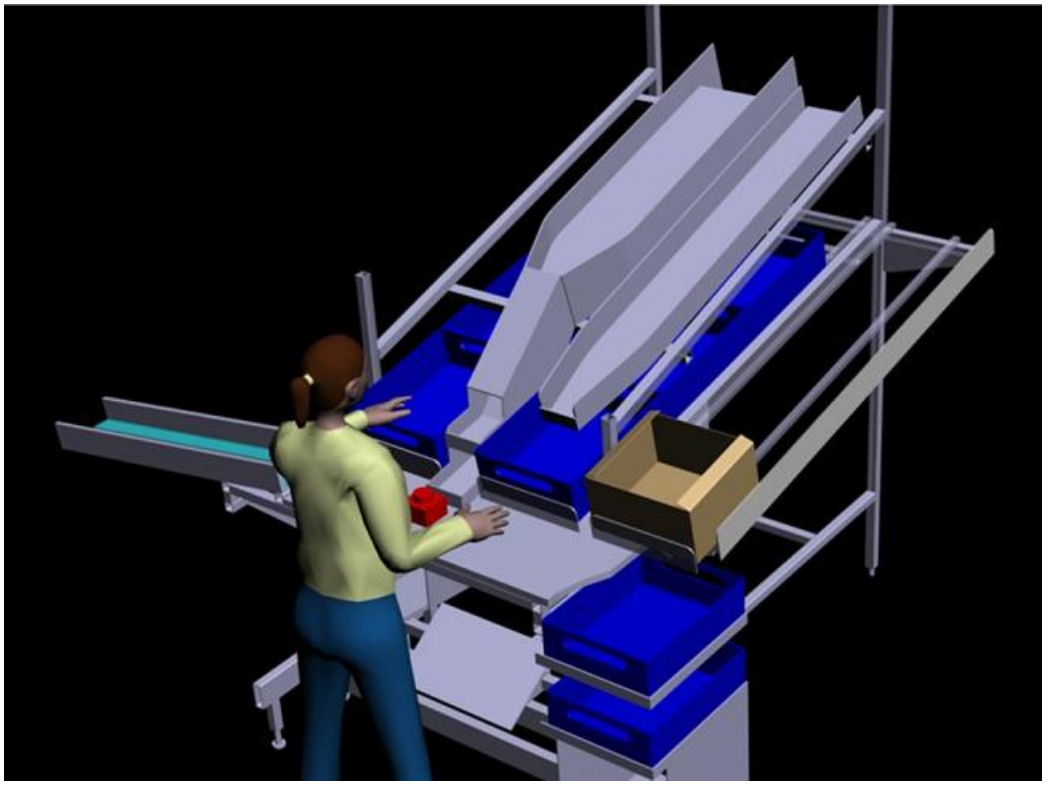

Figura 3. Simulação humana em ambiente virtual (MAHDJOUB et al., 2010).

Como detalhado anteriormente, a implementação de uma realidade mais imersiva conta com diversas vantagens, em especial na área de aprendizado, pois quanto mais próxima da realidade, mais aproveitáveis são seus resultados (SEABRA; SANTOS, 2005). Detalha-se a seguir algumas áreas e possíveis aplicações do trabalho a ser desenvolvido, onde a aproximação adotada (detecção de movimentos) poderia trazer várias vantagens. A saber:

- Educação: um exemplo é o desenvolvimento de habilidades de visualização espacial para estudantes (BRAGA, 2001) de Engenharia, Arquitetura, Desenho e afins. A Manipulação intuitiva em ambientes tridimensionais de formas geométricas poderia ajudar o aluno a aprender a se comunicar com representações gráficas (cognição espacial). Como ferramenta em aulas de Geometria, auxiliaria a compreensão de formas a partir de suas vistas principais, construir vistas particulares e toda a abstração e raciocínio ligado a essa atividades (SEABRA; SANTOS, 2005).
- Treinamento: pode ser utilizado no treinamento da execução de atividades (montagem, uso de maquinário etc.) de forma barata (sem gasto de material ou necessidade e utilização da máquina real), mais próxima da realidade (facilidade de aprendizado) e segura (especialmente importante em processos que lidam com materiais ou equipamentos de risco) (ROCHE; KURT; AHRENS, 2010).
- Inspeção: uso na inspeção de peças geradas pelo computador de modo a verificar se as mesmas atendem as especificações de projeto, montagem (KADIR; XU; HAMMERLE, 2011).
- Validação: uma possível aplicação é a de validação de ergonomia (localização e disposição dos controles de uma máquina, posição do operador etc.) e campo de visão (MAHDJOUB et al., 2010).
- Captação: captação computacional do conhecimento de processos, armazenamento e análise das ações individuais do projetista ou operador de modo a poder automatizá-las (SUNG et al., 2009). Um exemplo é na programação por aprendizagem, onde o operador determina, manualmente, a trajetória a ser percorrida pelo robô.

2.3. MICROSOFT XNA

Microsoft XNA é um conjunto de ferramentas que compõe um ambiente de desenvolvimento, com o objetivo de facilitar a produção de jogos e aplicativos capazes de funcionar no Xbox360 e Windows. O ambiente é capaz de auxiliar no processamento de áudio e vídeo, interação com periféricos, trabalhar com gráficos em 3D, etc.

O Game Studio é um ambiente integrado ao XNA focado no desenvolvimento de jogos, com aplicações mais voltadas para a área gráfica e interação com o usuário. Para trabalhar neste sistema é usado o C# (C-Sharp), linguagem de programação orientada a objetos. Por ser semelhante à linguagem de programação Java, altamente disseminada entre desenvolvedores, a aprendizagem costuma ser mais rápida se comparada a de outros ambientes de desenvolvimento (WU et al., 2010).

2.4. MICROSOFT KINECT

O dispositivo Microsoft Kinect (MICROSOFT, 2012a) é um sensor lançado pela Microsoft em novembro de 2010, com o intuito de servir como acessório para o seu console Xbox360 (MICROSOFT, 2012c). Sendo um dispositivo inovador e de baixo custo (US$150), suas vendas foram expressivas desde seu lançamento, tendo uma base instalada de 18 milhões de unidades mundialmente (MICROSOFT, 2012d). Devido ao seu sucesso, a Microsoft e a comunidade independente de usuários presta suporte ao trabalho nesta ferramenta, facilitando a inserção de novos desenvolvedores por meio de ferramentas intuitivas e ambientes de programação bem estruturados, como o Kinect SDK (MICROSOFT, 2012b). O Kinect SDK (MICROSOFT, 2012b) é um conjunto de ferramentas para facilitar o desenvolvimento para o Kinect (MICROSOFT, 2012a). Este ambiente possui diversos exemplos mostrando as funcionalidades do aparelho, possibilitando um aprendizado progressivo e didático.

Sendo composto por um arranjo de sensores, o Kinect (Figura4) apresenta uma câmera RGB de resolução 640x480 pixels a 30 frames por segundo, câmera com sensor de profundidade IR (*infraRed* - infra vermelho) com 11-bit de resolução e mesma especificação da imagem, um conjunto de microfones capaz de detectar a posição da fonte sonora e motores para movimentar a base da estrutura. Os

sensores possuem uma faixa de atuação de 0.7 - 6 metros de distância do aparelho, com um ângulo de visão de 57^{o} na horizontal e 43^{o} na vertical.

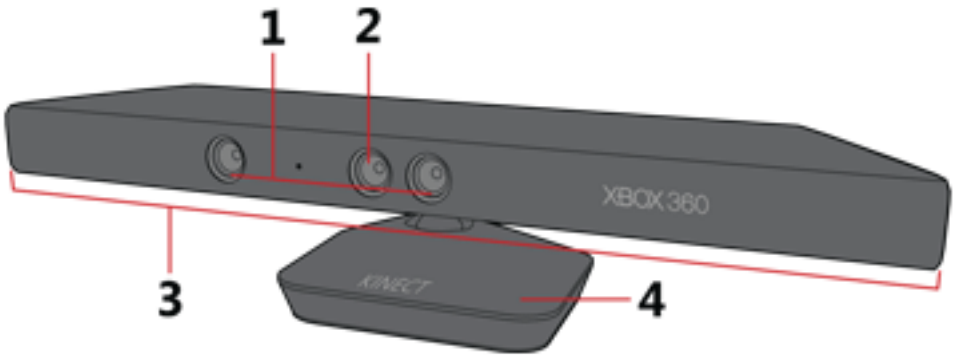

Figura 4. Kinect com seus sensores; (1) Sensores de profundidade, (2) Câmera RGB, (3) Microfones, (4) Base móvel. (MICROSOFT, 2012a).

Um dos grandes diferenciais deste equipamento é o sensor de profundidade, conhecido como *deph camera*. Ele determina a distância de cada pixel da imagem ao projetar um facho de pontos infravermelhos, que são deslocados ao atingir uma superfície opaca, possibilitando o cálculo da profundidade pela câmera sensível a infravermelho (Figura 5) (NOONAM et al., 2011). Esta funcionalidade possibilita um reconhecimento mais detalhado do ambiente, já tendo sido explorada por outros trabalhos (TONG et al., 2012) (RUCHANURUCKS; COUNDOUL, 2011) (STOWERS; HAYES; BRANBRIDGE-SMITH, 2011). O sistema é projetado para ser utilizado em ambientes internos, pois com a interferência da luz solar de ambientes externos (que também possui componentes infravermelhos) o sensor do Kinect não detecta corretamente o seu facho emitido.

Trabalhando em cima de seus sensores, uma das funcionalidades de grande utilidade que se consegue extrair do Kinect é a reconstrução do esqueleto de uma pessoa (Figura 6), possibilitando o conhecimento aproximado da posição de cada membro de seu corpo e detectando até seis pessoas simultaneamente. Por meio desse posicionamento corporal, integrado no Kinect SDK, é possível realizar diversas interações homem-máquina sem a necessidade de sensores acoplados, luvas, capacetes, etc. Sem esta dependência de outros equipamentos, o sistema

pode se tornar mais acessível e de fácil adaptação, por ser intuitivo e direto. Esta funcionalidade já foi explorada em outros trabalhos nas mais diversas áreas, como: auxílio na reabilitação de deficientes (BÓ; HAYASHIBE; POIGNET, 2011) ou na análise do comportamento de crianças (YU et al., 2011).

Figura 5. Demonstração dos resultados obtidos pelo sensor de profundidade (JANOCH et al., 2011).

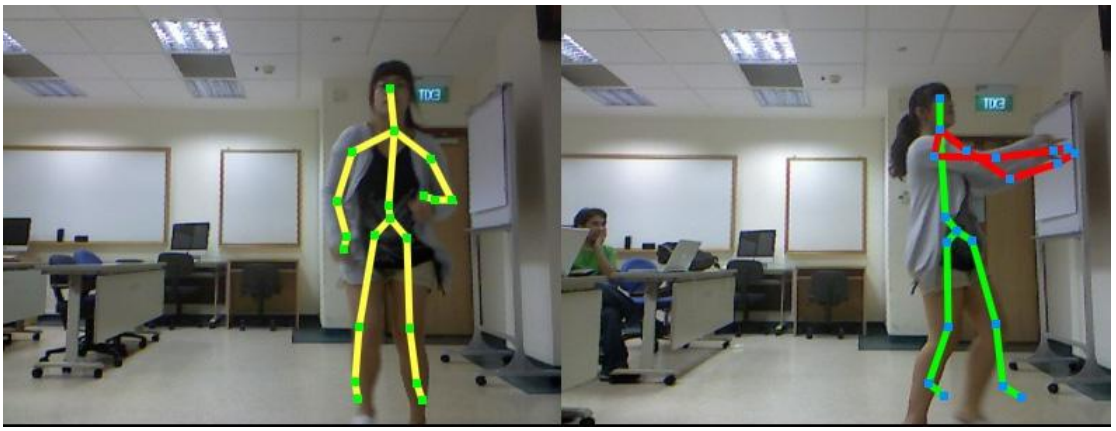

Figura 6. Utilizando a câmera RGB e sensor de profundidade para calcular a posição de uma pessoa, reconstruindo seu esqueleto (TONG et al., 2012).

Para a reconstrução do esqueleto humano, o código se baseia na segmentação do corpo e na definição de *joints* (Figura 7), que estão nos extremos de cada segmento. Os sensores são responsáveis por fornecer as coordenadas das

20 juntas: cabeça, pescoço, ombro esquerdo e direto, cotovelo esquerdo e direito, punho esquerdo e direito, mão esquerda e direita, espinha, centro do quadril, quadril esquerdo e direito, joelho esquerdo e direito, calcanhar esquerdo e direito e pé esquerdo e direito. Baseado na definição de que cada segmento possui uma distância fixa na anatomia humana, é possível estimar algumas juntas que podem não ter sido captadas ou que estão ocultas na imagem. Para determinar estes pontos já foram realizados alguns estudos (STRAKA et al., 2011), com diversas aproximações para tal problema.

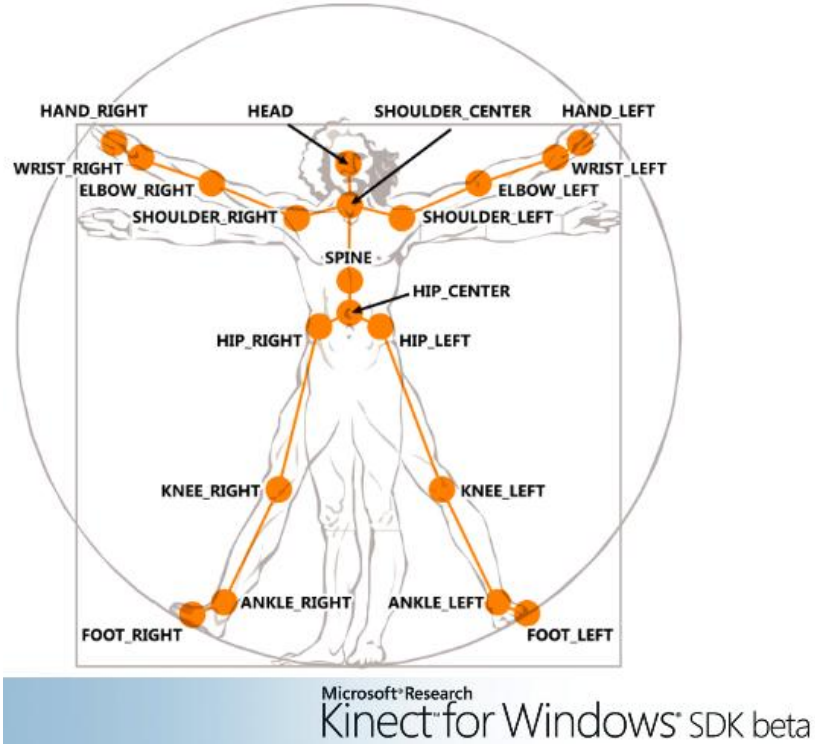

Figura 7. Modelo de esqueleto aproximado pelo Kinect, com 20 juntas (MICROSOFT, 2011).

A captura de posição livre de contatos, por captura de imagem e leitura de sensores externos, possui diversas vantagens frente aos métodos convencionais de aquisição de dados, como acelerômetros, lasers, etc. Contudo, para legitimar o seu uso é preciso conhecer a precisão dos dados obtidos, e validar a medição. Para a reconstrução do esqueleto humano, há estudos (STRAKA et al., 2011) que fazem a reconstrução usando algoritmos customizados e que conseguem delimitar os erros do resultado em diversas situações dinâmicas. Em geral, para aplicações de rastreamento, os dados devem ser adquiridos entre 1 e 3 metros do sensor. Em maiores distâncias a qualidade dos dados é prejudicada pelos ruídos e baixa resolução das medidas de profundidade, resolução esta de 7 cm à distância de 5 m

(KHOSHELHAM, 2011). Definindo uma confiabilidade de 50 mm, cerca de metade das juntas podem ser consideradas corretas, e para 100 mm de confiabilidade, 95% das juntas estão corretas (Figura 8). Estes valores podem ser melhorados para movimentos suaves e com o operador no intervalo ótimo de proximidade dos sensores, sendo os testes realizados em todo intervalo de detecção do dispositivo (STRAKA et al., 2011).

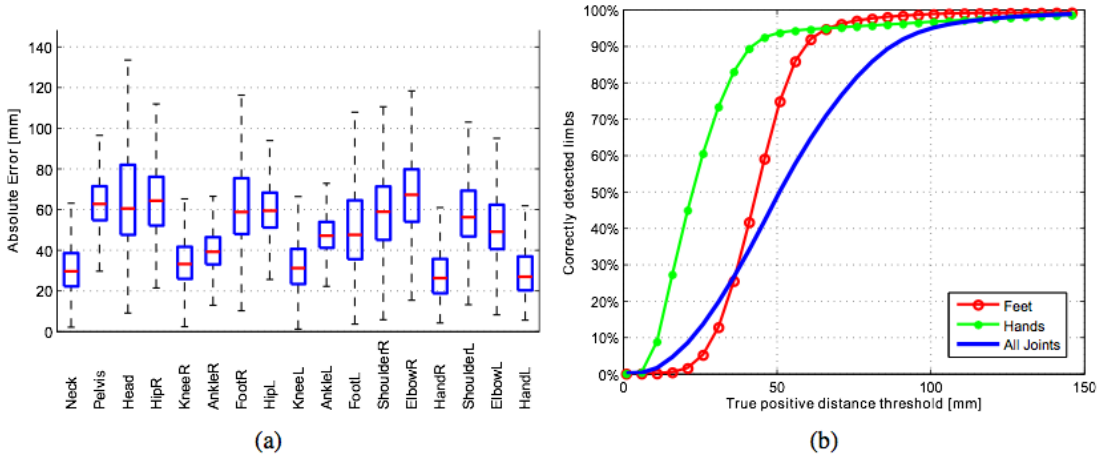

Figura 8. Distância entre pontos da junta estimados e dados reais (a), Número de juntas corretas em relação à variação da confiabilidade (b) (STRAKA et al., 2011).

2.5. UML

A necessidade de modelar sistemas esta presente no cotidiano de diversas áreas do conhecimento, pois é uma técnica fundamental para visualizar e facilitar o desenvolvimento de sistemas complexos. Para que isto seja possível, podem-se utilizar artifícios gráficos e ferramentas que permitem um contato mais amigável ao usuário, fazendo uso de artefatos e componentes que representam o que é relevante para o estudo.

A *Unified Modeling Language* (UML) *é* uma linguagem utilizada para o desenvolvimento de *software*, com o objetivo de facilitar a visualização, especificação e construção de um produto (VARGAS, 2008). Sozinha, a UML não soluciona problemas, e sim auxilia no processo de criação, proporcionando um entendimento melhor do que está sendo feito e tornando ideias, relações e conceitos de um projeto mais acessíveis a todos os membros envolvidos.

Por meio das diversas ferramentas que a UML proporciona é possível diagnosticar problemas e soluções de forma preventiva. Estas rotinas de constante melhoramento não só agradam os projetistas, mas também ajudam na formação de uma documentação do andamento do projeto.

A UML faz o uso de diagramas como principal ferramenta gráfica de trabalho (Figura 9). Entre eles, é possível destacar alguns dos mais utilizados:

- Diagrama de casos de uso: descreve as funcionalidades de um sistema. Esse diagrama é constituído de atores, externos ao sistema, e "casos de uso" que representam as funcionalidades. Esta ferramenta é mais utilizada para modelar os requisitos do sistema (Figura 9).
- Diagrama de classes: este diagrama se aproxima da estrutura de código de um programa. Ele indica as classes, seus atributos, métodos e o relacionamento entre estas classes, sendo um modelo próprio para especificações orientadas a objetos.
- Diagrama de componentes: tem por objetivo indicar os componentes de um *software* e de que forma eles se relacionam.
- Diagrama de sequência: representa a sequência de processos no programa. Ele determina a sequência de comportamento do programa de forma simples e lógica.

Para conseguir atingir seus resultados, a UML permite visualizar as seguintes faces do sistema (Figura 10) (QUEIROZ, 2008):

- Estrutural: parte estática, que constitui dos elementos básicos do sistema (diagramas de classes, objetos, componentes ou implantação).
- Comportamental: parte dinâmica, cuja função é relacionar os elementos do sistema e ditar relacionamentos e a forma com a qual eles devem interagir (diagramas de casos de uso (Figura 11), sequência, atividades, estados ou colaboração).

Figura 9. Diagramas usados pela UML para criação do modelo de estudo (POLLYSOFT, 2012).

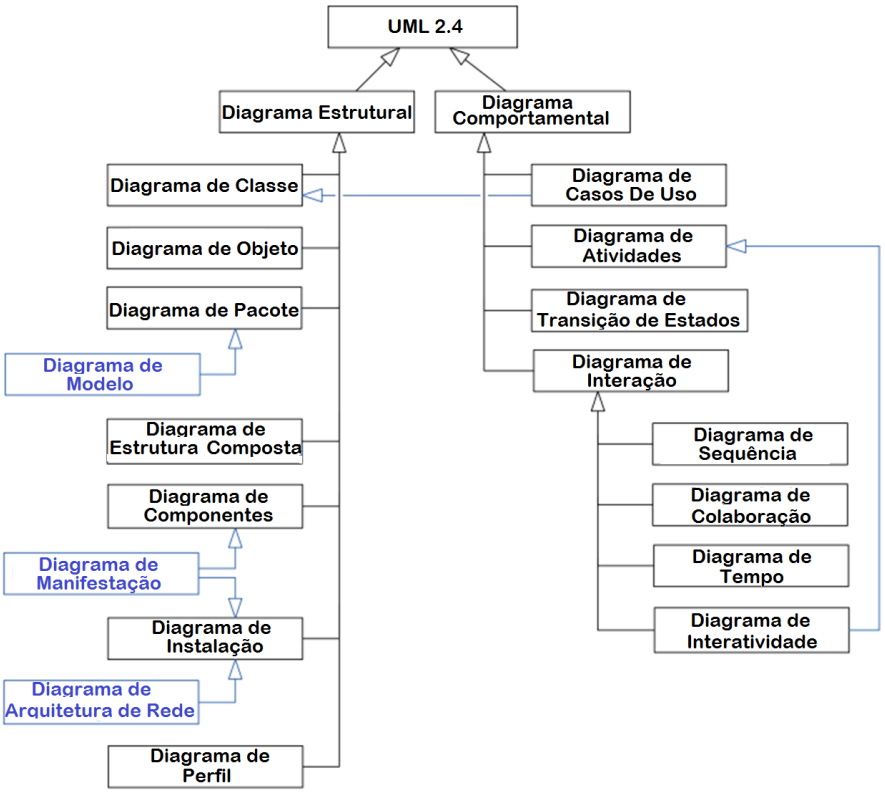

Figura 10. Classificação dos diagramas de acordo com a UML 2.4 (UML-DIAGRAM, 2012).

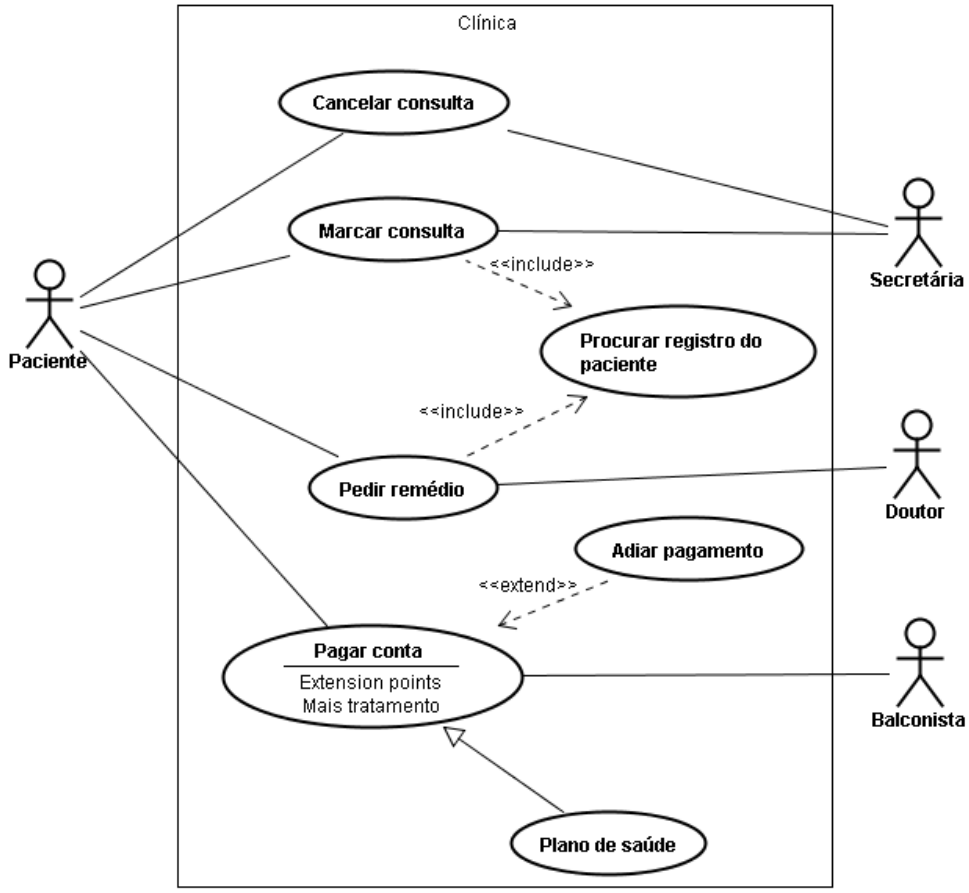

Figura 11. Exemplo de diagrama da UML, o diagrama de casos de uso (BEZERRA, 2006).

3. DEFINIÇÃO DO PROJETO

O objetivo é a construção de um ambiente de Manufatura Virtual mais imersivo, interativo e intuitivo, proporcionando uma experiência mais próxima possível da real.

Para tal foi projetado um *software* de simulação de um sistema de manufatura virtual onde os dispositivos de entrada convencionais (*mouse*, teclado ou *joystick*) foram parcialmente eliminados, sendo substituídos por sensores de som e movimento não intrusivos (que não interferem com o que for medido, ou seja, o usuário). Foi utilizado e aperfeiçoado um ambiente de manufatura virtual previamente implementado constituído de um braço robótico industrial (MIYASHIRO; SUGAWARA, 2011). Como sensor foi utilizado o Microsoft Kinect (MICROSOFT, 2012a).

Para manipulação do ambiente são utilizados os movimentos do usuário para movimentar o braço e a câmera. Os comandos de voz são utilizados em caráter secundário, de modo a auxiliar a utilização das funções implementadas, como a detecção e gravação de movimento. Na Figura 12 é mostrado o protótipo visual inicial de uma configuração possível da tela.

As principais funcionalidades do *software* são: detecção do movimento, movimentação do robô, cálculo de cinemática inversa, detecção de colisão e geração e interpretação de código de programação do robô.

Iniciando o projeto do *software* especificaram-se inicialmente os requisitos do mesmo. Podem-se dividir os requisitos em funcionais e não funcionais. A saber:

- Requisitos Funcionais: descreve o quê o sistema deve fazer, ou seja, as funções necessárias para atender os objetivos:
  - Movimentar ponta da ferramenta;
  - Movimentar câmera;
  - Identificar Movimento;
  - Identificar Fala;
  - Gerar código de programação do robô;
  - Executar código de programação do robô;
  - Calcular Cinemática Inversa;
  - Detectar colisão;

- Exibir tela da simulação.

- Requisitos Não Funcionais: estão relacionados às características do *software* (qualidade, segurança, desempenho, etc.):
  - Tempo de resposta baixo para movimentação em tempo real;
  - Movimentação suave apesar dos ruídos do movimento (filtro).
  - Interface amigável e intuitiva;
  - Imagens fluidas e agradáveis (resolução, fps, etc.).

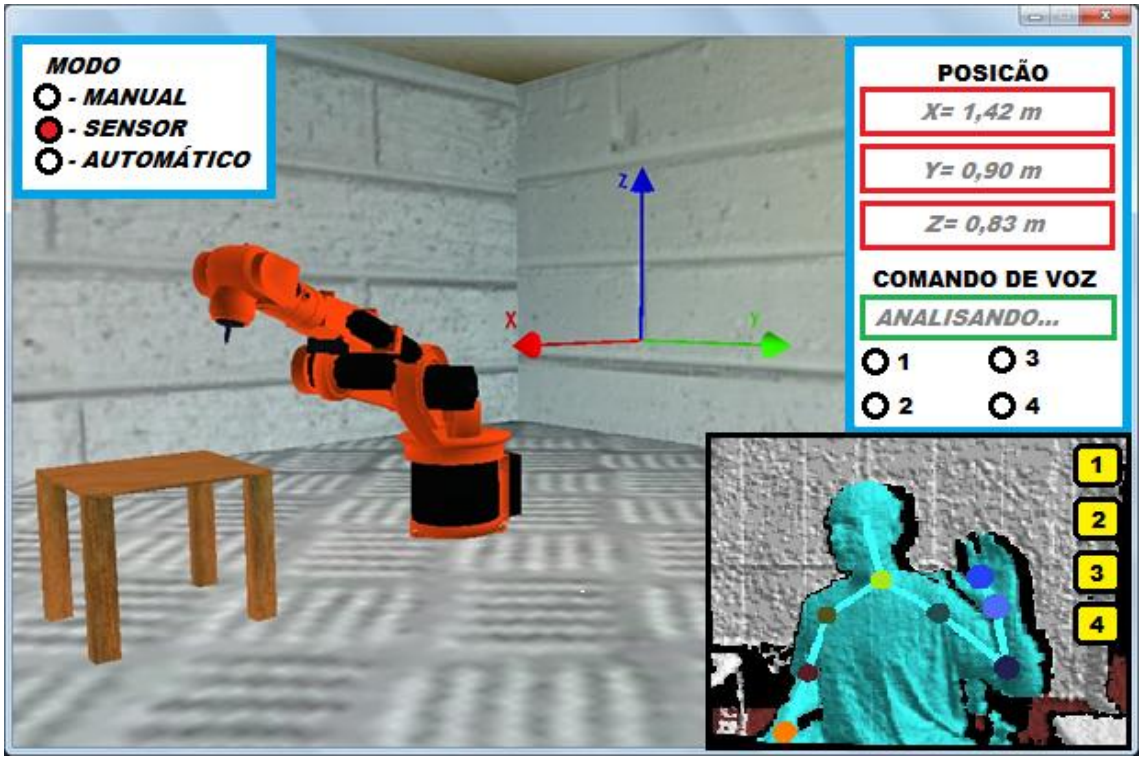

Figura 12. Protótipo da tela final.

Com os requisitos funcionais principais definidos, partiu-se para um maior detalhamento do comportamento de cada elemento do sistema. Para tal foi desenvolvido o diagrama de casos de uso da Figura 13.

O diagrama da Figura 13 exemplifica todas as ações (casos de uso) que podem ser tomadas pelo usuário (ator). Uma breve explicação de cada funcionalidade:

- Selecionar Modo (Figura 14) – essa seleção determina o modo de operação do programa. Pode ser dada por clique em botões ou por comandos de voz, em todos os modos. Pode-se selecionar entre modo Manual, Automático ou Sensor. No modo Manual os comandos são dados por meio do *mouse* e

teclado, no Automático por meio da execução do código e programação, e no Sensor por meio da combinação de movimentos e fala do usuário.

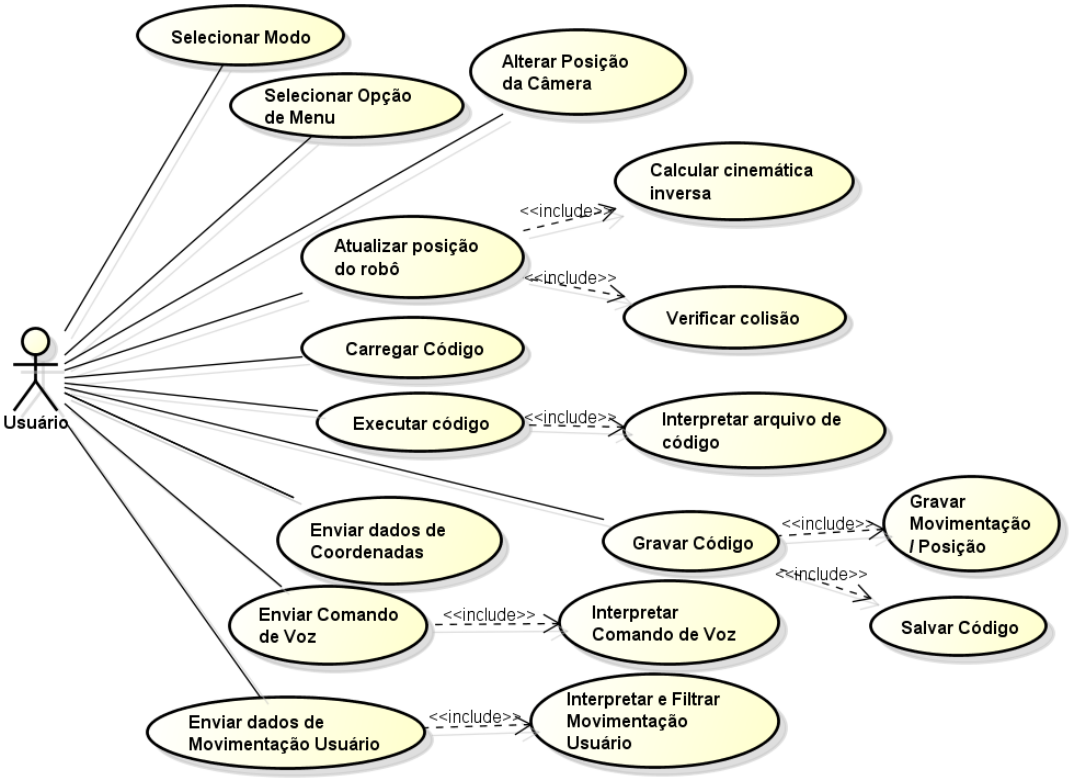

Figura 13. Diagrama de casos de uso do usuário.

- Selecionar opções de Menu (Figura 15) – em todos os modos. Diversos botões e campos com objetivos dos mais diversos (alguns equivalentes ao dos comandos de voz). Em todos os modos usam-se botões de seleção sendo que alguns comandos de voz são equivalentes a esses botões, em especial no modo sensor.
- Atualizar posição do robô (Figura 16) – em todos os modos de operação. Por meio da seleção de uma opção de menu, entra-se no modo de movimentação do robô (modo Manual e Sensor). No modo Automático não é necessária tal seleção. Depois de realizadas a cinemática inversa e a verificação de colisões, a posição do robô é atualizada (ou não, em caso de colisão). Essas duas operações determinam os casos de uso a seguir:
    - Calcula cinemática inversa – dadas as coordenadas da ponta da ferramenta desejadas, o programa calcula a rotação das juntas do braço robótico para aquela posição.

  - Verificar colisão – verifica a colisão do robô com objetos do ambiente e promove a alteração, retornando o robô para uma posição anterior. Além de verificar colisão também limita a movimentação ao volume de trabalho do robô.
- Carregar Código (Figura 17) – apenas no modo Automático. Usuário seleciona arquivo de texto com código de programação.
- Gravar Código (Figura 18) – as posições do robô são gravadas através de um comando de armazenar posição ou sequencialmente por meio de uma amostragem de gravação pré-determinada. O código é então armazenado em um arquivo de texto.
- Executar Código (Figura 19) – apenas no modo Automático. Código é executado, linha a linha, movimentando o robô (Atualizar posição do robô). Usuário interrompe, retoma ou cancela execução do código por meio de botões. Antes da execução, o código deve ser interpretado:
  - Interpretar arquivo de código – programa interpreta arquivo na linguagem de programação;
- Enviar Dados de Coordenadas (Figura 20) – apenas no modo Manual. Clica-se em botões que incrementam ou decrementam um campo (que também pode ser digitado diretamente). Tais valores são utilizados para movimentar a câmera ou o robô.
- Enviar Comandos de voz (Figura 21) – apenas no modo Sensor. Se alguma palavra-chave previamente armazenada é pronunciada, é tomada a ação respectiva ao comando. Essas ações podem ser, por exemplo, movimentar a câmera ou o robô (Seleção de Opções de Menu). Para determinar a pronúncia de uma palavra-chave, a fala deve ser antes interpretada:
  - Interpretar Comandos de voz – interpreta os dados recebidos e determina a palavra pronunciada com um grau de confiabilidade.
- Enviar dados de Movimentação Usuário (Figura 22) – apenas no modo Sensor. O programa recebe, a cada intervalo de tempo, dados da posição do corpo do usuário pelo Kinect. O movimento dos braços e mãos do usuário é capturado pela câmera, interpretado e enviado para movimentar o robô ou a câmera. Quanto à interpretação do movimento:

- Interpretar e Filtrar Movimentação Usuário – interpreta os dados recebidos, que passam por um tratamento. Ruídos e movimentos de pequena amplitude devem ser eliminados. Movimentos muito rápidos e de grande amplitude devem ser mantidos dentro dos limites permissíveis antes da transformação em coordenadas do robô/câmera;

- Alterar Posição da Câmera (Figura 23) – o usuário pode modificar a orientação e a posição da câmera. Essa funcionalidade está presente em todos os modos de operação. Nos modos Manual e Automático é executada por meio de comandos do teclado. No modo Sensor, é executada por meio da combinação de movimentos e fala.

São mostrados na sequência os diagramas de atividades de cada caso de uso base do programa.

Na Figura 14, a atividade “Selecionar Modo”, que exibe a tela referente ao modo selecionado.

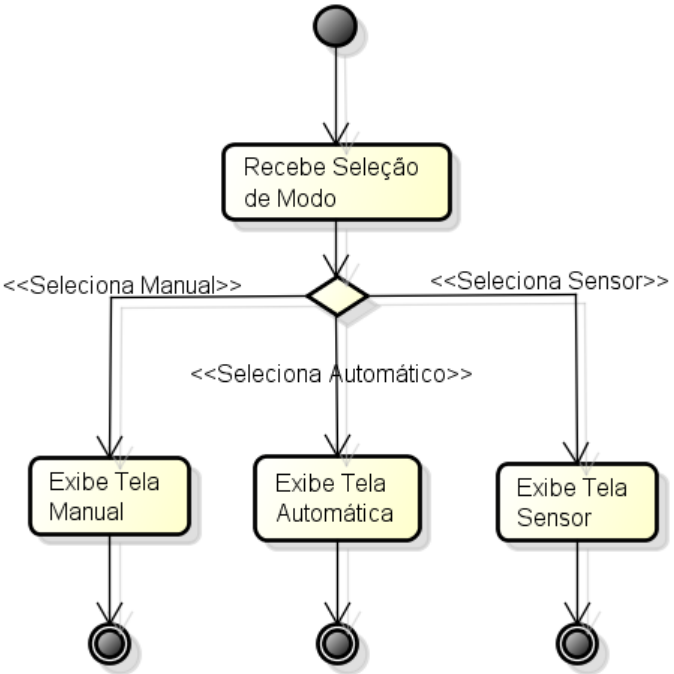

Figura 14. Diagrama de Atividades – Selecionar Modo.

Na Figura 15, a atividade “Selecionar Opção de Menu”, que altera a função referente à opção e menu selecionada.

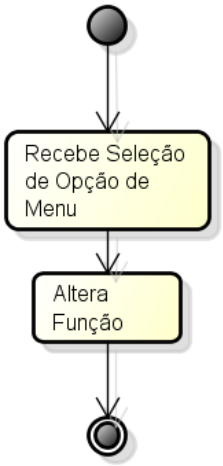

Figura 15. Diagrama de Atividades – Selecionar Opção de Menu.

Na Figura 16, a atividade “Atualizar posição do robô”, que recebe a nova posição, calcula a cinemática inversa e atualiza ou mantém a posição, com aviso sonoro, de acordo com o resultado da verificação de colisão.

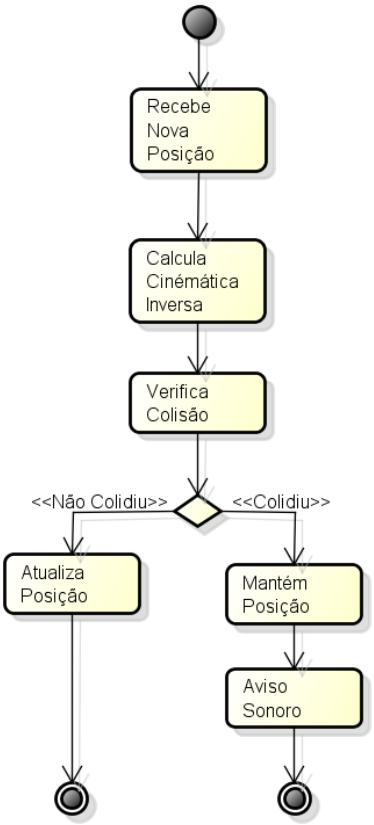

Figura 16. Diagrama de Atividades – Atualizar posição do robô.

Na Figura 17, a atividade “Carregar Código”, que recebe o arquivo selecionado e primeiramente verifica algum erro de leitura, exibindo na tela caso haja algum. Sem erros de leitura, é verificada a sintaxe, mostrando na tela se houver algum erro ou habilitando a execução para uma leitura bem sucedida.

Na Figura 18, mostra-se a atividade “Gravar Código”, que grava a posição atual do robô cada vez que o comando “Gravar Posição” é dado ou o timer de gravação de posição estoura no modo “Gravar Movimento”. Ao dar o comando “Salvar Código” o código é gravado em um arquivo de texto.

Na Figura 19, mostra-se a atividade “Executar Código”, que executa cada linha do código enquanto estiver no estado *PLAY*. Caso seja selecionado *PAUSE*, a execução fica pausada, e caso seja selecionado *STOP,* detectada uma colisão ou finalizado o código, a execução para.

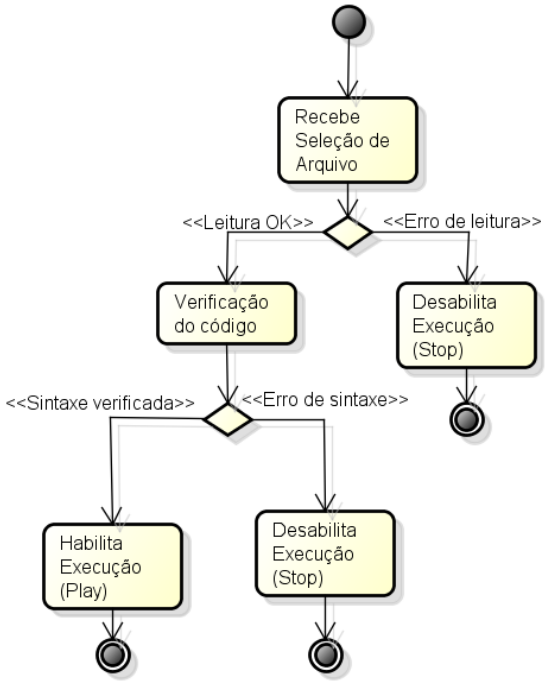

Figura 17. Diagrama de Atividades – Carregar Código.

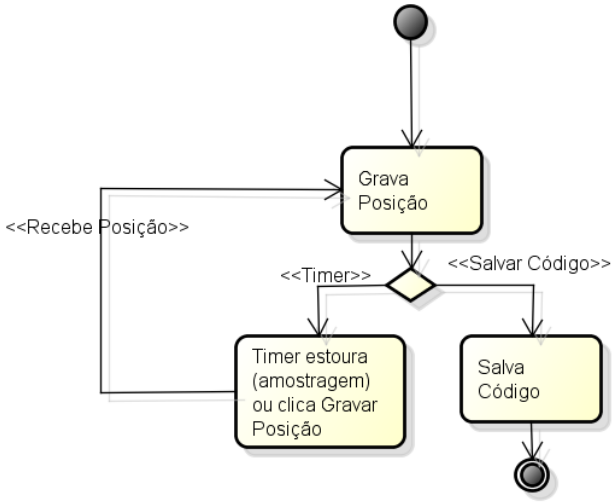

Figura 18. Diagrama de Atividades – Gravar Código

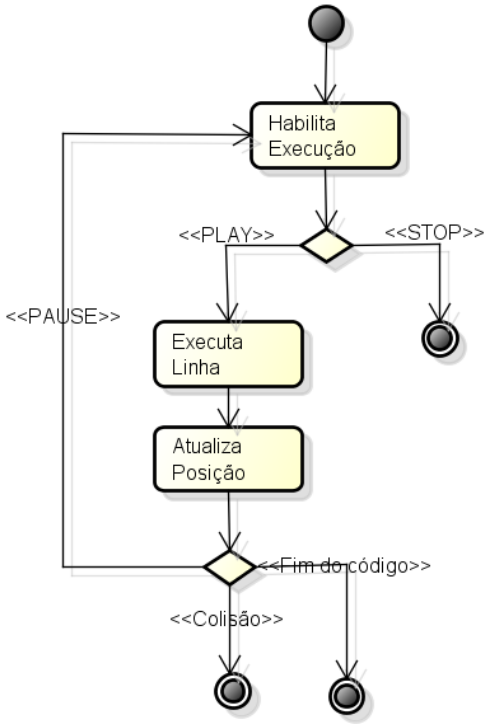

Figura 19. Diagrama de Atividades – Executar Código.

Na Figura 20 a atividade “Enviar dados de Coordenadas” recebe os dados de coordenadas e retorna os dados de posição.

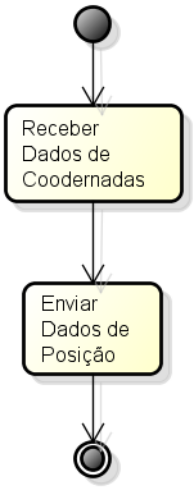

Figura 20. Diagrama de Atividades – Enviar dados de Coordenadas.

Na Figura 21, a atividade “Enviar Comando de Voz” recebe o comando de voz gerado pelo usuário, que posteriormente é interpretado e retorna a seleção da opção de menu referente ao comando, se for válido.

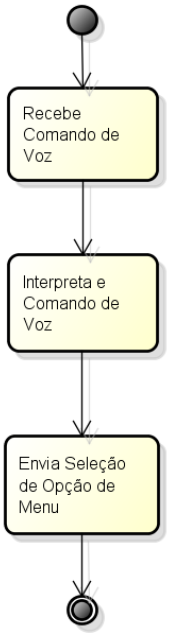

Figura 21. Diagrama de Atividades – Enviar Comando de Voz.

Na Figura 22, a atividade “Enviar dados de Movimentação do Usuário” recebe a movimentação do usuário, que é interpretada e retorna a nova posição.

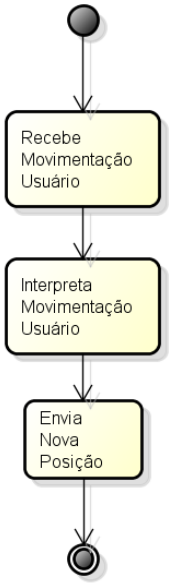

Figura 22. Diagrama de Atividades – Enviar dados de Movimentação Usuário.

Na Figura 23, a atividade “Alterar Posição da Câmera” que recebe a posição do corpo ou do braço, e altera a posição ou orientação da câmera.

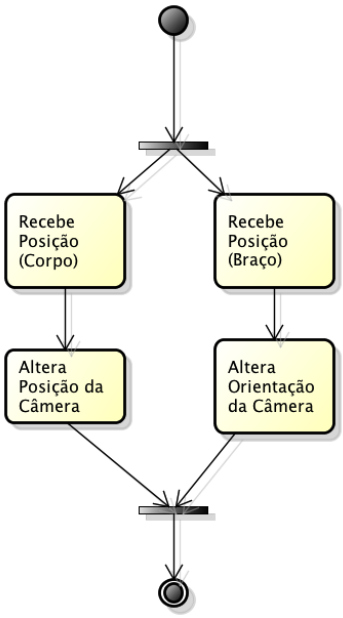

Figura 23. Diagrama de Atividades – Alterar Posição da Câmera.

Pela análise do problema, foi possível desenvolver o diagrama de classes simplificado indicado na Figura 24, que indica as novas classes, propriedades e funções que estão presentes no *software* desenvolvido.

Para entender e ilustrar o funcionamento do programa é apresentado adiante os diagramas de sequência para algumas das funcionalidades projetada, indicando a lógica do seu sistema e a maneira como foi abordada.

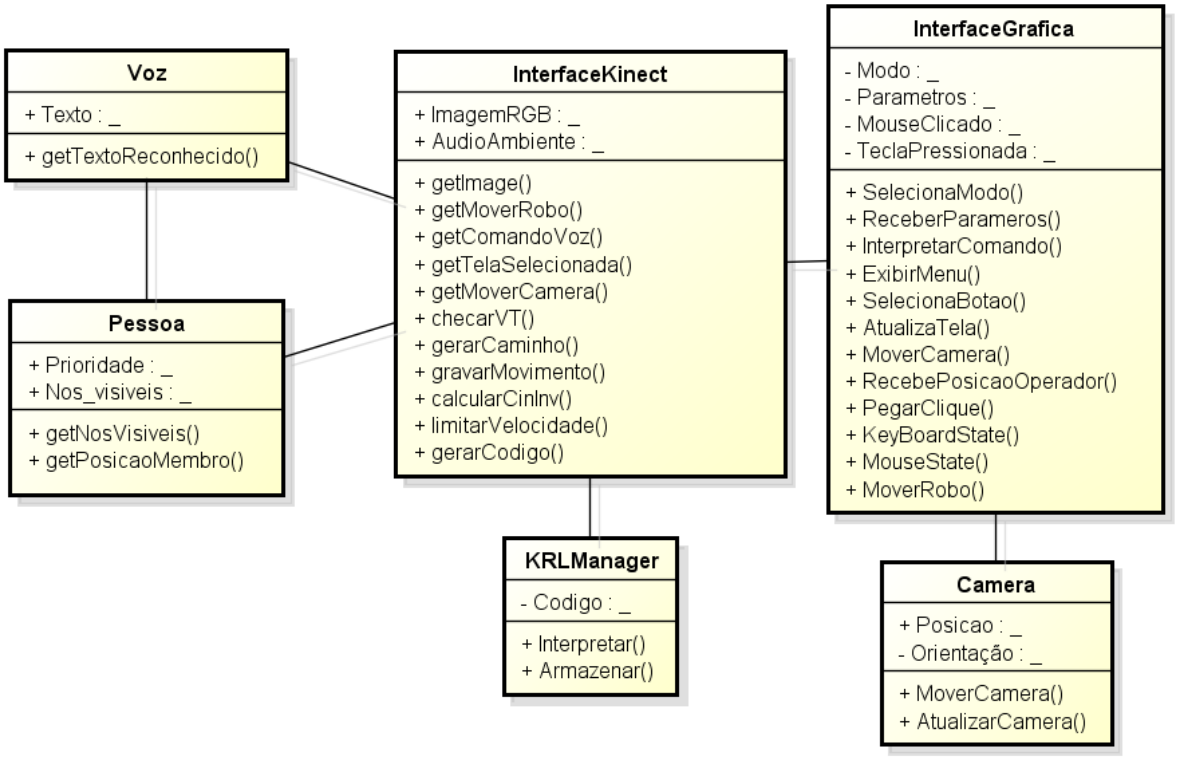

Figura 24. Diagrama de classes do projeto.

Na Figura 25, para a atividade "Alterar Posição da Câmera" no modo sensor a classe "InterfaceKinect" recebe o comando gestual do usuário pela função "getMoverCamera", recebendo a posição do membro que será referência para movimentar a câmera pelo mesmo processo descrito anteriormente, usando a classe "Pessoa". Com as coordenadas do membro, a classe "InterfaceKinect" chama a classe "InterfaceGrafica" pela função "MoverCamera", que será responsável por acionar a classe "Cameras" com a função "MoverCamera" para movimentar a câmera selecionada. Assim, a função "AtualizarCamera" é chamada para atualizar os valores e em seguida "AtualizarTela", da classe "InterfaceGrafica", para atualizar a tela.

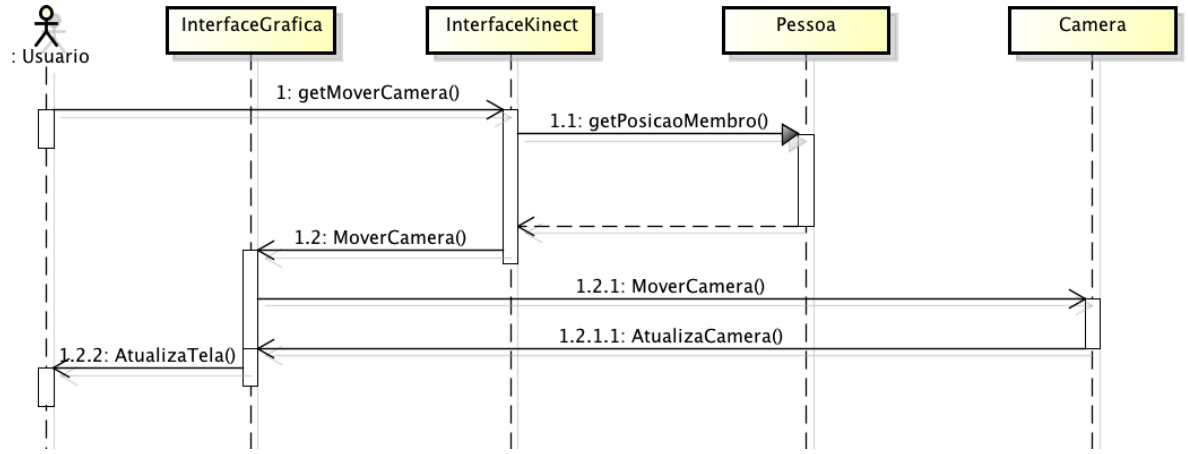

Figura 25. Diagrama de Sequência - Alterar Posição da Câmera.

Na Figura 26, para a atividade "Enviar Comando de Voz" a classe "InterfaceKinect" é responsável por reconhecer o que está sendo dito pelo usuário, com a função "getComandoVoz". Em seguida a classe "Voz" é chamada pela função "getFala", responsável por tratar o que foi dito, retornando um comando. Por fim, a classe "InterfaceKinect" aciona a classe "InterfaceGrafica" com a função "interpretarComando", que é responsável por acionar o que for preciso se o que foi dito for um comando válido.

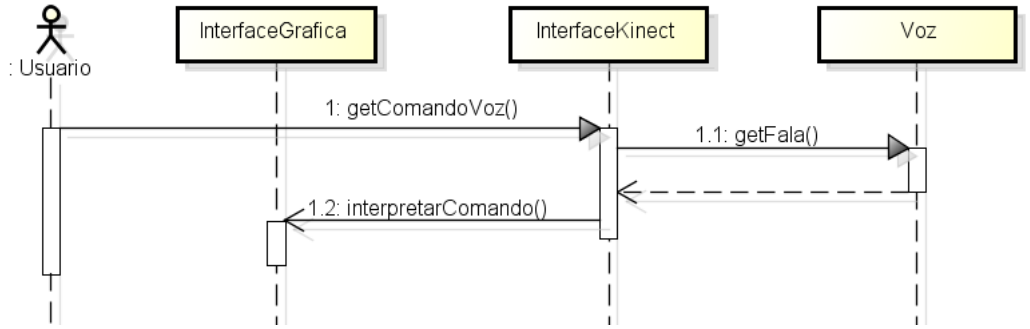

Figura 26. Diagrama de Sequência - Enviar Comando de Voz.

Na Figura 27, para a atividade "Atualizar Posição do Robô" no modo sensor a classe "InterfaceKinect" capta o movimento do usuário pela função "getMovimentoOperador". Em seguida o mesmo processo descrito anteriormente na atualização de posição da câmera é realizado, para obter as coordenadas dos nós do esqueleto modelado do operador, pelas classes "Pessoa". Com estes dados a classe "InterfaceKinect" checa se a posição desejada pertence ao volume de trabalho e se não há colisões pela função “checarVT”. O ponto então onde o robô

deve ser movimentado é dado pela função “gerarCaminho”, que leva em conta a velocidade e alguns tratamentos. A função “MoverRobo” da classe “Interface Gráfica” é chamada e em seguida "AtualizarTela”, para atualizar a tela.

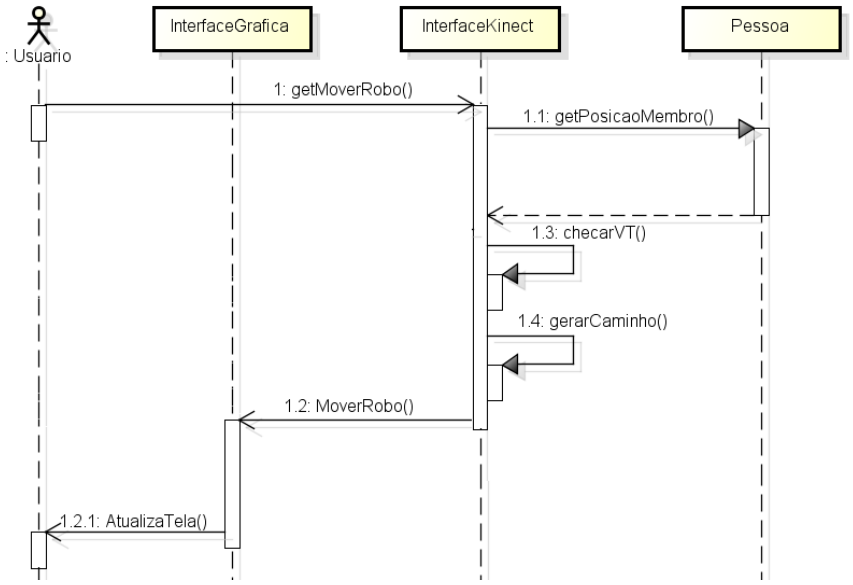

Figura 27. Diagrama de Sequência – Atualizar Posição do Robô.

A Figura 28 mostra a sequência de funcionamento da atividade de "Gravar Movimento" do sistema, que é acionada pela chamada da função "GravarMovimento()" da classe "InterfaceKinect", que pega a posição do membro por “getPosicaoMembro” e além de enviá-la para a interface gráfica, armazena-a através do método “Armazenar” da classe “KRLManager”.

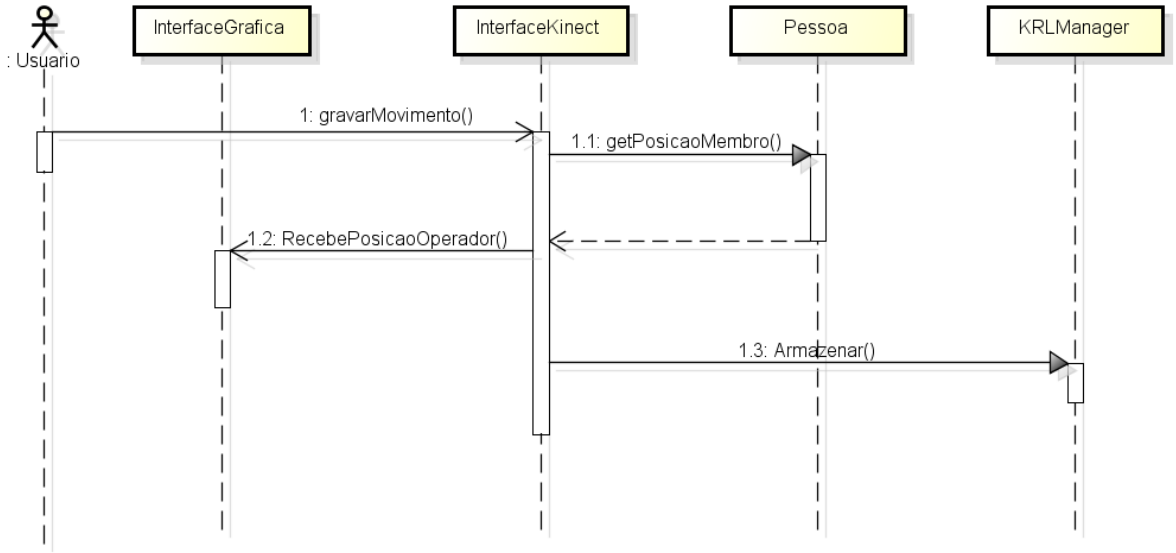

Figura 28. Diagrama de Sequência – Gravar Movimento

4. IMPLEMENTAÇÃO

4.1. DETECÇÃO DE MOVIMENTO

Optou-se por desenvolver aplicativos separados para as principais funcionalidades propostas para posterior integração com o projeto final.

Inicialmente foi desenvolvido o aplicativo para detecção de movimento. Para tal utilizou-se o Kinect SDK 1.5. A linguagem de programação é o C# e foi utilizado o ambiente de desenvolvimento Microsoft Visual Studio 2010.

Diversas particularidades podem ser mencionadas. Diversas funções são necessárias para verificar o estado dos possíveis sensores conectados e inicializar os módulos a serem utilizados. Contudo, apenas será detalhado o funcionamento geral do programa. Para um estudo mais detalhado, refira-se ao código do programa.

Para a detecção de movimento deve-se habilitar a classe *SkeletonStream.* Com ela realiza-se o rastreamento das juntas, utilizando uma função que recebe os dados do esqueleto sequencialmente pelo Kinect. Esses dados são compostos de um conjunto de matrizes que contém os dados das posições das juntas de todos os esqueletos detectados. Para este projeto, são necessários apenas os dados de um usuário. Desse modo define-se um usuário primário, cuja seleção é automática.

Logo de início optou-se pela utilização dos dados da junta da mão direita do usuário para o desenvolvimento do aplicativo (movimentação do robô) entre o conjunto de dados enviados pelo sensor.

Para determinar a precisão na detecção do movimento optou-se por desenhar em tempo real um círculo na posição onde se localiza a junta da mão direita e escrever na tela a profundidade detectada.

Dos dados enviados pelo sensor, a profundidade fornecida da junta observada é dada em metros. As coordenadas x e y, no entanto, são dadas na forma de um intervalo que varia entre -1 e 1 dependendo do lado.

Desse modo, inicialmente utilizou-se uma regra de três simples que convertia as coordenadas x e y (entre -1 e 1) no pixel da imagem onde se estimava o centro da junta. Note que a resolução escolhida na imagem foi a de 640x480 pixels e o resultado dessa conversão deveria resultar na coordenada de um pixel.

Contudo, o resultado de tal método não foi tão preciso quanto se esperava (Figura 29). A posição real da junta em alguns casos diferia grandemente do resultado obtido.

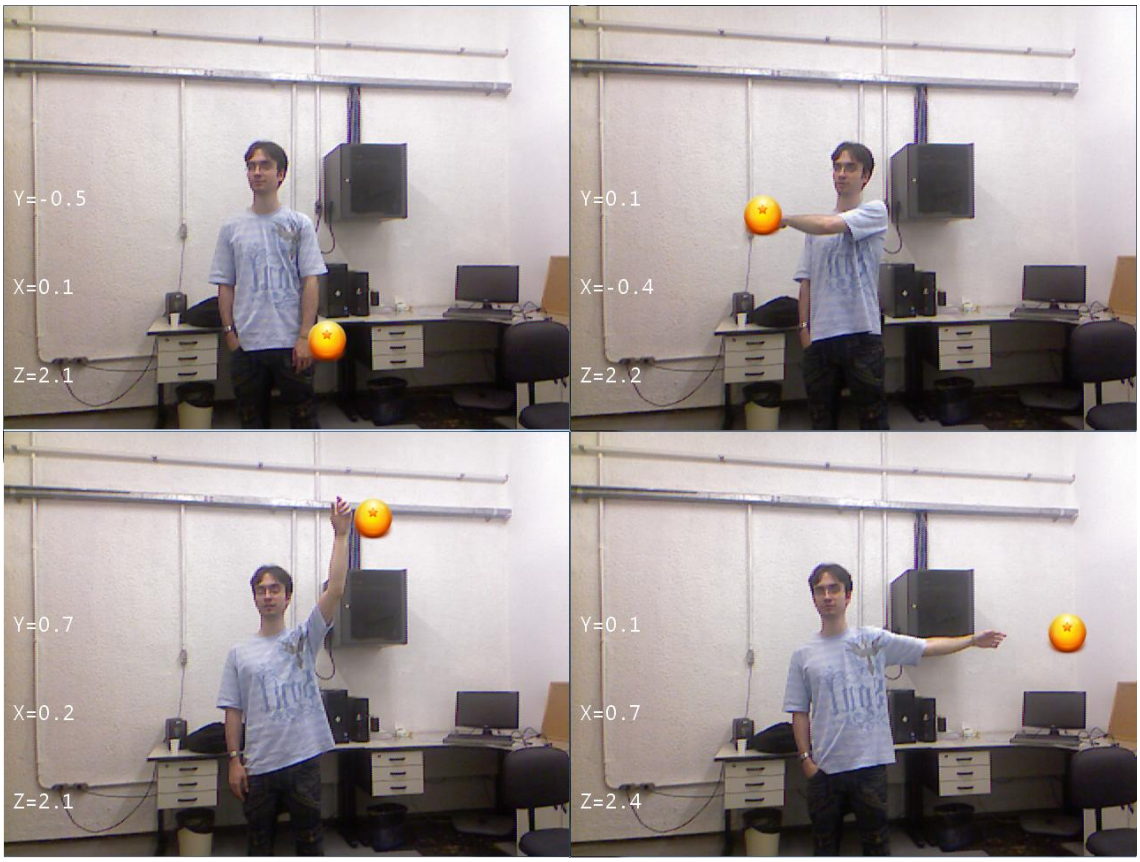

Figura 29. Resultado inicial obtido na detecção de movimento da junta da mão direita.

Um estudo mais minucioso mostrou que os valores máximos e mínimos de x e y dependem da distância do usuário à câmera. Eles podem variar aproximadamente entre 0,8 e -0,8, próximo da câmera (1m), até 1,2 e -1,2 longe da mesma (4m). Um esquema explicativo pode ser visto na Figura 30.

A origem do sistema de coordenadas é aproximadamente o centro da imagem obtida. O valor de x aumenta para a direita do usuário (esquerda do Kinect) e y aumenta para cima. Os dois planos mostrados, de forma exagerada, exemplificam a variação dos valores máximos e mínimos em relação distância da câmera. O plano é aquele considerado pelo Kinect como referência, com o qual ele compara o padrão infravermelho obtido pela câmera com uma referência armazenada, e cujas diferenças determinam a profundidade das juntas.

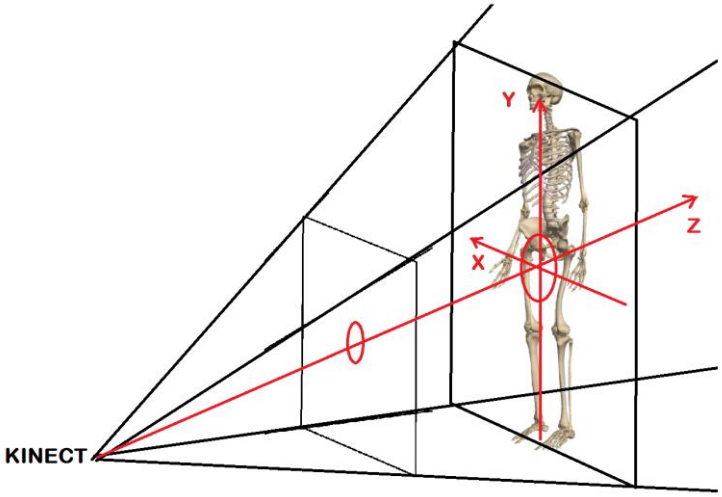

Figura 30. Esquema do sistema de coordenadas do Kinect.

Sendo assim, utilizando esses conhecimentos na conversão dos dados das juntas para as coordenadas dos pixels da imagem, obtemos o resultado da Figura 31. Um ponto a ser destacado é a necessidade de filtragem do sinal proveniente do Kinect. É comum observar oscilações das posições fornecidas pelo Kinect e, para evitar tais problemas, foi implementada uma filtragem por histerese dos dados coletados.

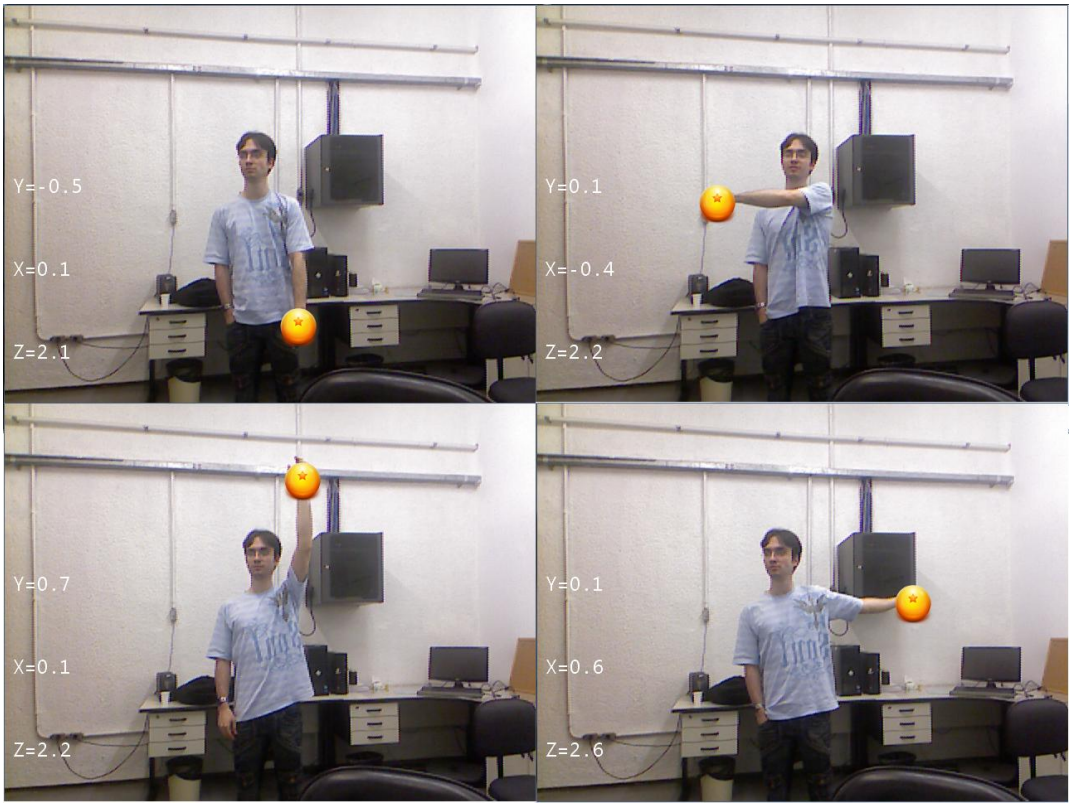

Figura 31. Resultado Final obtido na detecção da junta da mão direita.

4.2. DETECÇÃO DE SOM

Para a Detecção da Fala utilizou-se o *Microsoft Kinect Speech Recognition Language Pack*, presente no Kinect SDK 1.5. Como nesse pacote não está disponível no idioma Português, optou-se pelo Inglês.

Para a detecção de fala não é necessário o uso do Kinect, podendo ser utilizado um microfone convencional. O tratamento e reconhecimento do áudio ocorrem todo no computador. Isso é importante para os comandos para mudar, por exemplo, os modos de operação, que podem ser utilizados em qualquer modo de operação (Manual, Automático ou Sensor) que podem ser alterados por meio da fala. O Kinect pode não estar disponível em uma dada utilização, mas o usuário pode ainda optar por utilizar esses comandos de voz.

Diversas funções também foram aqui implementadas, para detectar erros e inicializar o *SpeechRecognizerEngine.* Após isso foram definidas as palavras-chave a serem mapeadas. Nesse ponto, ainda não é o objetivo mapear todas as frases necessárias para o projeto integrado.

As palavras definidas no dicionário de palavras-chave foram organizadas em duas categorias, Ações e Objetos, descritos na Tabela 1.

Tabela 1. Palavras-chave definidas

Ações	Objetos
"Move"	"Robot"
"Pause"	"Camera"
"Stop"	"Sensor"
"Operation"	"Automatic"
	"Manual"

Feito isso, partiu-se para o código que deve construir as frases esperadas. Foram criados dois objetos *Choices,* uma “acao” (para a Ação) e outro “objeto” (para o Objeto). Com cada objeto desses, foram determinadas todas as possíveis palavras em uma dada posição da frase.

Após isso, construiu-se a frase baseada nessas *Choices* e em valores estáticos. Utiliza-se para tal o *GrammarBuilder*, criando apenas uma possível configuração de frase. Começa-se a frase com a palavra estática “Kinect” para

determinar que será dado um comando “Kinect”+acao+objeto. Desse modo, é possível dar comandos como “Kinect Move Camera” ou “Kinect Operation Sensor”. Observe que há frases que podem ser formadas e que não fazem sentido no programa, como “Kinect Stop Manual” entre outras, que precisam ser ignoradas.

Implementou-se a detecção e análise constante do áudio de entrada, em todos os modos de operação. Após o processamento do áudio, o programa pode seguir três caminhos distintos. No primeiro, a frase é rejeitada por não ser compatível com as frases esperadas. No segundo, ela é reconhecida mas é compatível com mais de uma frase (resultado ambíguo). No terceiro, dentre todas as possibilidades, uma frase é reconhecida. A função implementada reconhece, dentre todas as hipóteses, aquela com a maior confiabilidade. Essa confiabilidade é comparada com uma confiabilidade mínima pré-estabelecida e caso a mesma seja maior do que a mínima, a frase é considerada como um comando válido.

No momento da integração, essa frase reconhecida será responsável por ações no programa. Nesse primeiro momento foi feito que a frase reconhecida seja impressa na tela. Na saída do programa é impressa a confiabilidade da frase reconhecida. O resultado pode ser visto na Figura 32.

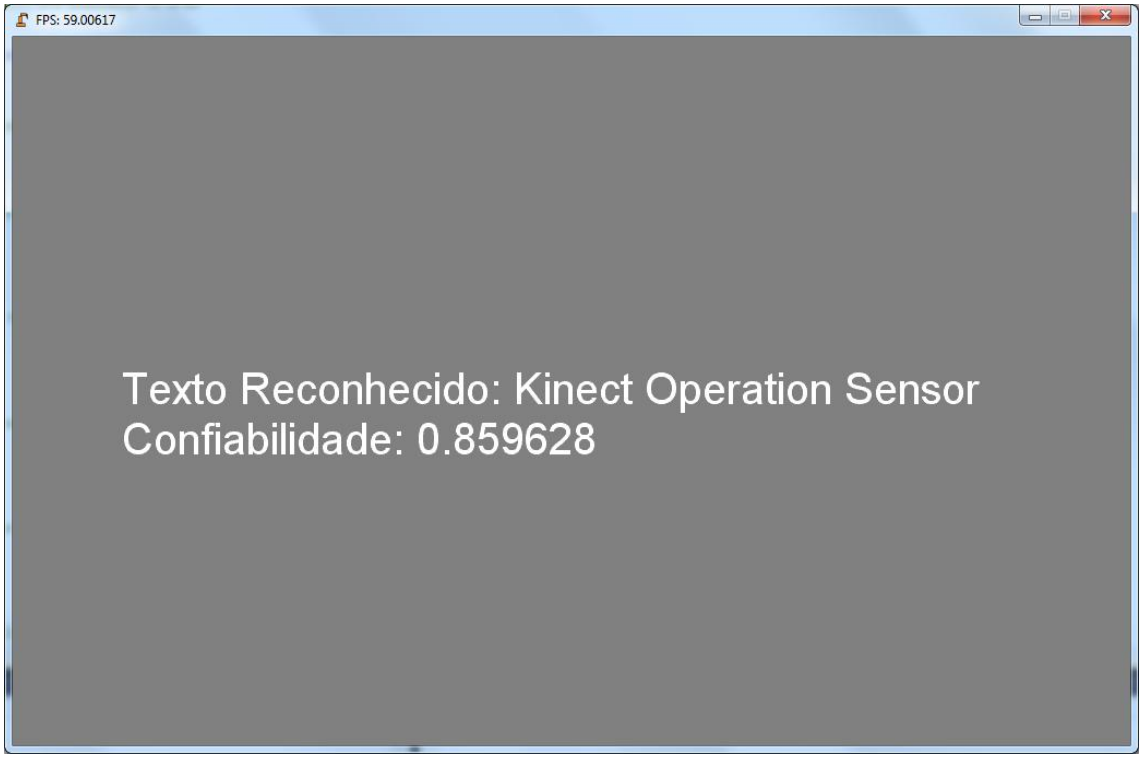

Figura 32. Valores impressos na tela do programa na detecção da fala.

4.3. INTEGRAÇÃO

4.3.1. MOVIMENTAÇÃO DO ROBÔ

O primeiro passo na integração dos programas desenvolvidos com o simulador do robô Kuka foi a da detecção de movimento. Analisou-se o programa para determinar as bibliotecas a serem importadas e novas funções e variáveis a serem criadas e modificadas.

Um novo modo no programa teve que ser criado, o modo Sensor. Novos menus foram criados para a seleção desse modo e para a impressão na tela de uma miniatura da imagem captada pela câmera.

Implementou-se a leitura constante dos dados e atualização dos valores da ponta da ferramenta do robô, seguido pela chamada da cinemática inversa com o objetivo de obter os ângulos dos três primeiros graus de liberdade (A1, A2 e A3).

4.3.1.1. CINEMÁTICA DE POSIÇÃO

O primeiro problema encontrado foi que a cinemática inversa implementada também não era própria para a solução do problema, levando ao travamento inesperado do braço em posições onde a mesma não devia ocorrer. Não havia cálculo do volume de trabalho do robô, o quê significa que ao dar coordenadas fora desse volume para o robô, o mesmo travava e não respondia aos comandos seguintes (Figura 33). Isso constitui uma singularidade de fronteira (SICILIANO *et al.*, 2010), que não era evitada. Como não era feita verificação do volume de trabalho, os ângulos ainda eram calculados nessas condições e após isso havia o travamento total. Isso era um problema no modo manual e impossibilitava o uso do modo sensor, da detecção de movimento.

Inicialmente foi trabalhada a cinemática inversa. Apesar de se tratar de um problema de difícil solução, por se tratar de um manipulador de seis graus de liberdade com as últimas três juntas se intersectando em um único ponto, é possível desacoplar a cinemática inversa em dois problemas mais simples, a cinemática inversa de posição e a cinemática inversa de orientação.

A cinemática inversa de posição deduzida no trabalho anterior (MIYASHIRO; SUGAWARA, 2011) estava correta. Apenas pecava em detalhes da implementação

computacional e tratamento de singularidades. Baseados em (SICILIANO *et al.*, 2010), (MIYASHIRO; SUGAWARA, 2011) chegaram aos seguintes resultados.

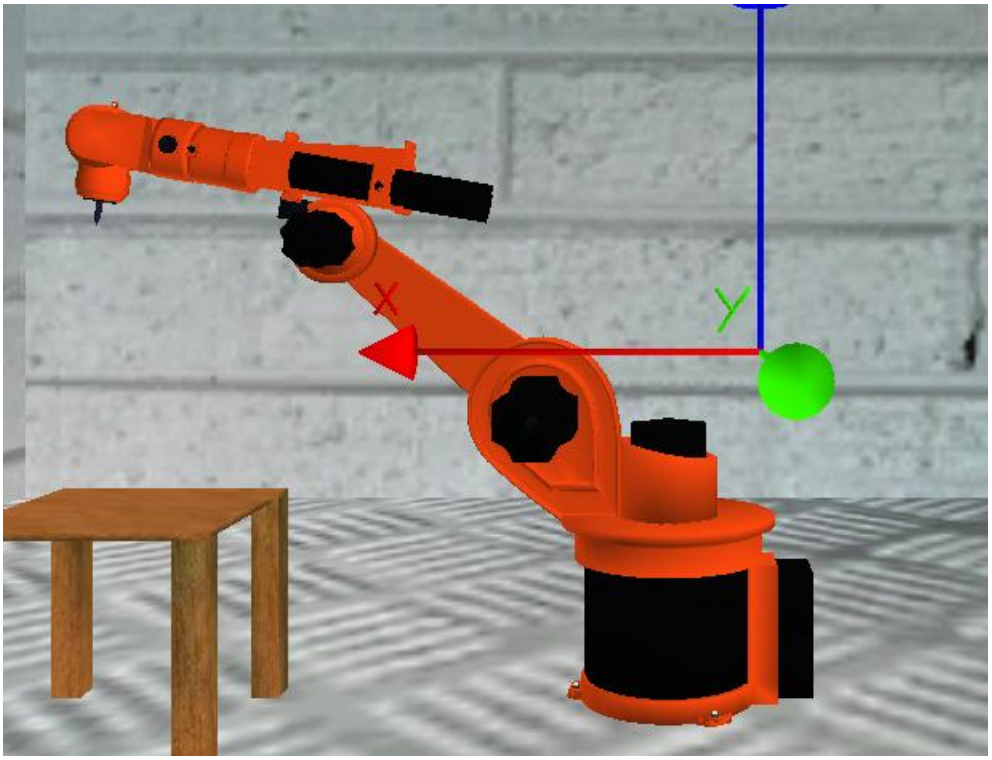

Figura 33. Posição de travamento do robô, no limite do volume de trabalho.

Pela Figura 34, obtêm-se:

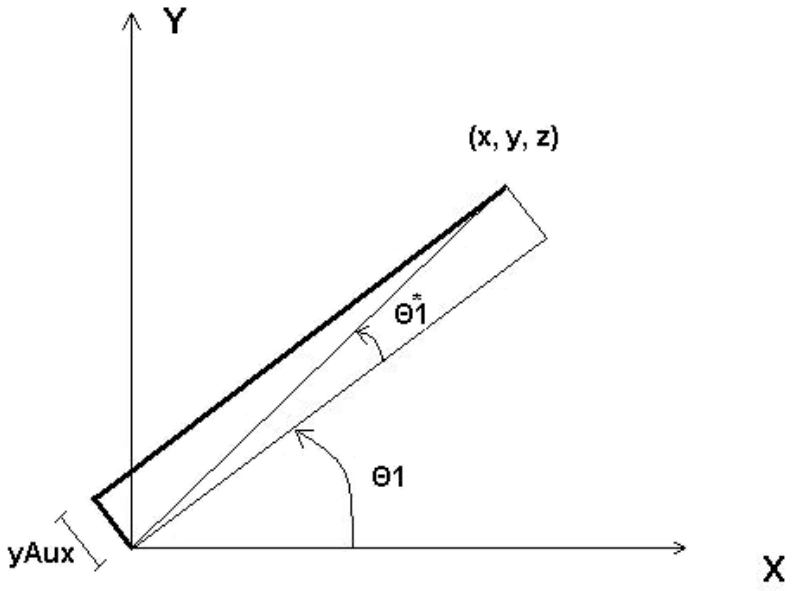

Figura 34. Vista superior do robô. (Miyashiro; Sugawara, 2011).

$$sen(\theta_1^*) = \frac{yAux}{\sqrt{x^2 + y^2}} \quad (1)$$

$$\theta_1 = atan2(y, x) - \theta_1^* \quad (2)$$

Onde yAux está relacionado com o deslocamento entre os eixos das juntas rotativas devido à estrutura do braço. A função

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

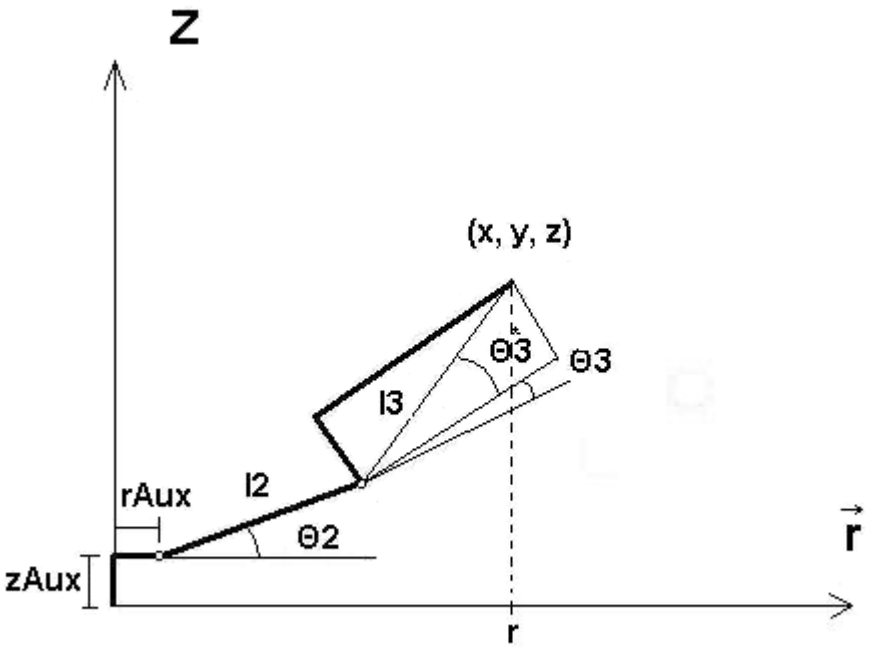

Figura 35. Vista lateral do robô com as variáveis para o cálculo deθ_3. (Miyashiro; Sugawara, 2011).

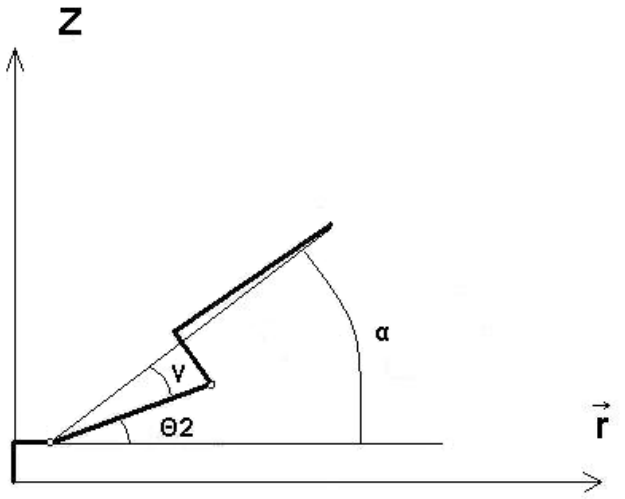

Figura 36. Vista lateral do robô com as variáveis para o cálculo deθ_2. (Miyashiro; Sugawara, 2011).

Onde zAux e rAux estão relacionados com o deslocamento entre os eixos das juntas rotativas devido à estrutura do braço. Agora para os ângulos das juntas A2 (θ_2) e A3 (θ_3), por meio das Figuras 35 e 36, obtêm-se:

$$(r - rAux)^2 + (z - zAux)^2 = {l_2}^2 + {l_3}^2 + 2l_2 l_3 \cos(\theta_3 + \theta_3^*) \quad (3)$$

Isolando-se θ_3 na equação (3), chega-se em:

$$\theta_3 = \arccos\left(\frac{(r - rAux)^2 + (z - zAux)^2 - {l_2}^2 - {l_3}^2}{2l_2 l_3}\right) - \theta_3^* \quad (4)$$

$$\propto = atan2(z - zAuz, r - rAux) \quad (5)$$

$$\gamma = atan2(l_3 sen(\theta_3 + \theta_3^*), l_2 + l_3 cos(\theta_3 + \theta_3^*)) \quad (6)$$

$$\theta_2 = \alpha - \gamma \quad (7)$$

A cinemática inversa foi então refeita, utilizando essas equações e levando em consideração o tratamento de singularidades e uma melhor implementação computacional de modo que a dedução anteriormente obtida fosse corrigida.

4.3.1.2. CÁLCULO DO VOLUME DE TRABALHO

Trata-se agora do problema do travamento do robô para pontos fora do volume de controle. Existe solução para o problema de cinemática inversa quando o ponto do centro do pulso, utilizado para o cálculo, respeita o volume dado pela Figura 37.

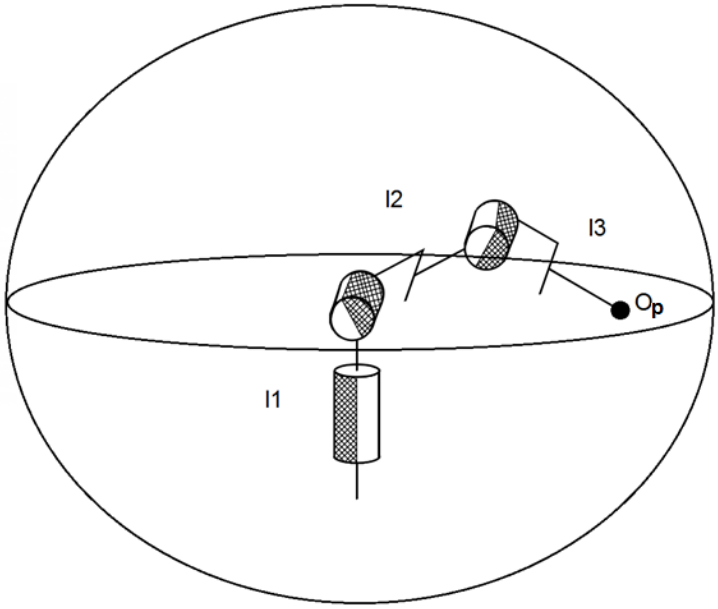

Figura 37. Volume de trabalho do Robô.

Para existência de solução:

$$\left\| O_p - \begin{bmatrix} X_{aux} \\ Y_{aux} \\ Z_{aux} \end{bmatrix} \right\| \leq l_2 + l_3 \tag{8}$$

Onde O_p é a posição do centro do pulso e $Z_{aux} = l_1$ e ${X_{aux}}^2 + {Y_{aux}}^2 = {R_{aux}}^2$.

Fazendo o quadrado da expressão anterior e isolando os termos obtêm-se a seguinte solução de existência:

$$\frac{(R - R_{aux})^2 + (Z - Z_{aux})^2 - {l_2}^2 - {l_3}^2}{2.l_2.l_3} \leq 1 \tag{9}$$

4.3.1.3. LIMITAÇÃO DE VELOCIDADE

Agora também surge o problema de limitar as juntas e a velocidade linear visto que as mesmas são limitadas no robô real, seja fisicamente ou por meio da velocidade imposta. O movimento do robô real está claramente limitado à potência máxima fornecida pelos motores ou à velocidade máxima desejada em uma determinada aplicação. Desse modo, foi necessário limitar o movimento do robô e sincronizá-lo (movimento linear), de modo que a direção de maior deslocamento tenha a maior velocidade e a trajetória entre os pontos desejados seja aproximadamente uma reta. Tal procedimento ainda não era realizado. Para a limitação da velocidade linear implementou-se uma representação vetorial do problema entre os pontos atual e desejado para o cálculo da cinemática inversa. Determinou-se também uma velocidade linear máxima. Ao receber um ponto, é calculado um vetor unitário na mesma direção e sentido que, limitado pela velocidade atual, determina qual é o ponto intermediário a ser utilizado para o cálculo da cinemática inversa e que respeita a velocidade definida. Após a movimentação, com esse novo ponto, calcula-se novamente a cinemática inversa, e assim consecutivamente, até o ponto desejado.

Observe que até o momento apenas foi limitada a velocidade de movimentação linear do programa. As juntas também estão limitadas a uma velocidade de rotação máxima, que deve ser respeitada. Do mesmo modo que foi obtida uma relação entre a velocidade linear e trajetória, também limitou-se as velocidades máximas das juntas, mantendo as devidas proporções entre elas de modo a manter a trajetória. Um exemplo onde é possível notar o efeito dessa limitação é ao mudar a posição de coordenada x positiva para uma posição de coordenada x negativa, onde apesar da velocidade linear ser pequena, a velocidade de rotação imposta à junta A1 é altíssima se não for limitada. Essa é uma singularidade interna, onde uma pequena velocidade operacional provoca uma grande velocidade das juntas (SICILIANO *et al.*, 2010).

4.3.1.4. PARTICULARIDADES DO MODO SENSOR

A estratégia de limitação de velocidade anterior foi particularmente útil para o caso do modo sensor. Observe a situação demonstrada na Figura 38, onde se mostra uma posição fora do volume de trabalho.

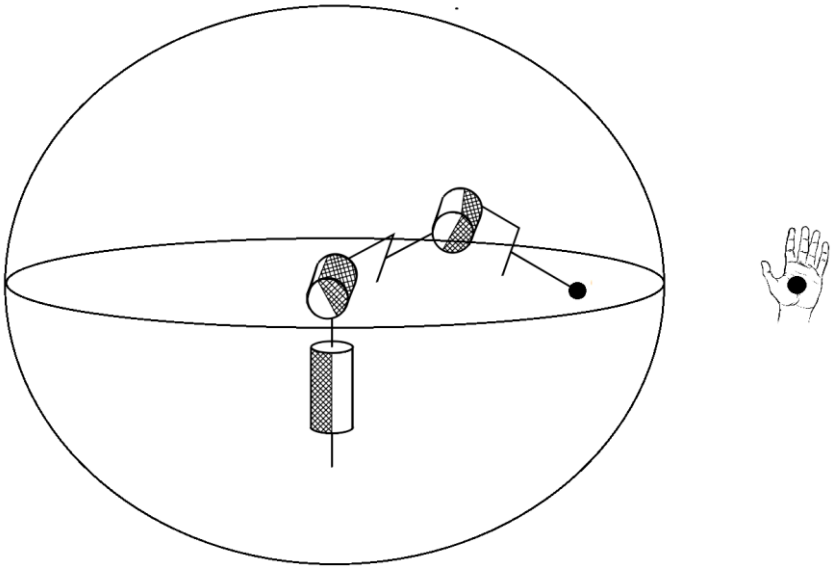

Figura 38. Posição desejada fora do Volume de Trabalho.

Ao posicionar a mão fora do volume de trabalho, ao checar a condição de existência, não havia solução e o robô não se movia. Isso era um grande problema, sobretudo no modo sensor onde os movimentos realizados pelo usuário são amplos e difíceis de controlar. Utilizando a estratégia de movimentação linear (Figura 39), entretanto, é possível contornar tal problema.

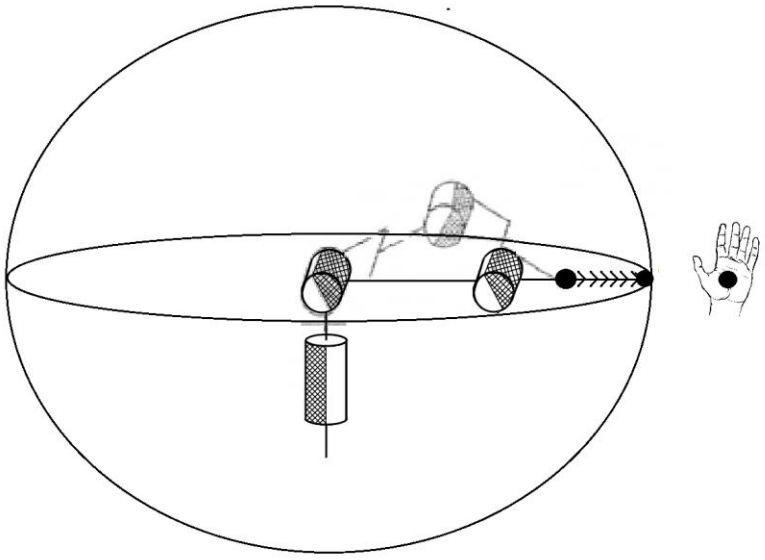

Figura 39. Posição desejada fora do Volume de Trabalho.

Como o cálculo da cinemática inversa é realizado em pequenos incrementos, devido à limitação de velocidade, ao posicionar a junta da mão fora do volume de controle, o robô se movimenta o mais próximo possível da mesma. Outro problema surge no modo sensor quando, com a posição desejada fora volume de trabalho, e com o robô já no limite do mesmo, movimenta-se a mão para outra posição, ainda fora do volume de trabalho. Nessa situação, como o caminho a ser percorrido pelo robô não possui solução, o mesmo fica travado. Esse travamento não é aceitável no modo sensor, onde a fluidez e resposta rápida são fundamentais para a experiência do usuário. Tal situação pode ser demonstrada pela Figura 40.

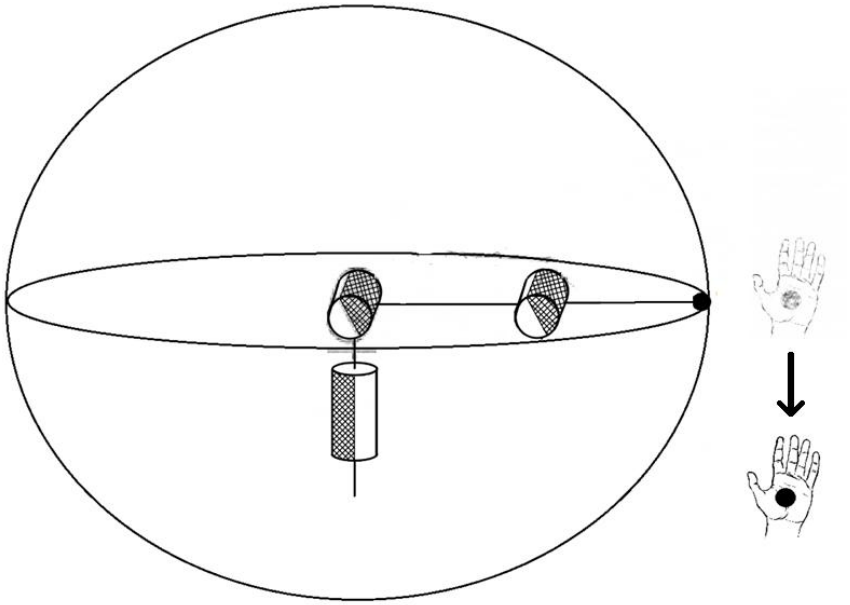

Figura 40. Movimentação partindo de Posição no limite do Volume de Trabalho.

Para solução desse problema implementou-se uma função que, a partir de um vetor localizado no centro do sistema de coordenadas, caso a posição desejada seja fora do volume de trabalho, obtêm-se o ponto nos limites do volume mais próximo do desejado, na intersecção dessa reta com a circunferência. O resultado disso pode ser visto na Figura 41.

4.3.2. CINEMÁTICA INVERSA DE ORIENTAÇÃO

Observe que até esse momento, apenas discutiu-se a obtenção dos valores dos três primeiros graus de liberdade, que posicionam o centro do punho do robô. Os três últimos, pertencentes ao próprio punho esférico, responsáveis pela destreza

e orientação do efetuador, foram ignorados. Esse é o problema da cinemática inversa de orientação.

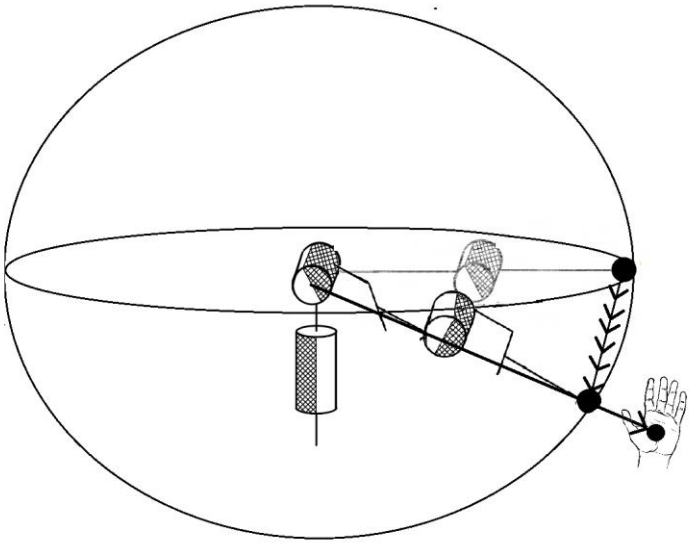

Figura 41. Movimentação linear partindo de Posição no limite do Volume de Trabalho com movimentação linear.

A cinemática inversa de posição, mostrada anteriormente, é feita utilizando como ponto o centro do punho esférico, de modo a desacoplar orientação e posição, como dito anteriormente.

$$O_p = O - d.R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{10}$$

Onde O_p é a posição do centro do pulso, O a posição desejada, R a orientação desejada na forma de Row-Pitch-Yaw (em relação ao sistema fixo) e d o comprimento do ligamento obtido pela notação de Denavit-Hartenberg (SICILIANO *et al.*, 2010).

Com os resultados obtidos pela cinemática de posição $\theta_1, \theta_2, \theta_3$, foi obtida a matriz de rotação R_3^0.

O problema de obter o ângulo das juntas 4, 5 e 6 ($\theta_4, \theta_5, \theta_6$) se resume então a achar os ângulos de Euler (sistema móvel) correspondentes a matriz de rotação:

$$R_6^3 = (R_3^0)^{-1}.R = (R_3^0)^T.R \tag{11}$$

Onde, pela composição de rotações pode-se deduzir (SICILIANO *et al.*, 2010):

$$R_3^0 = \begin{bmatrix} C_1C_{23} & -C_1S_{23} & S_1 \\ S_1C_{23} & -S_1S_{23} & -C_1 \\ S_{23} & C_{23} & 0 \end{bmatrix} \quad (12)$$

e

$$R_6^3 = \begin{bmatrix} C_4C_5C_6 - S_4S_6 & -C_4C_5S_6 - S_4C_6 & C_4S_5 \\ S_4C_5C_6 + C_4S_6 & -S_4C_5S_6 + C_4C_6 & S_4S_5 \\ -S_5C_6 & S_5S_6 & C_5 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (13)$$

Onde C denota a função Coseno e S a função Seno. Para maiores detalhes refira-se a (SICILIANO *et al.*, 2010).

Utilizando o valor obtido pela multiplicação das matrizes de (11), que são dadas, para obter R_6^3 e comparando com (13) obtêm-se:

$$\theta_4 = Atan2(r_{23}, r_{13})$$

$$\theta_5 = Atan2(\sqrt{{r_{13}}^2 + {r_{23}}^2}, r_{33})$$

$$\theta_6 = Atan2(r_{32}, -r_{31})$$

Para $\theta_5 \in (0, \pi)$ e

$$\theta_4 = Atan2(-r_{23}, -r_{13})$$

$$\theta_5 = Atan2(-\sqrt{{r_{13}}^2 + {r_{23}}^2}, r_{33})$$

$$\theta_6 = Atan2(-r_{32}, r_{31})$$

Para $\theta_5 \in (-\pi, 0)$

Assim implementou-se a cinemática inversa de orientação. Inicialmente buscou-se utilizar a junta do pulso do usuário para controlar os três primeiros graus de liberdade e a posição entre a mão e o pulso para controlar os outros. Devido a limitações do próprio punho humano e a imprecisões, dada a pequena distância entre essas juntas, tal aproximação do problema não obteve resultados satisfatórios. Desse modo, optou-se pelo uso da detecção de movimento do braço esquerdo o controle dos ângulos de Euler “Row, Pitch e Yaw”. O controle então se tornou confuso para o usuário, tendo que utilizar os dois braços para o controle, dificultando, possivelmente o aprendizado. Além disso, por vezes, o passar de um

braço por cima do outro afetava a detecção das juntas prejudicando o resultado obtido. Sendo assim, optou-se por utilizar uma orientação fixa, determinada pelo programa, que deveria ser ativada por comando de voz. Tal orientação seria, por exemplo, a de um plano onde se deseja realizar uma operação de pintura ou escrita, assim como manipular objetos sobre uma mesa.

4.3.3. DETECÇÃO DE COLISÃO

Verificou-se que a detecção de colisão utilizada em (MIYASHIRO; SUGAWARA, 2011) tinha falhas e não atendia as necessidades do projeto. Apenas detectava a colisão da ponta da ferramenta com a mesa (Figura 42). Além disso, apenas detectava colisão no movimento individual dos eixos. Isso significa que ao mover o robô na direção x, caso ocorra choque, o robô era movimentado na direção oposta. A movimentação em uma direção não paralela a nenhum eixo possuía falhas que gerava resultados indesejados. Os valores das posições ficavam incorretos, e com isso, a cinemática inversa resultava em erros e o controle do robô ficava prejudicado.

Para correção desse problema, na função responsável pelo movimento das juntas, caso detectada colisão, as juntas são movidas para o valor angular imediatamente antes da colisão, de modo que as colisões em qualquer direção são tratadas corretamente. Além disso, determinaram-se volumes de controle não apenas na mesa e na ferramenta, mas também no chão e em todas as juntas e ligamentos de modo a simular o mais próximo possível da colisão real do robô. Esses volumes de controle são dinâmicos e se movimentam junto do robô.

4.3.4. MOVIMENTAÇÃO DA CÂMERA

No programa anterior (MIYASHIRO; SUGAWARA, 2011) estava implementada a função “FirstPersonCamera”. Apesar de funcional para os propósitos anteriores, tal função não permitia o controle direto da orientação e posição da câmera de modo simples. Desse modo, decidiu-se por implementar uma nova função que determinasse e atualizasse a posição e orientação da câmera, através da qual é determinada a projeção da imagem na tela. Por meio dos movimentos do braço direito do usuário determina-se a orientação desejada da câmera (*Row* e *Pitch*) e através da distância do usuário ao dispositivo determina-se

a translação. Com esses valores, foram utilizados os conceitos de matrizes de rotação e translação para a obtenção da matriz de visão da câmera e posterior matriz de projeção perspectiva.

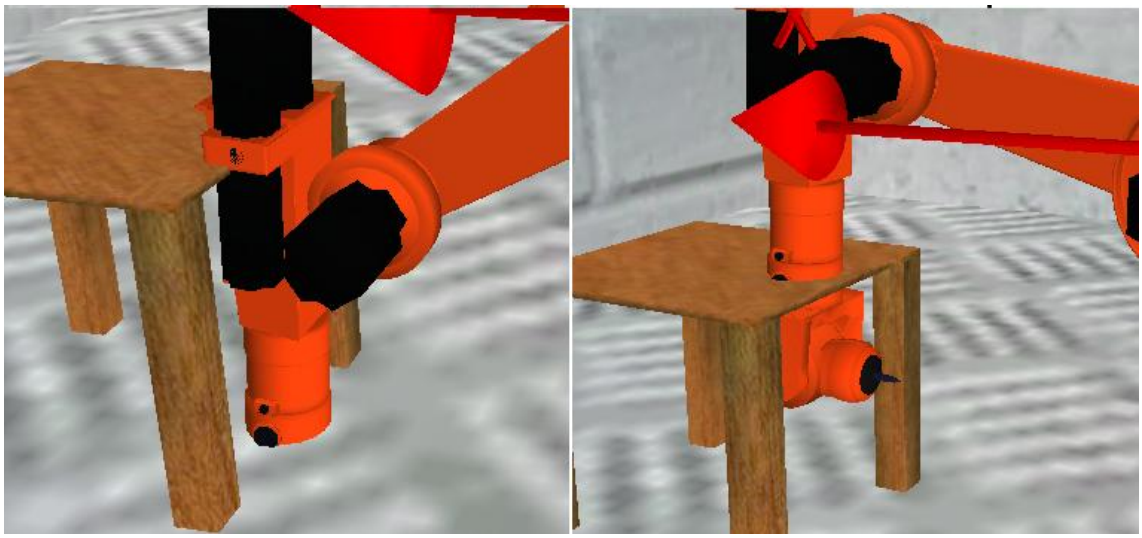

Figura 42. Problemas na detecção de colisão.

Observe que é necessário manter coerente a movimentação da câmera com a movimentação do braço, que está atrelada ao sistema de coordenadas do robô e sua relação (que foi feita fixa) com a posição do sistema de coordenadas do usuário. Sendo assim, as translações da câmera apenas podem ser realizas na direção y, perpendicular à tela. Para a translação optou-se por utilizar a distância entre o ombro direito do usuário e o sensor.

4.3.5. GRAVAÇÃO DE MOVIMENTO

4.3.5.1. MODO AUTOMÁTICO

Decidiu-se no decorrer do projeto por tentar armazenar os dados de movimentação do robô de modo a poder controlar o robô real por meio dos movimentos do corpo, aplicação futura deste projeto. Para tal decidiu-se utilizar a própria linguagem de programação KRL (KUKA ROBOTER, 2010), criada pelo fabricante Kuka, para armazenar esses dados que posteriormente podem ser lidos pelo robô.

Analisando então o modo automático previamente implementado, observou-se que o mesmo não funcionava como deveria. Ele apenas funcionava para a

movimentação de um eixo por vez, ou seja, se a posição desejada necessitasse de movimentação nos eixos x e y, ele se movimentava em um eixo por vez. Isso se utilizado o único comando aceito, o LIN, que deveria traçar uma trajetória reta entre os pontos. Além disso, não era possível se movimentar no eixo z e apenas no sentido positivo dos eixos x e y. Não era possível pausar (*pause*) ou parar (*stop*) o código. Uma vez executado, apenas a reinicialização total do programa permitia uma nova execução. Além disso, o arquivo lido era fixo, algo que o grupo achou necessário mudar.

Foram utilizadas os recursos das classes KRLParser e KRLLexer, que haviam sido geradas pelo *software* ANTLRWorks, para auxiliar na interpretação.

O grupo então optou por refazer toda classe KRLManager e a parte do programa principal responsável pela interpretação dos comandos de modo a permitir movimentação em todos os eixos, e com controle de velocidade (através de leitura do código).

Quanto aos modos de movimentação, implementaram-se três comandos aceitos: LIN, PTP e LIN CONT. LIN é o comando de movimentação Linear (Figura 43). Nele o robô se movimenta de modo contínuo através de pontos com pequenos incrementos entre si que determinam uma trajetória reta entre os pontos iniciais e finais. PTP (*Point to Point*) é o comando no qual as posições intermediárias não são inseridas na programação da trajetória, que de fato não existe (Figura 44). Todos os motores são acionados ao máximo (limitado pela velocidade programada) de modo que a movimentação entre os dois pontos seja a mais rápida possível, em cada eixo. LIN CONT é a movimentação contínua na qual o ponto de destino não é exato, mas sim aproximado (Figura 45). Desse modo o robô não precisa ser desacelerado até parar no ponto de destino, ocorrendo menos desgaste no robô real e um menor tempo de ciclo (CABRAL, 2012).

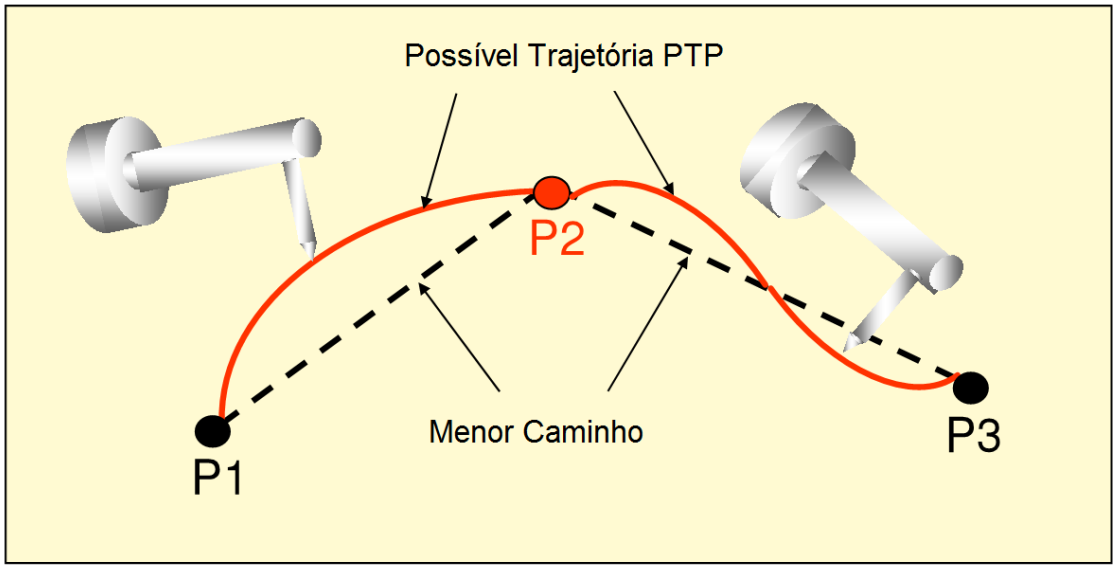

Figura 43. Movimentação Point to Point (CABRAL, 2012).

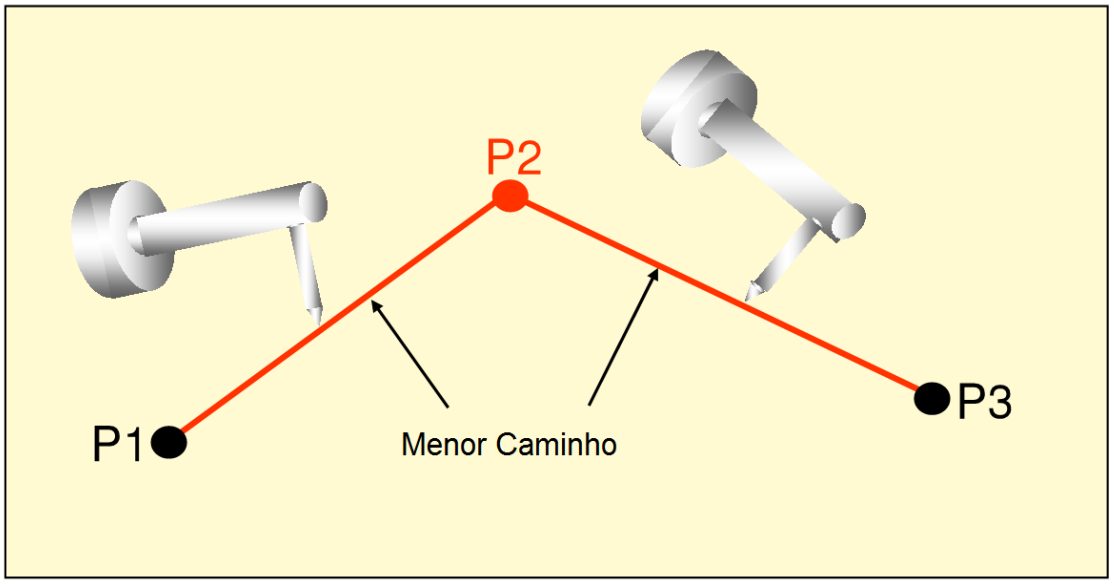

Figura 44. Movimentação Linear (CABRAL, 2012).

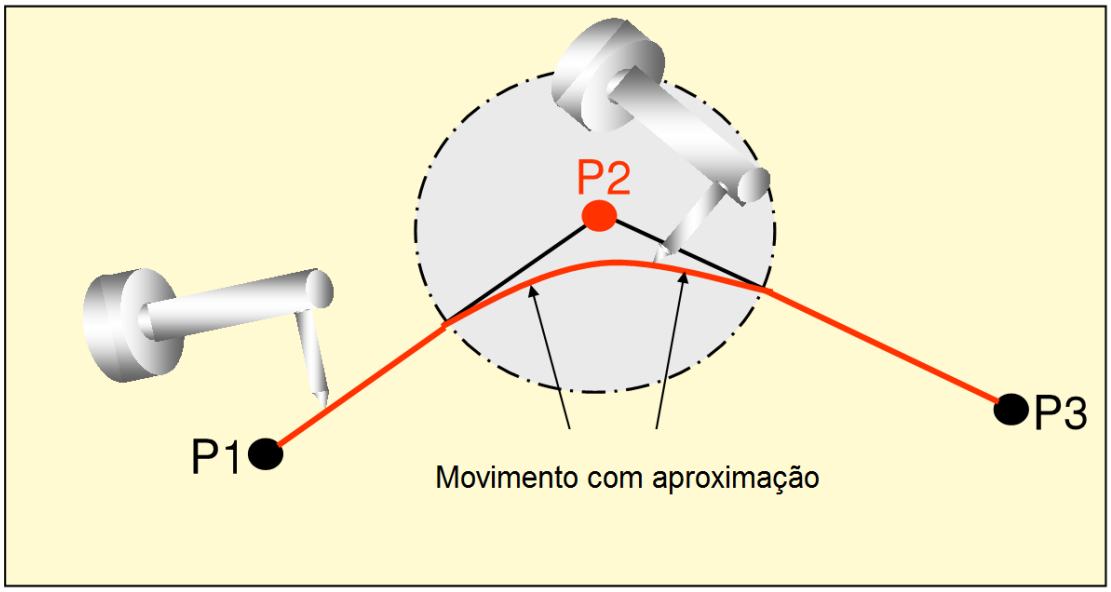

Figura 45. Movimentação Linear com aproximação (CABRAL, 2012).

4.3.5.2. GERAÇÃO E LEITURA DO CÓDIGO

Inicialmente definiu-se uma posição *Home* padrão. Essa posição é a posição padrão inicial das juntas do robô e define uma posição e orientação no espaço. Implementaram-se dois modos de gravação, o *Record Position* e *Record Movement*, que podem ser dados por meio do uso de botões ou de comandos de voz. O *Record Position* grava o ponto atual. O *Record Movement* grava a posição do robô em intervalos pré-definidos, por meio de um campo denominado “amostragem” nas preferências do programa controlado por um timer. Nesse modo as posições (se alteradas) vão sendo gravadas consecutivamente até que seja dado o comando *Stop Recording*. Toda sintaxe do programa é obedecida, desde a inicialização do programa, quanto à declaração das variáveis de posição a serem utilizadas e finalização do programa. Os dados de velocidade podem ser alterados por meio de botão se a gravação é no modo manual ou por meio de comandos de voz no modo sensor. A velocidade é dada na forma de porcentagem da velocidade máxima ou em m/s dependendo do modo de gravação, e é atualizada a cada ponto gravado, de acordo a com a sintaxe pré-estabelecida. Através do menu de preferências do programa, o usuário pode escolher gravar a movimentação em PTP, LIN ou LIN CONT, conforme detalhado na seção 4.3.5.1. Ao iniciar a gravação o robô automaticamente se movimenta para a posição *Home* pré-estabelecida e inicia a gravação de movimento a partir deste ponto. Isso ajuda a evitar problemas onde

uma gravação poderia ser inicializada em uma posição de colisão ou singularidade, posição tal que uma movimentação linear partindo da *Home* (no momento da execução do código) não seria capaz de atingir, algo que poderia causar um travamento.

Os dados são gravados em um arquivo de texto (*.txt*). Tal arquivo pode então ser aberto por meio de um botão seguido de uma caixa de seleção implementada. Além disso, criaram-se botões de *Pause* e *Stop*. Se o usuário pausar o código, o mesmo se interrompe onde está e continua caso seja pressionado o botão *Play*. Caso clique em *Stop,* o código é retornado para o início, e o robô volta para a posição *Home* que foi estabelecida. Por padrão, ao terminar a execução do código, o mesmo é pausado. Também pode ser notada a impressão de código na tela, que antes era estática, e era limitada pelo tamanho da janela, e agora foi feita uma função para a impressão linha a linha, auxiliada por botões.

4.3.6. COMANDOS DE VOZ

Constantemente, em todos os modos de operação e até na tela inicial, o áudio é analisado e as modificações são geradas no programa. O resultado obtido é sempre escrito na parte inferior do menu de modo a facilitar ao usuário o controle dos comandos dados.

Na tela inicial pode-se dar o comando de iniciar ou sair do programa. Ao entrar no programa, que tem como modo inicial padrão o modo manual, podem-se dar comandos para alterar o modo. É possível então dar comandos para movimentar o robô, a câmera, travar o movimento, gravar a posição, gravar o movimento, salvar código, apagar posições salvas, mover para *Home* etc. A Tabela 2 resume os principais comandos e seus efeitos.

Após diversos testes, devido a problemas de pronuncia e barulhos do ambiente, o grupo decidiu fazer a detecção de comandos de voz palavra por palavra. Ao pronunciar a primeira palavra de uma frase, a mesma é fixada e escrita na tela, e na tela são impressas todas as possíveis opções de palavras a serem ditas a partir da anterior. É dita uma Chamada (“Kinect”) + Ação + Objeto. Cada ação tem uma lista de Objetos que podem ser reconhecidos. Caso, após certo tempo, nenhuma palavra seja reconhecida, a frase construída até o momento é descartada. Os resultados obtidos com esse método foram muito mais precisos do

que aquele utilizado anteriormente, apesar de ter como efeito negativo uma maior demora na execução dos comandos, que agora dependem de três detecções de fala consecutivas.

Tabela 2. Lista de Comandos

Ordem dos Comandos			
1º	**2º**	**3º**	Resultado
Kinect	**Start**		Estando na tela de menu inicial, entra no programa no modo manual.
	Record	**Position**	Grava a posição atual em código Kuka.
		Movement	Inicia gravação do movimento do robô, em código Kuka.
	Stop		Para a gravação do movimento.
	Save		Salva o movimento gravado em código Kuka em um arquivo de texto.
	Operation	**Sensor**	Ativo o modo de operação no modo sensor.
		Automatic	Ativa o modo de operação no modo automático.
		Manual	Ativa o modo de operação no modo manual.
	Move	**Robot**	Ativa movimentação do robô.
		Camera	Ativa movimentação da câmera.
	Lock / Unlock	**Orientation**	Trava / Destrava a orientação fixa da ferramenta.
		Position	Trava / Destrava a posição do robô.
	Speed	**Slow**	Seleciona a velocidade lenta de movimentação do robô.
		Medium	Seleciona a velocidade média de movimentação do robô.
		Fast	Seleciona a velocidade rápida de movimentação do robô.

4.3.7. INTERFACE COM O USUÁRIO

4.3.7.1. ESTEREOSCOPIA

Um dos principais objetivos deste projeto é a construção de um ambiente imersivo, interativo e intuitivo de realidade virtual.

A estereoscopia, ou visão em três dimensões, está intimamente ligada à característica de imersividade. A imersividade será tão maior quanto mais próxima de uma experiência real for a experiência virtual. Informações quanto à

profundidade, distância, posição e tamanho de objetos podem ser adquiridas ou simuladas em ambientes virtuais, assim como em reais (ALVES, 1999).

O Nvidia 3D Vision (GATEAU, 2009) é um equipamento de visão estereoscópica em alta definição lançado em 2009. Ele gera imagens de duas perspectivas diferentes. Possui um emissor infravermelho de alta potência que transmite dados para um óculos sem fio. Também há a necessidade de um monitor com frequência de atualização de tela de ao menos 120Hz.

A estereoscopia é obtida quando os dados do modelo 3D são enviados ao *driver* da placa de vídeo. Ele renderiza a cena duas vezes, com um pequeno deslocamento horizontal da câmera, formando duas imagens, uma para cada olho. O monitor então deve exibir as imagens para o olho esquerdo em quadros pares e direito para os quadros ímpares. Por isso a necessidade de alta frequência de atualização. O emissor infravermelho então envia dados para os óculos que fecha cada uma das lentes alternadamente. Sendo assim a frequência final para cada olho é metade da frequência de atualização do monitor, ou seja, 60 Hz (GATEAU, 2009).

Utilizando os equipamentos do laboratório, que possuía computadores e monitores compatíveis com a tecnologia 3D Vision e o *kit* de óculos e emissor infravermelho, adaptou-se o projeto para a visão estereoscópica. Tal modo, denominado “Modo 3D”, é ativado automaticamente ao colocar o aplicativo desenvolvido em Tela cheia, caso o *driver* da placa de vídeo esteja configurado para tal (GATEAU, 2009). Para colocar o aplicativo em tela cheia, o usuário deve pressionar F11. Inicialmente é colocado o modo tela cheia com a resolução nativa do monitor. Ao pressionar F11 novamente, a resolução escolhida nas preferências do programa é forçada em tela cheia. Ao pressionar pela terceira vez, volta-se ao modo janela do aplicativo. Em ambos os modos de tela cheia, o modo 3D está disponível. Para desativá-lo, é necessário pressionar a combinação de teclas CTRL+T. Pressionando a mesma novamente volta a ativá-lo. Na Figura 46 apresenta-se uma foto do modo 3D sendo utilizado.

4.3.7.2. MENUS DO PROGRAMA

Quanto à interface, devido às diversas adições e correções que foram feitas, tornou-se necessário refazer a mesma. Foram feitos menus, telas e botões. Através das coordenadas e da monitoração do clique do *mouse* e do passar do *mouse*

(*hover*) em áreas especificadas da tela, controlou-se a execução de funções e eventos necessários para a execução do programa. As preferências do programa, como resolução, amostragem da captura e tipo de gravação de movimento são armazenadas em um arquivo e abertas automaticamente.

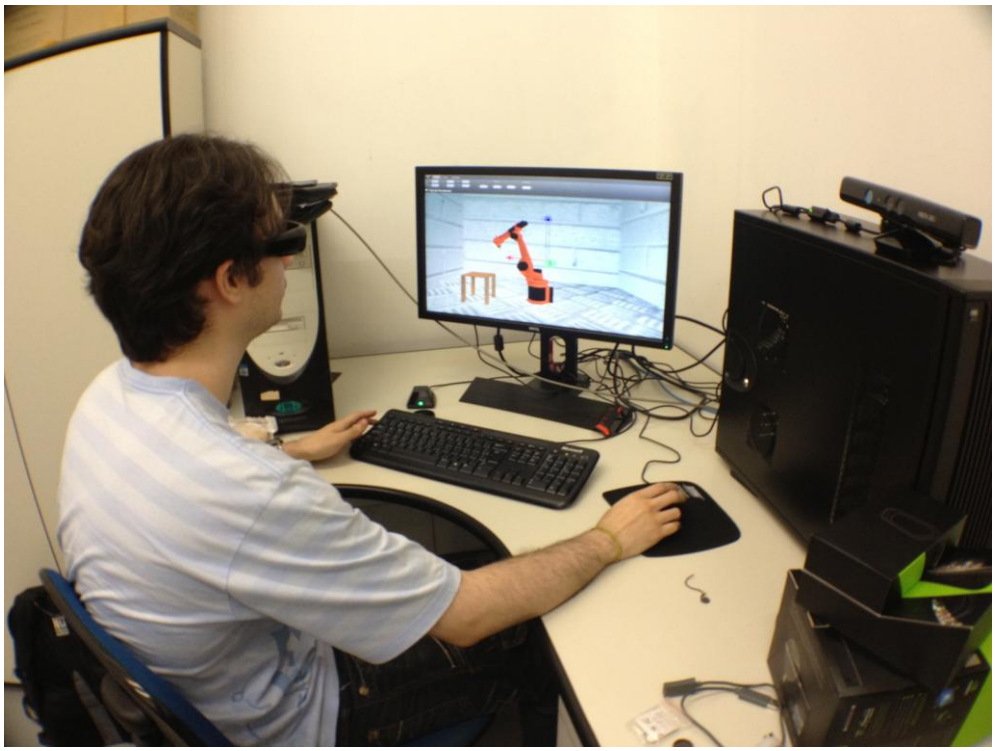

Figura 46. Utilizando o modo 3D.

A Figura 47 mostra o menu inicial com as opções de iniciar ou fechar o programa e a Figura 48 mostra o menu de seleção de preferências de programa. Um novo nome foi dado ao projeto, que passou a se denominar KALAS.

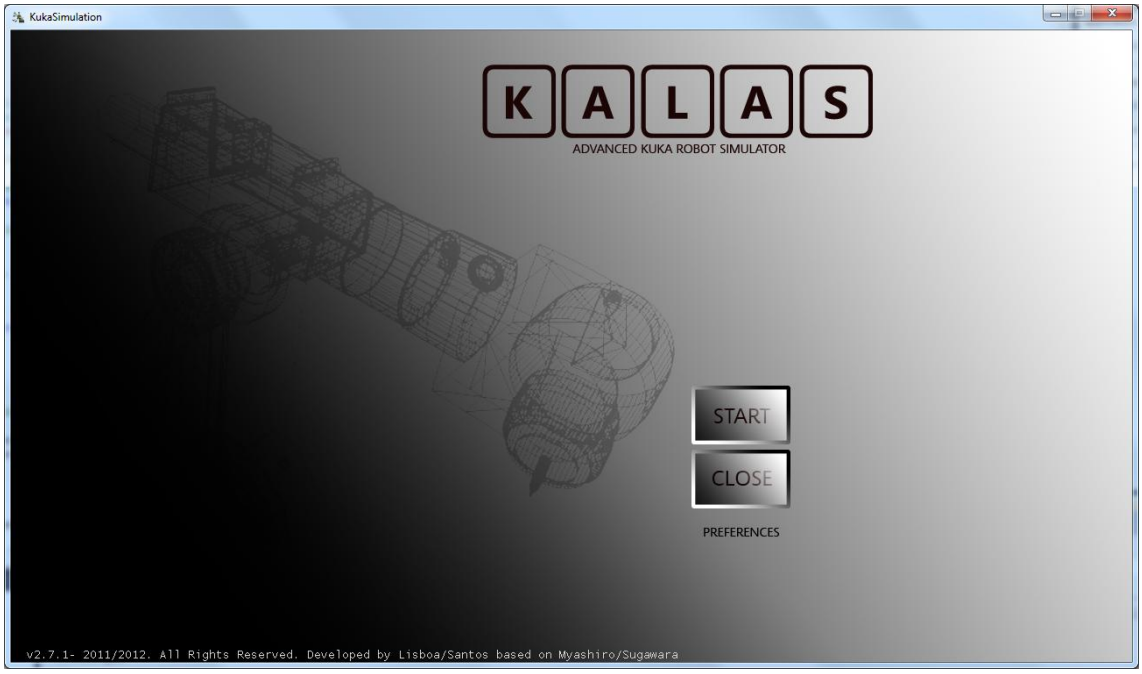

Figura 47. Imagem do menu inicial.

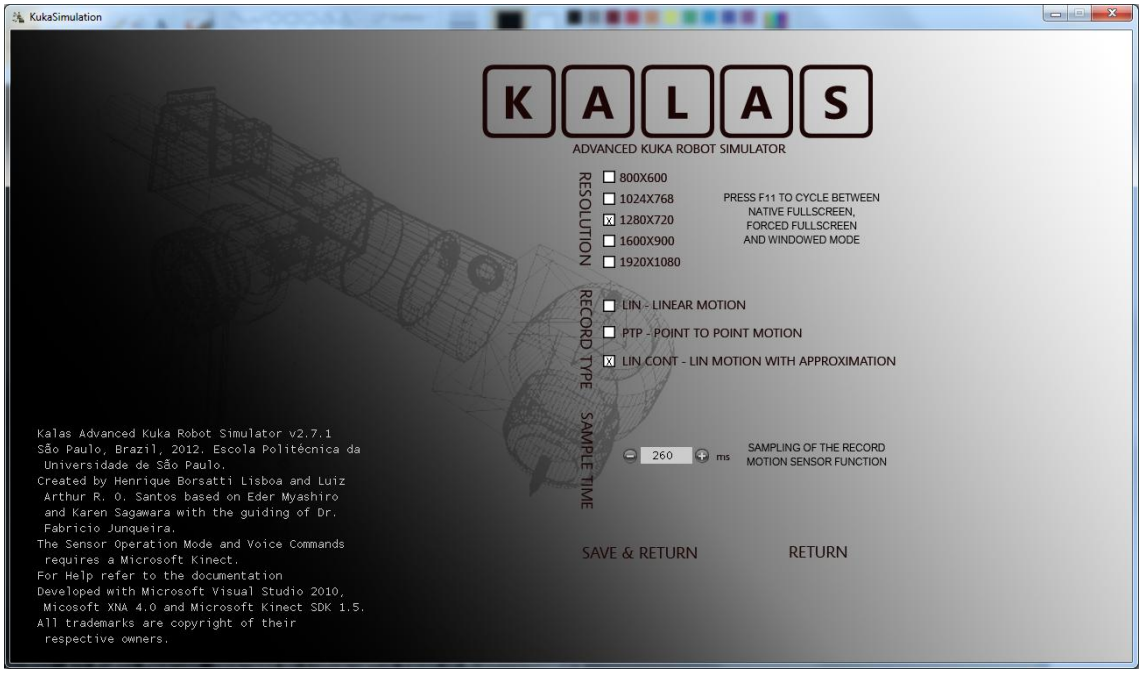

Figura 48. Imagem do menu de preferências.

Na tela inicial temos as opções de iniciar o programa (*Start*), fechar o programa (*Close*) e entrar na tela de preferências do usuário.

Na tela de preferências pode-se escolher a resolução da tela (entre uma lista de opções), escolher que modo de gravação será utilizado (Linear, Ponto a Ponto ou Linear com aproximação) e escolher a amostragem desse mesmo modo, ou seja, de

quanto em quanto tempo o programa armazenará a posição do robô no código. Ao clicar em Salvar e Retornar (*Save and Return*) as modificações selecionadas são aplicadas automaticamente e volta-se para a tela inicial. Todas essas preferências são armazenadas em um arquivo de texto que é lido na inicialização do programa, de modo que elas são salvas e utilizadas desse ponto em diante.

O efeito utilizado nos botões, que alteram com o passar do *mouse* (*hover*), é o exemplificado pela Figura 49.

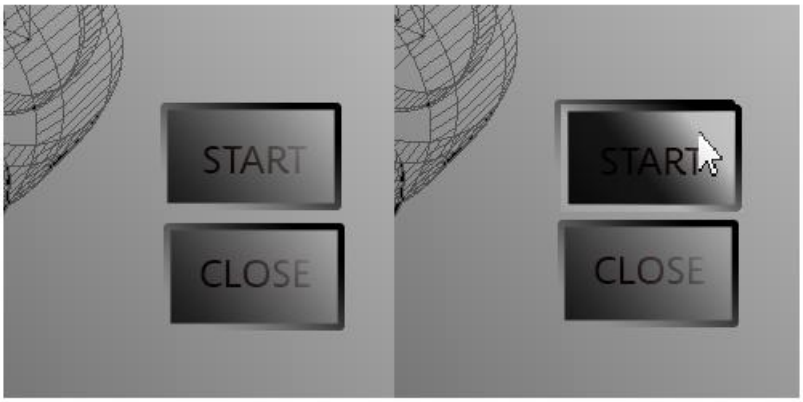

Figura 49. Detalhe do botão que se altera ao passar o *mouse*.

Utilizou-se uma organização de abas. As informações necessárias para cada modo de operação são impressas na tela. Há botões para as principais funcionalidades de cada modo. Exemplos de tela de cada modo de operação são dados nas Figuras 50, 51 e 52.

Ao clicar na primeira aba do menu o programa é pausado e volta-se ao menu inicial. Nas abas seguintes o usuário pode escolher que modo de operação deseja utilizar. Por padrão, o modo inicial é o modo Manual. Pode-se na área Juntas (*Joints*) selecionar o valor dos ângulos de cada uma das juntas do robô através de botões que incrementam ou decrementam os mesmos. A área Posição (*Position*) decrementa ou incrementa as coordenadas da ponta da ferramenta do robô, onde então se realiza a cinemática inversa e movimenta-se o mesmo. O campo velocidade escolhe a velocidade de movimentação, que é dada em função de uma porcentagem da velocidade máxima, fixada em 250 mm/s para o modo manual de velocidade reduzida utilizado, denominado “T1” pelo fabricante (KUKA ROBOTER, 2010). O espaço ao lado do balão de fala, na parte inferior da área do menu, mostra os comandos de voz reconhecidos.

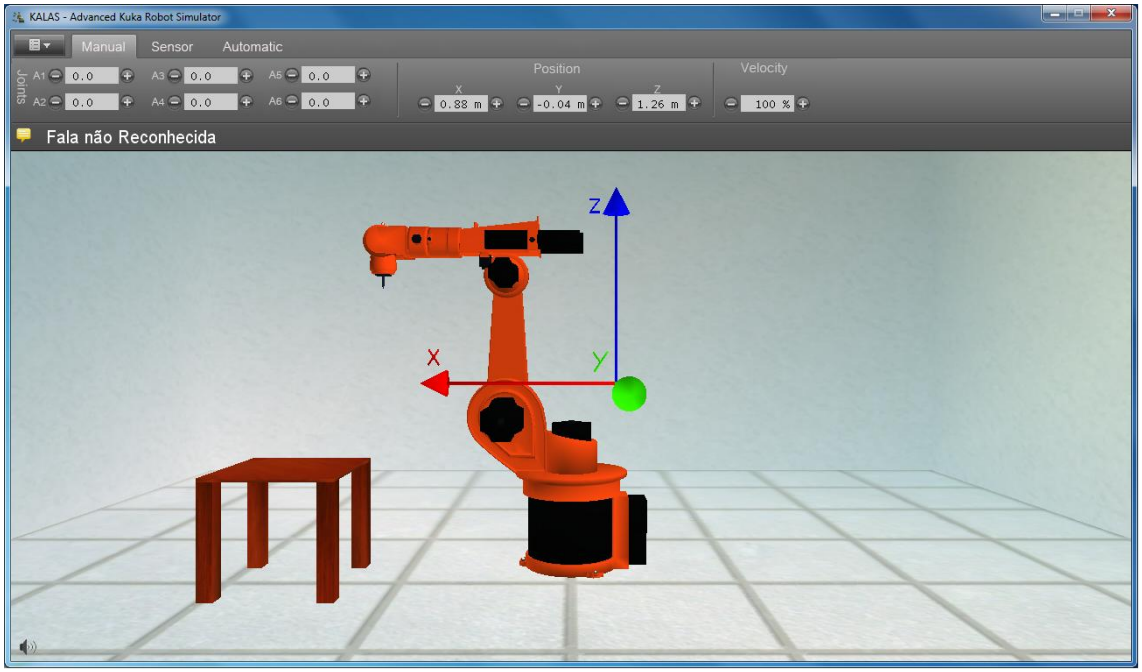

Figura 50. Exemplo de tela do modo Manual.

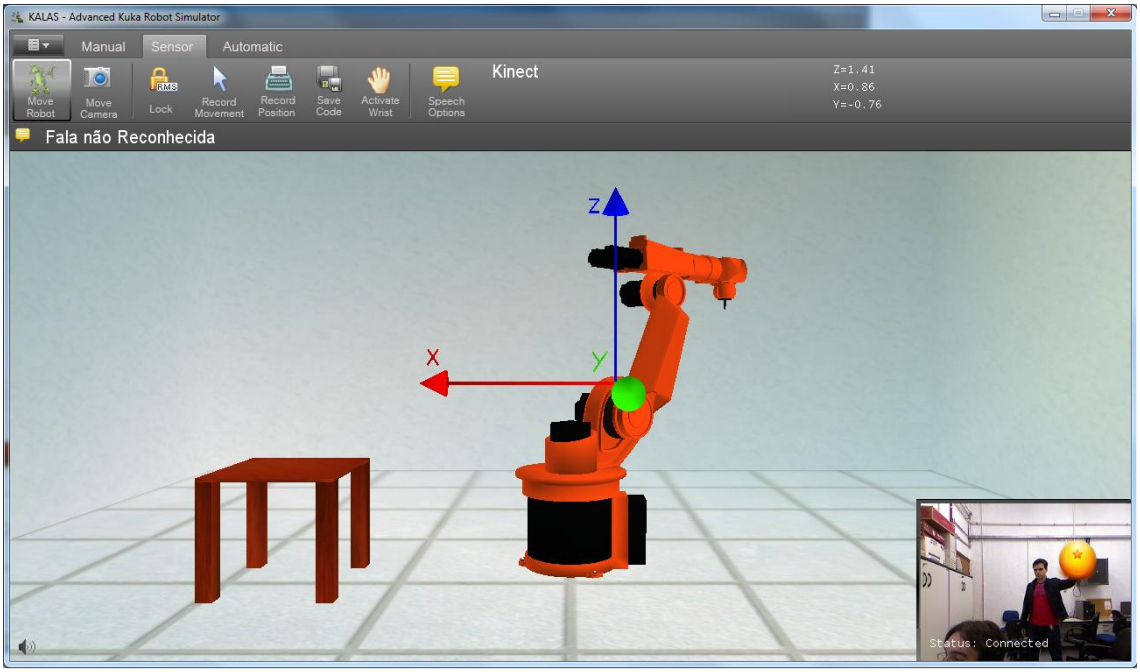

Figura 51. Exemplo de tela do modo Sensor.

Ao clicar na aba *Sensor* entra-se no modo de operação Sensor. É possível inicialmente como primeira opção escolher movimentar o robô ou a câmera. Ao movimentar o robô é possível travar a posição pelo botão *Lock*, Gravar o movimento pelo botão *Record Movement*, Gravar a Posição atual pelo Record Position, Salvar o código pelo botão *Save Code* e ativar a cinemática inversa do pulso esférico

(*Activate Wrist*), com orientação fixa. O texto reconhecido até o momento é impresso na parte inferior do menu e na área da direita são impressos todos os comandos de voz válidos a partir dos comandos já reconhecidos. Ainda mais à direita é mostrada a posição da ferramenta

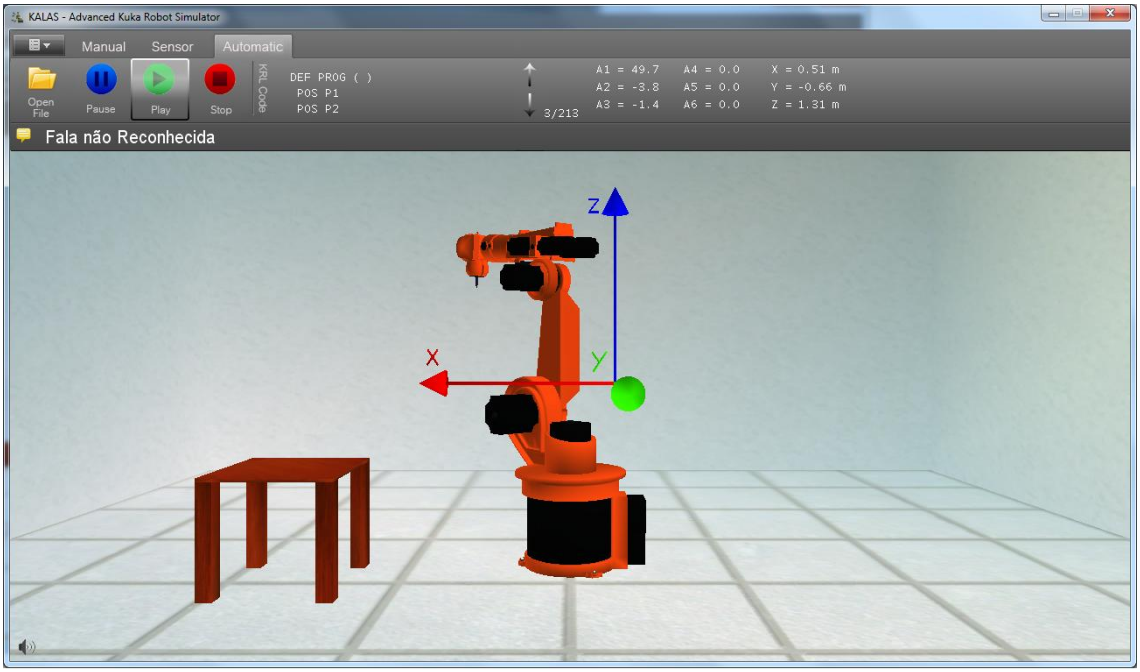

Figura 52. Exemplo de tela do modo Automático

No modo Automático o arquivo previamente gravado é automaticamente carregado. Ao entrar neste modo o robô se move automaticamente para *Home* e espera o comando de execução do código. Pode-se escolher abrir outro código pelo botão *Open File*. Ao clicar em *Play* o código é executado, onde é possível pausá-lo com o botão *Pause* ou interrompê-lo pelo botão *Stop*. Ao terminar a execução do código, o robô vai automaticamente para o estado de pausa. Para executar novamente o código, deve-se apertar *Stop*, que volta o robô para *Home*. Há ainda o campo *KRL Code,* onde o código é carregado e é possível navegar com o uso de setas.

Observe que muitos comandos de voz não são específicos apenas do modo sensor. Pode-se utilizar a cinemática inversa de pulso ou a gravação de movimento, por exemplo, em qualquer modo de operação, seja pelos comandos de voz ou pelo apertar de botões na tela do modo Sensor, seguido por posterior mudança de modo, pois a tela do modo Sensor é a única que possui um botão para tal função.

4.3.7.3. CONSIDERAÇÕES FINAIS

Ocorreram diversos empecilhos para o lançamento do programa na versão final na forma de um instalador que pudesse ser utilizado em qualquer máquina, sem a necessidade do Visual Studio. Problemas esses como o tratamento de erros e exceções pelo programa, que até o momento haviam sido negligenciados, e devido à mudança dos diretórios utilizados pelo programa na versão final. Outro grande problema foi o uso das referências importadas no programa do ANTLR, que apresentavam incompatibilidade com o Visual Studio na hora de gerar a versão *Release* e impossibilitavam o modo automático. Além disso, também foi necessário importar referências do Visual Studio e do Kinect SDK e gerar um instalador que verificasse os requisitos e instalasse todos os programas necessários no computador. Diversas modificações tiveram que ser feitas. O resultado foi um instalador completamente funcional do programa.

5. CONCLUSÕES

O objetivo do projeto foi a construção de um ambiente de Manufatura Virtual mais imersivo, interativo e intuitivo, com aplicações nas áreas de educação, treinamento, inspeção, validação e captação de conhecimentos. Para tal foi projetado um *software* de simulação de um sistema de manufatura virtual onde os dispositivos de entrada convencionais foram substituídos.

O projeto desenvolvido obtém sucesso em criar um ambiente de realidade virtual imersivo com o uso de sensores de movimento e tecnologia de estereoscopia.

Pode ser utilizado tanto em situações de aprendizagem em faculdades e escolas de engenharia como para treinamento de funcionários especializados sem a necessidade de supervisão de uso ou de interromper o funcionamento da máquina real no chão de fábrica ou na unidade de ensino.

A função de gravação de movimento pode ser utilizada em programação por aprendizagem passiva. Nela, o programador, que nesse caso é o usuário, movimenta o robô virtual pela trajetória desejada e grava as posições. Uma aplicação típica desse método de programação é a pintura. Implementou-se mais de um modo de gravação, que dependendo da amostragem selecionada, comporta-se como movimentação ponto a ponto ou movimentação contínua, com ou sem aproximação. Esse método de programação é útil e até imprescindível em algumas ocasiões tendo a vantagem de ser fácil de aprender e utilizar. Um trabalho posterior no código pode ser utilizado para otimizar as velocidades para cada trajetória (se necessário, pois é possível modificar a velocidade com comandos de voz) e excluir movimentos inúteis ou não intencionais. O código gerado então pode ser carregado e utilizado no robô real.

As coordenadas das juntas e as posições podem ser observadas e controladas na tela. Verifica-se a colisão entre todas as partes constituintes do robô e todo o ambiente. Grandes vantagens podem ser obtida na programação caso o ambiente simulado seja coerente com o ambiente onde o robô real está instalado.

Também se realizou a cinemática inversa do punho esférico que, dada as limitações do sensor e a necessidade de criação de um controle simples do robô, optou-se por uma orientação fixa. Tal orientação fixa pode ser utilizada, por

exemplo, em uma operação de pintura ou de escrita em uma superfície, assim como a manipulação de peças sobre uma superfície.

A detecção de fala obtida também é bem eficiente e serve ao seu propósito. Apenas tem como limitação a necessidade de uma boa dicção em inglês para correta identificação dos comandos. O resultado final foi um programa robusto, com cinemática inversa e geração e interpretação de códigos.

5.1. FUTURAS MELHORIAS

Um primeiro ponto a melhorar seria o da detecção de movimentos que, caso fosse mais preciso, poder-se-ia controlar o pulso esférico pelo próprio movimento do pulso humano. Com o desenvolvimento tecnológico e até atualizações do sensor Kinect e de seu *kit* de desenvolvimento tal funcionalidade terá sua implementação facilitada.

Quanto à interpretação de código, além de comando de posição, a linguagem de programação dos robôs Kuka possui comandos de orientação, lógicas condicionais, *loops*, possibilidade de chamar funções e outros programas etc. Um trabalho intensivo poderia ser realizado de modo a implementar no simulador todas essas funções presentes no robô real, de modo que ele pudesse operar como um simulador de código completo e independente. Também poderia ser implementada a movimentação em tempo real do robô através do programa, em lugar da utilização de códigos gravados.

Por fim, poderia ser realizado um estudo mais completo da física do robô, levando em conta a dinâmica e estática do mesmo, considerando as propriedades físicas e geométricas, calculando deformações e acelerações e se aproximando mais do comportamento do robô real, aumentando a confiabilidade dos resultados.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Alves, A. R. *Princípios Fundamentais da Estereoscopia*, UFSV – Santa Catarina, 1999, Disponível em <http://www.inf.ufsc.br/~visao/1999/aline/estereo.html>.

Banerjee, P.; Zetu, D. ***Virtual manufacturing***, New York: John Wiley & Sons, 2001.

Bezerra, E. **Princípios de Análise e projetos de sistemas com UML**, 1 ed. 380p., 2006.

Bó, A. P. L.; Hayashibe, M.; Poignet, P. *Joint Angle Estimation in Rehabilitation with Inertial Sensors and its Integration with Kinect*, **33rd Annual International Conference of the IEEE EMBS**, Boston, Massachusetts USA, pp. 3479-3483, 2011.

Braga, M. Realidade Virtual e Educação, **Revista de Biologia e Ciências da Terra**, Volume 1, 2001, Disponível em <http://eduep.uepb.edu.br/rbct/sumarios/pdf/realidadevirtual.pdf>

Burdea, G.; Coiffet, P. ***Virtual Reality Technology***, 2nd ed., A Wiley-Interscience publication, 2003.

Cabral, E. L. L. **Aulas de Laboratório**, 2012. Disponíveis em <http://poli.usp.br/d/PMR2560>

Cheng, K.; Bateman, R. J. *e-Manufacturing: characteristics, applications and potentials*, In. **Progress in Natural Science**, Vol. 18. No. 11, pp. 1323-1328, 2008.

Cheng, Y., Wang, S. *Applying a 3D virtual learning environment to facilitate student's application ability - The case of marketing*, In. Computers in Human Behavior, Vol. 27, No. 1, pp. 576-584, 2011.

Gateau, S. ***3D Vision Technology Develop, Design, Play in 3D Stereo***, 2009. Disponível em <http://developer.download.nvidia.com/presentations/2009/SIGGRAPH/3DVision_Develop_Design_Play_in_3D_Stereo.pdf>

Gorini, A.; Gaggioli, A.; Riva, G. *Letters: Virtual worlds, real healing*, In. **Science**, Vol. 318 No. 5856 p. 1549, 2007.

Grantcharov, T. P. *Is virtual reality simulation an effective training method in surgery?* In. **Nature Clinical Practice Gastroenterology & Hepatology**, Vol. 5, pp. 232-233, 2008.

Gutiérrez, M. A.; Vexo, F.; Thalmann, D. ***Stepping into Virtual Reality***, London, Springer, 2008.

Janoch, A.; Karayev, S.; Yangqing J; Barron, J. T.; Fritz, M.; Saenko, K.; Darrell, T. *A Category-Level 3-D Object Dataset: Putting the Kinect to Work*, In. IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp. 1168-1174, 2011.

Jiang, L.; Feng, X-L. *Design and application of intelligent virtual reality system*. In. **ICAT'06 Proceedings of the 16th International Conference on Artificial Reality and Telexistence**, pp. 89-92, 2006.

Kadir, A. A.; Xu, X.; Hammerle, E.; *Virtual Machine Tools and Virtual Machining – A Technological Review*, In. **Robotics and Computer-Integrated Manufacturing**, Vol. 27, No. 3, pp. 494-508, 2011.

Karaseitanidis, I.; Amditis, A.; Patel, H.; Sharples, S.; Bekiaris, E.; Bullinger, A.; Tromp, J. *Evaluation of virtual reality products and applications from individual, organizational and societal perspectives- the "VIEW" case study*, In. **International Journal of Human-Computer Studies**, Vol. 64, No. 3, pp. 251-266, 2006.

Khoshelham, K. *Accuracy Analysis of Kinect Depth Data*, In: **ISPRS Workshop Laser Scanning**, 2011.

Koç, M.; Ni, J.; Lee, J.; Bandyopadhyay, P. *Introduction to e-Manufacturing.* In. **Proceedings of The Industrial Information Technology Handbook**. 2005.

Kuka Roboter. *Kuka System Software 5.5 - Operating and Programming Instructions for System Integrators*, 2010. Disponível em http://sites.poli.usp.br/d/ PMR2560/Manual%20KUKA.pdf

Kybartaite, A.; Nousiainen, J.; Malmivuo, J. *Technologies and methods in virtual campus for improving learning process*, In. **Computer Applications in Engineering Education**, 2010.

Mahdjoub, M., Monticolo, D., Gomes, S., Sagot, J.C. *A collaborative Design for Usability approach supported by Virtual Reality and a Multi-Agent System embedded in a PLM environment*, In. **Computer-Aided Design**, Vol. 42, No. 5, pp. 402-413, 2010.

Martín-Gutiérrez, J.; Saorín, J. L.; Contero, M.; Alcaniz, M.; Pérez-López, D. C.; Ortega, M. *Design and validation of an augmented book for spatial abilities*

development in engineering students, In. **Computers & Graphics**, Vol. 34, No. 1, pp. 77-91 2010.

Microsoft. **Componentes do sensor Kinect**. Disponível em <http://support.xbox.com/pt-BR/kinect/setup-and-playspace/kinect-sensor-components>, Último acesso 10 jun 2012a.

Microsoft. ***Develop with Kinect to Change the World***, Disponível em <http://www.microsoft.com/en-us/kinectforwindows/>, Último acesso 10 jun 2012b.

Microsoft. ***Microsoft Investor Relations – Earnings Release FY12 Q2***, 2012, Disponível em <http://www.microsoft.com/investor/EarningsAndFinancials/Earnings/SegmentResults/EntertainmentAndDevicesDivision/FY12/Q2/performance.aspx>, Último acesso 10 jun 2012d.

Microsoft. ***Programming with the Kinect for Windows SDK***, 2011. Disponível em: <http://research.microsoft.com/en-us/events/fs2011/jancke_kinect_programming.pdf>. Último acesso 10 jun 2012.

Microsoft. **XBOX**, Disponível em <http://www.xbox.com/pt-BR>, Último acesso 10 jun 2012c.

Miyashiro, E. R; Sugawara, K. J. **Simulação de um robô virtual utilizando manufatura virtual**. Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, 2011.

Noonam, P. J.; Cootes, T. F.; Hallett, W. A.; Hinz, R. *The Design and Initial Calibration of an Optical Tracking System Using the Microsoft Kinect*, IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), pp. 3614-3617, 2011.

Payatagool, C. ***Theory and Research in HCI: Morton Heilig, Pioneer in Virtual Reality Research***, 2008. Disponível em <http://www.telepresenceoptions.com/2008/09/theory_and_research_in_hci_mor/>, Último acesso 10 jun 2012.

Petrolo, L.; Testi, D.; Taddei, F.; Viceconti M. *Effect of a virtual reality interface on the learning curve and on the accuracy of a surgical planner for total hip replacement*, In. **Computer Methods and Programs in Biomedicine**, Vol. 97, No. 1, pp. 86-91, 2010.

POLLYSOFT. ***Unified Modeling Language (UML)***. Disponível em <http://www.pollysoft.com.br/?m=fabrica&p=uml> Último acesso 5 jun 2012.

Porto, A. J. V.; Souza, M. C. F.; Ravelli, C. A.; Batocchio, A. Manufatura Virtual: conceituação e desafios, **Gestão da Produção**, Vol.9, No. 3, 2002.

Pouliquen, M.; Bernard, A.; Marsot, J.; Chodorge, L. *Virtual hands* and *virtual reality multimodal platform* to *design safer industrial systems*, In. **Computers in Industry**, Vol. 58, No. 1, pp. 46-56, 2007.

Queiroz, G. R. **UML: Visão Geral**, 2008. Disponível em <http://www.dpi.inpe.br/~gribeiro/apresentacoes/uml_2008_02_29.pdf>

Rheingold, H. ***Virtual Reality***, Simon & Schuster Inc., New York, 1992.

Roche, L.; Kurt, C.; Ahrens, S. *An avatar approach to distributed team training for mission operations*, In. **Acta Astronautica**, Vol. 67, No. 9-10, pp. 1164-1169, 2010.

Ruchanurucks, P. R. M.; Coundoul, A. *Kinect-based Obstacle Detection for Manipulator*, SI International, In. **IEEE/SICE International Symposium on System Integration (SII)**, pp. 68-73, 2011.

Seabra, R. D.; Santos, E. T. Utilização de Técnicas de Realidade Virtual no Projeto de uma Ferramenta 3D para Desenvolvimento da Habilidade de Visualização Espacial. In: **Educação Gráfica**, No. 9, Bauru, 2005.

Siciliano B.; Sciavicco L.; Villani L.; Oriolo G. ***Robotics: Modelling, Planning and Control***. London: Springer-Verlang, 632p, 2010.

Souza, M. C. F.; Sacco, M.; Porto, A. J. V. *Virtual Manufacturing as a way for the factory of the future*, In. **Journal of Intelligent Manufacturing**, Vol. 17, No. 6, pp. 725-735, 2006.

Stowers, J.; Hayes, M.; Bainbridge-Smith, A. *Altitude Control of a Quadrotor Helicopter Using Depth Map from Microsoft Kinect Sensor*, In. **Proceedings of the 2011 IEEE International Conference on Mechatronics**, Istanbul, Turkey, pp. 358-362, 2011.

Straka, M.; Hauswiesner, S.; Ruther, M.; Bischof, H. *Skeletal Graph Based Human Pose Estimation in Real-Time*, In **Proceedings of the British Machine Vision Conference**, BMVA Press, pp. 69.1-69.12, 2011.

Sung, R. C. W.; Ritchie, J. M.; Robinson, G.; Day, P. N.; Corney, J. R. *Automated design process modelling and analysis using immersive virtual reality*, In. **Computer-Aided Design**, Vol. 41, No. 12, pp. 1082-1094, 2009.

Tong, J.; Zhou, J.; Liu, L.; Pan, Z.; Yan, H. *Scanning 3D Full Human Bodies Using Kinects*, In. **IEEE Transactions on Visualization and Computer Graphics**, Vol. 18, No. 4, pp. 643-650, 2012.

UML-DIAGRAM. Disponível em <http://www.uml-diagrams.org/uml-24-diagrams.html>, Último acesso 5 jun 2012.

Vargas, T. C. S. **A história de UML e seus diagramas**. Universidade Federal de Santa Catarina, Departamento de Informática e Estatística, Santa Catarina, 2008.

Wu, B.; Wang, A. I.; Ruud, A. H.; Zhang, W. Z. *Extending Google Android's Application as an Educational Tool*, In. **Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL)**, pp. 23-30, 2010.

Yu, X.; Wu, L.; Liu, Q.; Zhou, H. *Children Tantrum Behavior Analysis Based on Kinect Sensor*, In. **Third Chinese Conference on Intelligent Visual Surveillance (IVS)**, IEEE, pp. 49-52, 2011.