Universidade Federal de Ouro Preto - UFOP
Escola de Minas
Colegiado do curso de Engenharia de
Controle e Automação - CECAU

Érick Victor de Oliveira Guedes

Controle de Manipuladores com Base em Gestos

Monografia de Graduação em Engenharia de Controle e Automação

Ouro Preto, 2017

Érick Victor de Oliveira Guedes

Controle de Manipuladores com Base em Gestos

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Prof. M.Sc. João Carlos Vilela de Castro

Ouro Preto, 2017

G924c Guedes, Érick Victor de Oliveira.
Controle de manipuladores com base em gestos [manuscrito] / Érick Victor de Oliveira Guedes. - 2017.

46f.: il.: color; tabs.

Orientador: Prof. MSc. João Carlos Vilela de Castro.

Monografia (Graduação). Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia de Controle e Automação e Técnicas Fundamentais.

1. Controle automático - Programas de computador. 2. Controle incremental de movimento - Gestos. 3. Manipuladores (Mecanismo). 4. Realidade virtual - Leap Motion. 5. Robos - Sistemas de controle. I. Castro, João Carlos Vilela de. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 681.5

Catalogação: ficha@sisbin.ufop.br

Monografia defendida e aprovada, em 09 de fevererio de 2017, pela comissão avaliadora constituída pelos professores:

Prof. M. Sc. João Carlos Vilela de Castro - Orientador

Profa. MSc. Adrielle de Carvalho Santana – Professora Convidada

Prof. M. Sc. Danny Augusto Vieira Tonidandel – Professor Convidado

Este trabalho é dedicado a todos aqueles
que tiveram paciência e me apoiaram durante os anos de estudos.

Resumo

Assim como o uso de robôs nas indústrias tem aumentado incessantemente nas últimas décadas, os mecanismos de controle destes robôs têm evoluído de forma cada vez mais rápida. O uso de gestos para realizar atividades de controle tem sido destaque nos últimos anos, principalmente após a popularização do *Microsoft Kinect*, e representa uma forma prática e intuitiva de controle. Neste trabalho é feito um estudo sobre robôs e controle com base em gestos, bem como um experimento utilizando o controlador *Leap Motion* e o simulador *RoboDK*, visando identificar a viabilidade desta aplicação. O experimento apresentou boa resposta e os dados coletados mostram um grande espaço para melhorias antes da utilização do método em massa, sendo a visualização e o reconhecimento de gestos quando há sobreposição de objetos, as maiores dificuldades identificadas.

Palavras-chaves: controle, manipulador, gestos, leap motion.

Abstract

In the same way as the use of robots in the industry has largely increased in the last few decades, the control mecanisms of these robots have evolved progressively. The use of gestures to perform control activities has taken pride of place in the last few years, mainly after the popularization of the *Microsoft Kinect*, and represents a practical and intuitive form of controling devices. On this paper, a study about robots and gesture based control is done, as well as an experiment using the *Leap Motion* controller and the *RoboDK* simulator, aiming at identifying this application's viablility. The experiment presented a good response and the colected data shows a great room for improvements before the use of the method in scale, being the visualization and the gesture recognition when there are overlaping objects, the biggest identified obstacles.

Key-words: control, manipulator, gestures, leap motion.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

No fim do século XVIII e início do século XIX, a Revolução Industrial contribuiu fortemente com o êxodo rural na Europa e o início das indústrias com uso de máquinas a vapor para execução das etapas de produção. Já para o final do século XIX, a evolução da máquina elétrica e dos compressores de ar, utilizados para o acionamento de máquinas pneumáticas, foi o próximo passo para a introdução do robô que conhecemos hoje, o que aconteceu em meados do século XX (HAGIS, 2003).

1.1 Controle de manipuladores robóticos

Com o incessante uso de máquinas e desenvolvimento de tecnologias diversas, é cada vez mais frequente ver automatizados processos que sempre foram feitos manualmente, desde a abertura de um portão ou do passar de marchas de um carro, até a realização de um processo de soldagem. Esta automatização funciona por meio de algum tipo de programação, ou comandada, com um operador enviando comandos por meio de algum dispositivo. Em fábricas automotivas, por exemplo, é possível ver vários robôs fazendo a colocação de peças nos carros, sem a intervenção de uma pessoa, enquanto a forma mais comum de abrir um portão em residências é por meio do acionamento de um controle remoto.

O controle de equipamentos tem evoluído constantemente e de forma cada vez mais rápida, do controle manual de válvulas, passando pelo governador centrífugo de James Watt, que oferece um controle proporcional para a velocidade de um motor, até os PLCs (sigla em inglês para Controladores Lógicos Programáveis), que oferecem grande flexibilidade de aplicações e são programados de acordo com a necessidade do processo (HAYDEN; ASSANTE; CONWAY, 2015).

Com a programação dos manipuladores deixando de ser feita com modificações no *hardware* e passando a ser por *software*, a facilidade de se utilizar robôs em diferentes aplicações aumentou, assim como sua adoção. Hoje, a programação *offline* é cada vez mais utilizada, principalmente quando a aplicação demanda longos períodos de desenvolvimento dos programas (SILVA, 1996). Na programação *offline* o programa é desenvolvido, testado em um simulador em tempo real e, após sua conclusão, é enviado ao robô, dispensando seu uso durante o período de desenvolvimento do código. No entanto, a utilização de texto para descrever as coordenadas do robô não é tarefa simples, por isso outros métodos têm sido pesquisados e desenvolvidos nos últimos anos, um deles é a detecção de gestos, que

são movimentos bem definidos usados para exprimir ideias.

Os movimentos feitos com as mãos para realizar os mais diversos tipos de atividades já são naturais, por estarmos acostumados a eles. Usar alguma ferramenta intermediária adicional, como um *joy-stick*, para fazer tal atividade é, geralmente, contraintuitivo. Com o uso dos sensores de gestos, estas atividades podem ser realizadas por robôs controlados por uma pessoa realizando os mesmos movimentos que ela faria se não houvesse um robô.

Sensores de movimentos já vêm sendo estudados e desenvolvidos há alguns anos, com um grande crescimento recente do interesse nos sensores de gestos. O uso de sensores de mãos baseados em visão tem ganhado bastante espaço na mídia e em estudos, principalmente após a popularização do *Microsoft Kinect*. Um sensor de gestos pode ser usado nas mais diversas tarefas, desde o controle de um jogo eletrônico, passando por controles de dispositivos, como celulares, carros, luzes, trancas e cortinas, até o controle de máquinas complexas, como braços industriais ou hospitalares.

O uso de gestos para o controle de robôs se mostra prático e intuitivo, sendo uma escolha que pode otimizar o tempo de programação inicial dos manipuladores. A adoção deste método é de interesse de toda a indústria, por economizar tempo e diminuir a dificuldade do processo sempre que um robô precisa ter sua função alterada, já que permite que o programador pense mais no movimento e na posição do que nos controles. Apesar disto, seu uso ainda não é amplo, assim como sua usabilidade e viabilidade não estão confirmadas.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho objetiva o estudo e aplicação do controle de robôs manipuladores usando sensores de gestos, visando identificar a viabilidade desta metodologia para a execução da referida tarefa, que não é simples.

1.2.2 Objetivos Específicos

De forma mais detalhada, os objetivos buscados são:

- A contextualização teórica sobre o assunto, por meio de uma revisão bibliográfica sobre robôs, seu controle e simulação em computadores;
- O levantamento teórico de metodologias já utilizadas para captação de movimentos;
- Desenvolvimento de um exemplo prático utilizando o simulador *RoboDK* e o controlador *Leap Motion*.

1.3 Estrutura do trabalho

O presente trabalho está organizado em 5 capítulos. O Capítulo 1 traz uma introdução sobre o assunto abordado, com uma breve história do desenvolvimento dos robôs e da detecção de gestos, bem como apresenta os objetivos do trabalho.

A revisão da literatura sobre os principais tópicos abordados é feita no Capítulo 2, falando sobre robôs, programação e simulação de robôs, sensores de gestos, assim como a utilização de gestos para controle de robôs. No Capítulo 3 é apresentado um experimento em que foi feita uma programação *offline*, utilizando o controlador *Leap Motion* e o simulador *RoboDK*, assim como as etapas realizadas e as observações feitas durante o desenvolvimento. Os resultados do experimento são apresentados no Capítulo 4. O Capítulo 5 contém uma síntese do que foi apresentado ao longo do trabalho, outras considerações relevantes e sugestões para trabalhos futuros.

Ao final, estão disponíveis os apêndices, com os dados coletados durante os ensaios do experimento.

2 Revisão bibliográfica

2.1 Robôs

Um robô é comumente definido como um dispositivo capaz de realizar tarefas dispensando total ou parcialmente o controle humano, possuindo sensores e atuadores para interação com o ambiente. Este conceito vem sendo utilizado há um bom tempo pela humanidade, ainda que sem um termo específico para ele. Uma das ideias mais antigas que se conhece sobre autômatos, máquinas que se operam de maneira automática, data de 350 a.C., chamada de "O Pombo" e criada por Arquitas de Tarento, contemporâneo de Platão. O primeiro projeto documentado de um autômato com características humanas foi feito por Leonardo da Vinci, por volta de 1495, porém não há confirmação de que o projeto tenha sido colocado em prática. Desta forma, Jacques de Vaucanson é tido como o criador do primeiro robô funcional, um tocador de flautas construído em 1738 (SANTOS, 2014).

A palavra *"robot"*, que significa robô, em português, no entanto, só veio a aparecer na literatura em 1921, em uma peça de teatro chamada *R.U.R.* (iniciais de *"Rosumovi Univerzální Roboti"* e traduzida para o português como *"A Fábrica de Robôs"*), do escritor checo Karel Capek, que se inspirou na palavra *"robota"*, checo para corvéia[1] . O robô moderno demoraria ainda um pouco mais para aparecer, pela década de 1940, com a chegada dos computadores (HAGIS, 2003). Em 1941, Isaac Asimov utilizou a palavra "robótica" para descrever a ciência que trata da teoria e aplicação dos robôs.

Com a expansão da indústria automotiva, no fim da década de 1950 e início da década de 1960, o conceito de robôs começou a se tornar popular e robôs industriais começaram a ser utilizados para facilitar as operações nas fábricas. Os robôs industriais não têm semelhanças físicas com os seres humanos, mas podemos relacioná-los, em geral, com nossos braços e mãos, capazes de operar ferramentas com maior precisão e melhor repetibilidade que os operadores humanos, além de suportarem maiores cargas (HAGIS, 2003).

Inicialmente, para que pudesse realizar uma tarefa diferente, o robô precisava ser alterado fisicamente, o que dificultava sua operação e aumentava os custos. Contudo, com o desenvolvimento da microeletrônica e o aparecimento dos primeiros PLC's, os robôs passaram a ser programados por *software*, o que trouxe maior flexibilidade, reduziu os

[1] Corvéia é a prestação de trabalhos gratuitos ao senhor feudal para obtenção de parte ou uso de terras do feudo.

custos, por não precisar de alterações físicas para cada tarefa, e proporcionou um aumento na adoção deste tipo de máquina pelas indústrias (MADALENO, 2011).

O uso de robôs mais facilmente programáveis traz diversas vantagens, como a substituição de pessoas em tarefas desagradáveis, repetitivas e até mesmo perigosas. Além disso, os robôs propiciam uma grande facilidade de padronização, aumento da qualidade, da produtividade e podem trabalhar por muitas horas ininterruptas.

Apesar de ter trazido maior flexibilidade e facilidades, a programação de robôs apresenta particularidades que a tornam mais complicada, se comparada com a programação de outros tipos de máquinas. De acordo com Bernard et al. (2012):

> "[...] a programação de robôs tem algumas peculiaridades:
>
> A programação de robôs diz respeito à geração de movimentos, na maioria das vezes bastante complexos. Estes movimentos são muito difíceis de imaginar para os seres humanos, sem a ajuda do próprio robô ou de um sistema de simulação e animação gráficas;
>
> Os dados geométricos do mundo real (por exemplo, posições e orientações dos componentes a serem manipulados) apresentam desvios significativos dos dados usados no seu modelo para programação *offline*; daí que muitas vezes sejam usados sensores para medir tais quantidades geométricas (os erros). Estas leituras são a base de um programa para compensação dos desvios;
>
> Tipicamente, o robô tem que estar sincronizado com dispositivos periféricos, tais como: sensores, atuadores finais, alimentadores, fixações, etc.
>
> [...] Resumindo, um método de programação de robôs deve ser capaz de:
>
> ser simples o suficiente para ser facilmente aprendido e manipulado;
>
> ser orientado a problemas: o método de programação deve ser adaptado à imaginação do usuário e à aplicação particular;
>
> suportar a programação *offline* do robô, de forma a evitar o uso da cara célula do robô para sua programação."

Com o aumento da produtividade usando robôs, o tempo gasto para a programação destes passou a ser um ponto relevante, já que a programação de um robô pode levar até mesmo semanas. Para reduzir este tempo, em que a produção é interrompida, surgiu a programação *offline*.

2.1.1 Programação *online* e *offline*

Inicialmente, a programação de robôs era feita apenas utilizando o robô. Esta forma de programação é chamada *online*. A atividade principal nesta forma de programação é a definição de trajetórias e pode ainda ser dividida entre manual e automática.

Na programação manual, o programador leva o robô manualmente às posições que devem ser memorizadas, geralmente quando se tem uma trajetória complexa e se deseja definir uma sequência de pontos no espaço. Já na programação automática, um

dispositivo do tipo *joy-stick* é utilizado para definir alguns pontos discretos no espaço, como em aplicações do tipo *pick-and-place*. A programação *online* tem como vantagens a fácil utilização e rápido aprendizado, porém a necessidade de utilizar o robô para ser realizada é uma grande desvantagem, visto que isto pode fazer com que a produção seja interrompida por vários dias (SILVA, 1996).

Para evitar que o robô fique indisponível para realizar tarefas enquanto é programado, a programação *offline*, ou seja, sem o uso do robô durante o desenvolvimento, tem se desenvolvido cada vez mais, especialmente para uso em aplicações complexas e que demandam longos tempos de desenvolvimento, uma vez que ela permite que o robô seja usado para outras tarefas durante o tempo de programação. Inicialmente, uma solução mista começou a ser utilizada. Nesta abordagem, o código é escrito no computador, tal como no desenvolvimento de outros *softwares*, porém, dada a dificuldade em se determinar textualmente as coordenadas dos pontos, a programação do robô é finalizada utilizando a forma *online* para gravação de tais pontos. Para reduzir esta etapa, um sistema de simulação gráfica pode ser utilizado. Inicialmente, as simulações permitiam que o programa escrito *offline* fosse apenas visualizado e testado. *Softwares* mais recentes de simulação utilizam uma abordagem ainda mais eficiente, permitindo a visualização dos movimentos do robô de forma imediata e interativa. Com isso, as definições das trajetórias podem ser testadas e corrigidas antes mesmo de serem enviadas ao robô (SILVA, 1996).

O desenvolvimento da programação *offline* de robôs propicia uma inserção mais rápida de uma nova aplicação e até mesmo mais segurança para os operadores. Em termos de custos, os benefícios oferecidos são grandes, sendo que um robô pode ficar até 85% menos tempo fora de produção com o uso de programação *offline* no lugar da *online*. Um exemplo é um estudo da Chrysler, que demonstrou uma redução no tempo de programação de algo entre 12 e 18 horas para 6,5 horas por robô (WITTENBERG, 1995).

Apesar destas vantagens, empresas de pequeno e médio porte não têm uma utilização tão ampla da programação *offline*, devido ao alto custo de um pacote de desenvolvimento, tornando difícil uma justificativa financeira para baixos volumes de produção. O pacote de desenvolvimento para programação *offline* consome muito tempo para ser produzido e deve ser customizado para cada tipo de aplicação e equipamentos utilizados. Isto faz com que seja necessário uma certa experiência por parte dos engenheiros, o que não é tão simples de encontrar, e uma grande precisão na modelagem do robô e de sua célula de trabalho, requerendo ainda sensores e métodos de calibração adicionais em muitos casos (PAN et al., 2012).

Como regra geral, é possível estabelecer algumas etapas básicas da programação *offline*, a começar pela obtenção de um modelo CAD (*Computer-aided design*, ou Desenho Assistido por Computador) em 3D (três dimensões) do robô e das partes com as quais ele irá interagir. Em muitos casos, o modelo do robô pode ser conseguido com seu fabricante,

no entanto esta etapa pode ser trabalhosa caso não seja possível conseguir o modelo pronto. A etapa seguinte é a de criação de marcações de posições do robô, como a inicial, final, de aproximação, entre outras. Aqui, muito tempo é gasto e a especificação deve ser feita manualmente, usando os dados do modelo CAD. Depois, deve ser feito o planejamento da trajetória, visto que a cinemática inversa de robôs articulados apresenta, muitas vezes, várias soluções. O planejamento deve ser feito considerando o alcance, colisões e quantidade de transições de configurações, sendo que a maioria dos *softwares* de programação *offline* não são capazes de fazer este planejamento automaticamente, ficando a cargo do programador ou necessitando o uso de outro *software* ou API (*Application Programing Interface*, ou Interface de Programação de Aplicações). A etapa de pós-processamento pode incluir a adição de sinais de controle de entrada e saída, ajuste fino do caminho e conversão da linguagem de programação específica do robô. Por requerer compatibilidade entre robôs de diferentes marcas, este é um passo geralmente necessário em *softwares* genéricos de programação *offline*. Posteriormente vem a etapa de simulação, que permite a verificação do programa sem o uso de um robô real e é considerada uma ferramenta significativa trazida pelo *software* de programação *offline*. Finalmente, a etapa de calibração é executada. Esta etapa é necessária devido às diferenças existentes entre a geometria nominal e a real do robô e seu ambiente (PAN et al., 2012).

Para uma programação *offline* eficiente, um ambiente simulado em computador é de grande necessidade e importância. A maioria dos fabricantes de robôs oferecem um *software* específico para programação *offline*, o que permite maior compatibilidade com o *hardware* a ser programado. De acordo com Madaleno (2011), os principais requisitos para uma programação *offline* eficaz são:

- Uma representação gráfica do modelo do "mundo", por sistemas de CAD e CAM (*Computer-aided manufacturing*, ou Manufatura Auxiliada por Computador);
- Que o sistema detenha informações sobre o processo ou tarefa a ser programado;
- Uma boa representação da geometria, cinemática e dinâmica dos robôs;
- O sistema deve ser capaz de reconhecer e utilizar todos os dados de forma correta;
- Verificação em busca de possíveis erros nos programas produzidos;
- Existência de uma interface adequada de comunicação ao controlador do robô ou ao sistema de programação *offline*.

Mesmo atendendo a estes requisitos, o método estudado ainda pode apresentar problemas, como o uso inadequado do método de programação, modelo geométrico e do robô, falta de padronização de linguagens de programação e interfaces de desenvolvimento

entre diferentes robôs, de diferentes marcas. Além disso, os fatores mais complicados de analisar e melhorar são os erros e desvios no sistema real. Estes podem decorrer de diferentes formas de fabricação e até mesmo do desgaste dos componentes. Nos robôs, a falta de rigidez da estrutura, falta de tolerâncias na construção ou a diferença entre robôs, até mesmo da mesma marca, são os principais causadores destes problemas. O controlador do robô pode ter problemas com a resolução e a célula de trabalho deve ter todos os componentes localizados com precisão (MADALENO, 2011).

Diante do exposto, antes de fazer o *download*[2] de um programa para o robô, uma calibração é necessária. Cada robô apresenta uma assinatura individual, criada pelas tolerâncias de fabricação, desgastes e outras consequências de seu uso. Esta assinatura é detectada por meio da medição de diversas coordenadas em 3D e usando um *software* de calibração. Com este procedimento, um modelo mais preciso do robô é conseguido e a programação se torna mais acertada (WITTENBERG, 1995).

Com a vantagem de oferecer maior flexibilidade e compatibilidade entre diferentes fabricantes, existem também *softwares* de programação *offline* genéricos, que podem ser capazes de simular várias interações de processos de manufatura, recursos e manutenção de produtos.

Devido ao alto custo e acessibilidade limitada de *softwares* de programação *offline* comerciais, diversos pesquisadores tem desenvolvido *softwares* alternativos e gratuitos, alguns baseados em *softwares* de CAD já existentes e outros iniciados a partir dos fundamentos.

2.2 Simulação

Neste trabalho será apresentado um exemplo de programação *offline* utilizando o *software* de simulação *RoboDK*. Criado por Albert Nubiola, inicialmente com o nome *RoKiSim*, o *RoboDK* possui uma grande biblioteca de robôs e permite a programação de qualquer um deles por meio da linguagem *Python*, além de fazer a calibração e exportar os programas para os robôs. Sua interface, pode ser vista na Figura 1. A possibilidade de utilizar uma linguagem de programação que muitos programadores já estão habituados permite um trabalho mais prático e rápido, com a conversão dos comandos para a linguagem específica do robô ficando a cargo do pós-processamento do *RoboDK* (NUBIOLA, 2016).

[2] Download é o termo usado para o ato de copiar um ou mais arquivos de um local, geralmente da *internet*, para outro.

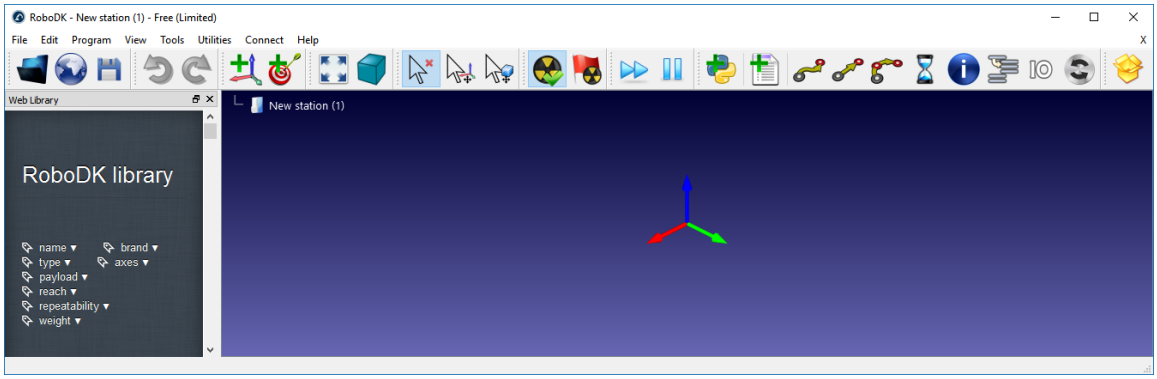

Figura 1: Interface do *RoboDK*

2.3 Sensor de Gestos

Inicialmente, sensores baseados em visão usavam marcadores nos dedos para que fossem identificados pelo algoritmo. Usando duas ou mais câmeras, as posições eram correlacionadas e as coordenadas 3D podiam ser calculadas. Para esta abordagem, câmeras são usadas para captar a imagem e um algoritmo computacional fica responsável por identificar os marcadores presentes nos dedos e definir a posição e o movimento de cada um dos marcadores (STURMAN; ZELTZER, 1994).

Um outro método utilizado para reconhecer os movimentos consiste no uso de materiais condutores, que variam a resistência conforme a curvatura da mão e dos dedos. Acoplados a uma luva, é possível determinar os movimentos feitos pelos dedos e pelas mãos, ainda que não seja possível identificar movimentos da mão no espaço tridimensional (LORUSSI et al., 2005).

No entanto, o uso de qualquer dispositivo acoplado às mãos ou braços pode gerar inconveniências e, assim, alternativas vêm sendo encontradas, como o uso do *Kinect* e do *Leap Motion*. Estes dispositivos utilizam de câmeras e emissores infravermelho (IR) para fazer a leitura dos movimentos.

O *Leap Motion* é um dispositivo sensor de movimentos 3D usado para detectar gestos das mãos. Para isso, ele usa duas câmeras e três LEDs emissores infravermelho, o que o permite alcançar uma precisão de aproximadamente 0,01mm, desempenho melhor que o do *Microsoft Kinect*. A Figura 2 mostra a disposição das câmeras e dos emissores no dispositivo. As imagens capturadas pelas câmeras são pós-processadas no computador, onde o ruído é removido e são criados modelos das mãos, dedos e ferramentas pontudas utilizadas (WEICHERT et al., 2013).

Atualmente, o reconhecimento de gestos é usado em diversas áreas e em várias aplicações, ainda que a maioria ainda esteja em fase de desenvolvimento. Um exemplo de aplicação na educação, é a identificação de gestos da língua brasileira de sinais (Libras) (MELO, 2015). Em veículos, pode ser usada para controlar o rádio, o limpador

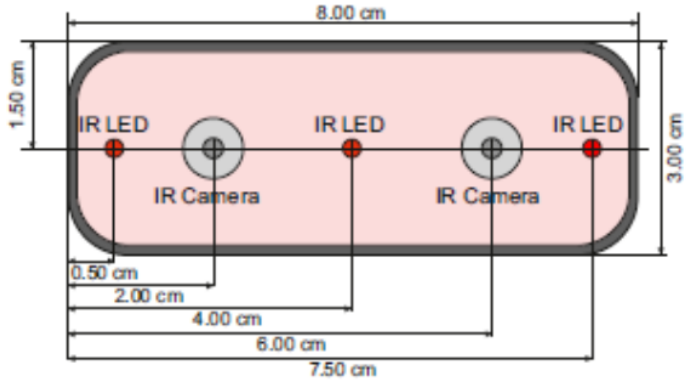

Figura 2: Vista esquemática do controlador *Leap Motion*

Fonte: (BASSILY et al., 2014, p. 79)

de para-brisa e outros comandos do painel (GARBER, 2013). Lojas já podem oferecer uma navegação por catálogo em 3D controlado por gestos ou fazer visitas virtuais a imóveis (BATISTA, 2016). Na saúde, médicos e dentistas podem utilizar do mecanismo para explorar imagens durante uma cirurgia, dentro da sala de operação, sem perder a esterilização, por não precisar tocar no equipamento (ROSA; ELIZONDO, 2014). Para imersão em um ambiente de realidade virtual aumentada, os gestos são a forma mais intuitiva e natural de interação. Na indústria, a tecnologia já é utilizada para controle de robôs e até para projeto e manipulação de modelos de partes de foguetes, com o uso da realidade virtual (LEAP..., 201-?b).

Por meio do kit de desenvolvimento de software (SDK, do inglês *software development kit*), o desenvolvedor consegue receber as informações sobre os gestos feitos pelo usuário, além da posição e orientação das mãos e dedos, assim como a rotação, velocidade e tipo de movimento, a partir de uma estrutura de dados[3] anterior. A biblioteca do *Leap Motion* é escrita em C++, no entanto é possível usar outras linguagens de programação para trabalhar com o SDK, como C#, *Python*, Java, *JavaScript*, entre outras.

O *Leap Motion* adota um sistema de coordenadas cartesianas com a origem centralizada sobre o topo do controlador, como mostra a Figura 3. Os eixos x e z compõem o plano horizontal, com o eixo x paralelo às bordas do dispositivo. O eixo y é vertical, com os valores positivos aumentando de baixo para cima, e o z tem valores positivos aumentando na direção do usuário.

Para a programação *offline* desenvolvida sobre o *RoboDK* neste trabalho, o *Leap Motion* será a interface entre o usuário e o simulador por meio do reconhecimentos de gestos.

[3] A estrutura de dados também é conhecida como *frame*.

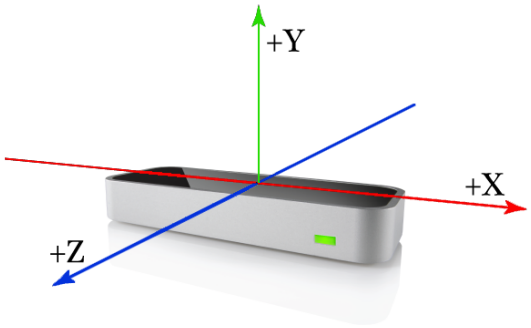

Figura 3: Coordenadas Cartesianas do Leap Motion

Fonte: (LEAPMOTION, 201-?a)

2.4 Estado da arte

No contexto do uso de gestos para controle de manipuladores robóticos, vários estudos têm sido realizados nos últimos anos usando diferentes técnicas.

Landgraf et al. (2016) faz o controle de um robô com o uso de sensores baseados em elastômeros dielétricos, que são colocados em uma peça de tecido e vestidos sobre o braço do usuário. Os sensores utilizados são flexíveis e podem ser esticados, variando a capacitância e permitindo a identificação de um movimento. Com esta abordagem, Landgraf et al. (2016) conseguiu, neste trabalho, o reconhecimento de 3 graus de liberdade.

Para fazer o reconhecimento de gestos de corpo inteiro, Yang et al. (2007) usa as câmeras do *T-Rot*, um robô pessoal direcionado ao suporte de pessoas mais velhas. Yang et al. (2007) descreve um conjunto de características, codificando uma série de relações angulares entre partes do corpo e, então, faz o mapeamento de um vetor de características para identificação correta dos gestos. Com isto, é possível identificar uma pessoa andando, correndo, sentando em uma cadeira e até mesmo sentando ou deitando no chão. Isto pode resultar na identificação de uma situação de perigo e fazer com que o robô peça ajuda, entrando em contato com alguém por telefone, por exemplo. Já Angulo (2015) usou o *Microsoft Kinect*, para fazer a leitura das características corporais e a identificação dos gestos, e o robô humanoide NAO, para responder aos movimentos identificados.

Com a implementação do algoritmo de reconhecimento de imagens da mão obtidas com uma câmera comum, Grzejszczak e Adrian (2015) destaca a ocorrência de pontos de singularidade, que podem existir dependendo do algoritmo usado para a resolução da cinemática inversa. Para contornar este problema, Grzejszczak e Adrian (2015) propõe um conceito misto. Nele, o usuário opera usando um sistema de coordenadas base, mais simples e intuitivo, mas que necessita a resolução da cinemática inversa, e o robô opera no espaço das juntas, o que evita o problema das singularidades.

Yu et al. (2015) utilizou do *Leap Motion* para fazer o reconhecimento de gestos

das duas mãos para controlar o robô NAO. Neste trabalho, o usuário usa a mão direita para controlar os movimentos de navegação do robô e a mão esquerda para operação do braço esquerdo do robô. Com isso, é possível fazer, de forma simples, o robô ir até um local, pegar um objeto e levá-lo para outro local, por exemplo.

3 Desenvolvimento

Foi realizado um experimento de programação utilizando o simulador *RoboDK* e incorporando o *Leap Motion* para fazer a identificação de gestos e o controle do manipulador em tempo real no *software*. A partir deste experimento, foram coletados dados para análise da viabilidade do uso deste controlador para o controle de manipuladores.

O robô manipulador escolhido para este trabalho foi o KUKA KR 6 R900 sixx, mostrado na Figura 4. Este é um robô industrial com 6 graus de liberdade e raio de alcance máximo de 901mm, usando uma garra Robotiq de 2 dedos como ferramenta.

Figura 4: Foto do Robô KUKA KR 6 R900 sixx usado em simulador para os experimentos

Fonte: (KUKA, 201-?)

3.1 O simulador *RoboDK*

Para este trabalho, o simulador gratuito *RoboDK* foi utilizado. Um dos pontos observados para a sua escolha foi a possibilidade de realizar a programação usando a linguagem *Python*, que já é amplamente aceita e difundida. Esta característica faz com que a aprendizagem da linguagem específica do robô não seja um requisito e torna o desenvolvimento mais ágil. Este simulador possui, também, uma grande variedade de modelos de robôs disponíveis em sua biblioteca, assim como objetos e ferramentas para compor o ambiente de trabalho. Convém destacar que a possibilidade do uso da linguagem *Python* é de grande importância neste trabalho, pois facilita a integração entre o simulador e o controlador *Leap Motion*, que oferece suporte a esta mesma linguagem.

A interface é simples e de fácil aprendizagem. O ambiente utilizado para as simulações deste trabalho conta com o robô, posicionado sobre uma mesa, assim como uma bola e uma caixa, como mostrado na Figura 5. Uma alternativa à programação em *Python* é o uso de comandos próprios do *RoboDK*, que permitem marcar pontos no espaço, novos sistemas cartesianos de referência, mover os objetos e enviar comandos ao robô. Uma vez que os comandos são colocados em ordem na forma de uma rotina, este método se mostra simples e permite pequenos testes rápidos sem o uso de programação textual. Contudo, esta forma oferece uma menor dinamicidade e, à medida que as tarefas se tornam mais complexas, se torna menos conveniente.

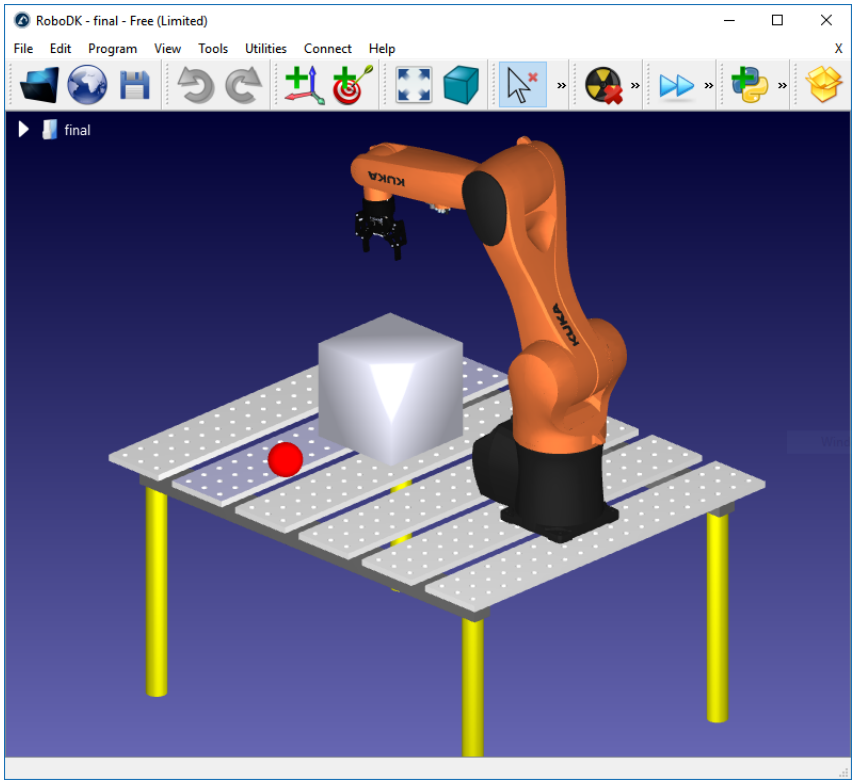

Figura 5: Ambiente de trabalho utilizado no *RoboDK* para os experimentos

O código em *Python* é adicionado dentro da interface do *RoboDK* da mesma forma que outros comandos são adicionados e precisa apenas da importação de duas bibliotecas para funcionar corretamente. Exemplos estão disponíveis na pasta criada durante a instalação e no site do simulador[1] e são suficientes para o início da programação.

3.2 O controlador *Leap Motion*

Para facilitar o interfaceamento entre o usuário e o robô, foi utilizado o *Leap Motion*, um controlador compacto, focado na detecção das mãos. Com baixo custo, é uma boa opção para desenvolvimento de projetos com base em gestos ou Realidade Virtual,

[1] http://www.robodk.com/

pois ele faz a identificação das mãos, das juntas dos dedos e de gestos, fornecendo dados já processados relativos ao que foi detectado. Para ser utilizado, é necessário instalar os *drivers* e o programa principal, onde é possível baixar alguns dos programas disponíveis, como controlador do computador por gestos, jogos, entre outros.

Um dos aplicativos padrões é o *Leap Motion Visualizer*. Nele é possível ver como o dispositivo reconhece as mãos, áreas de sensibilidade e alguns dados coletados por ele. A Figura 6 mostra um exemplo da tela do *Visualizer* com duas mãos e algumas informações básicas sobre elas.

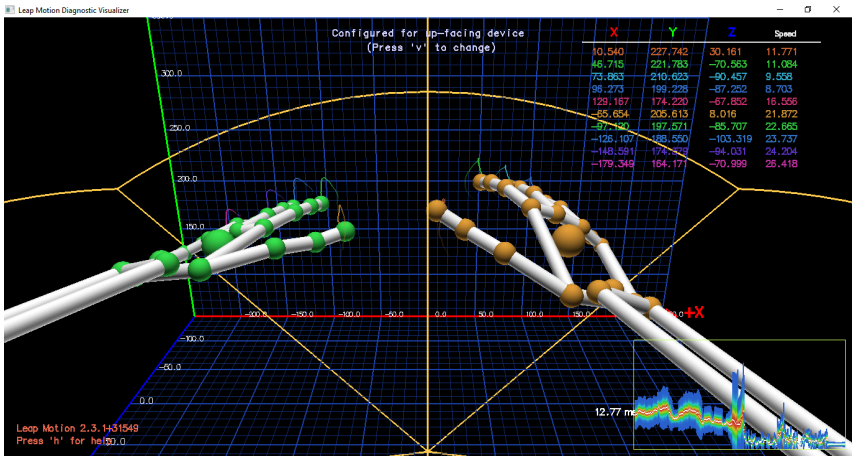

Figura 6: Tela do *Leap Motion Visualizer* com duas mãos detectadas e com as coordenadas e velocidades das pontas dos dedos

Por ter o código fechado, não é possível saber como é feito o tratamento das imagens pelo *software* do *Leap Motion*, porém todos os dados coletados estão disponíveis para o desenvolvedor através do *SDK*, bastando importar suas DLLs (*Dynamic-link library*, ou bibliotecas de vínculo dinâmico).

3.3 O Código

O desenvolvimento consiste, basicamente, na programação do código em *Python* para fornecer a comunicação entre o controlador e o robô, de forma que ambos funcionem juntos em uma relação amigável para o usuário entre os movimentos das mãos e a resposta do manipulador. Esta programação foi dividia em 3 programas. O primeiro faz a comunicação com o controlador e pega os dados coletados. O segundo programa é o responsável por estabelecer a relação entre os dados coletados e o robô presente no simulador e fazê-lo movimentar como uma "sombra" aos movimentos do usuário. Neste programa também são identificados e salvos, em um arquivo, gestos que indicam as ações de pegar ou largar um objeto presente no ambiente simulado, assim como os pontos do caminho percorrido. O último programa lê um arquivo contendo uma série de ações e

posições (com as orientações) e faz o robô executá-las. Este é o equivalente à programação de uma tarefa pré-definida para execução no robô.

Para permitir uma análise dos resultados, foi definido um caminho padrão para o robô repetir. Começando de uma posição inicial fixa, o manipulador deveria ir até uma bola posicionada sobre uma mesa, sobre a qual o robô também estava colocado, e pegá-la. Depois, deveria colocar a bola sobre a caixa que estava na mesa, retornar à posição inicial, pegar a bola novamente e devolvê-la à sua posição original. Esta sequência de posições é exibida na Figura 7.

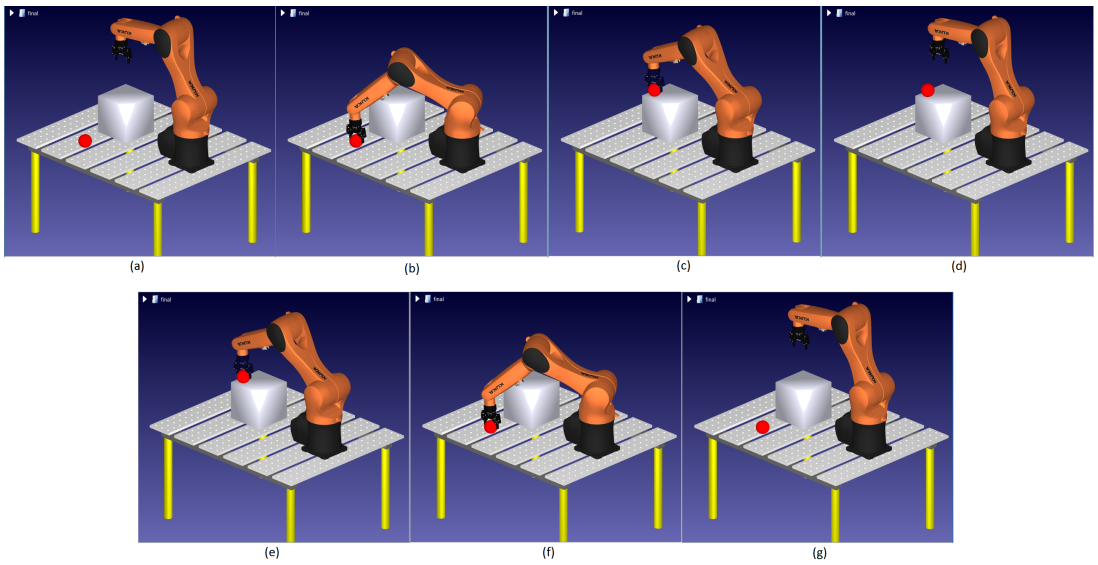

Figura 7: Sequência de posições chave que o robô deve percorrer para completar o caminho definido

3.3.1 Bibliotecas utilizadas

Em computação, uma biblioteca é uma coleção de subprogramas que podem ser utilizados por outros programas para executar tarefas e fornecer dados. O uso deste conceito traz consigo a modularização do código, que permite a reutilização de partes do código em outros programas sem a necessidade de refazer e recompilar todas as funções do novo programa.

Para trabalhar com o *Leap Motion* usando *Python*, é preciso importar a biblioteca *"Leap"*, que irá fornecer acesso a todos os dados que o controlador disponibiliza, como quantidade de mãos, dedos, ferramentas e gestos identificados e suas respectivas posições e orientações. A biblioteca precisa estar em um local acessível ao programa que, neste caso, fica dentro do *RoboDK* e é colocado em uma pasta temporária do *Windows* quando vai ser executado. Para facilitar este acesso, a biblioteca foi colocar dentro de uma pasta em *"C:"*.

A API do *RoboDK* é dividia em dois módulos, ou bibliotecas, que precisam ser importados no código para que suas funções possam ser utilizadas, que são o *"robolink"* e

o *"robodk"*. Eles permitem a comunicação entre o programa em *Python* e o simulador e oferecem as ferramentas para trabalhar com os dados transmitidos nesta comunicação.

O fluxograma da Figura 8 mostra, de forma resumida, como foi feito o uso das bibliotecas e o que é feito em cada um dos programas.

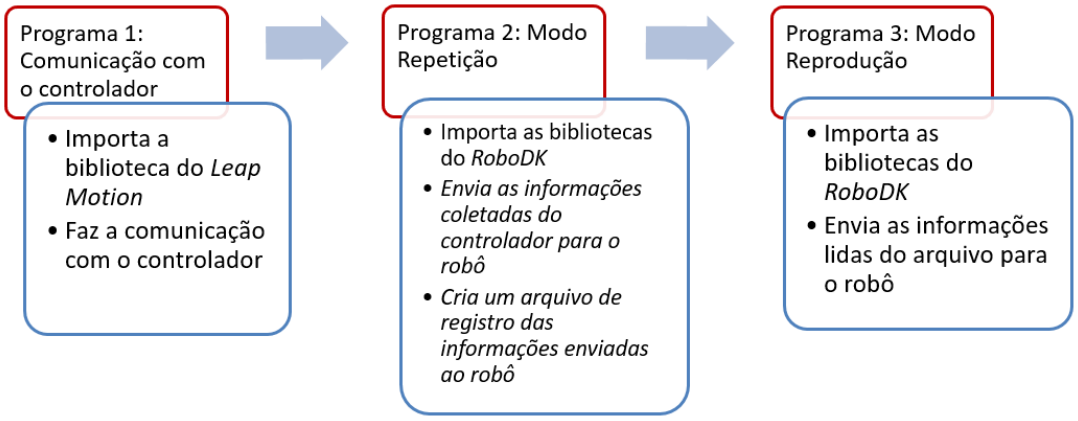

Figura 8: Fluxo simplificado das atribuições de cada programa, quais bibliotecas são importados e a relação entre os programas

3.3.2 Comunicação com o Controlador

Os movimentos das mãos, captados pelas câmeras do *Leap Motion*, são "traduzidos" para coordenadas pelo software fornecido e o primeiro programa, dedicado a coletar os dados fornecido pelo *Leap Motion*, pega a informação de cada estrutura de dados enviada e faz a seleção do que é importante para este trabalho. O algoritmo completo é mostrado no Algoritmo 1, onde a estrutura de dados é indicado por *frame*.

Para indicar que o robô deve começar a seguir os movimentos da mão, um sinal deve ser passado, caso contrário, posições e movimentos descoordenados seriam passados ao robô. Dito isto, usamos duas mãos para movimentá-lo, sendo uma para sinalizar o início do movimento, por meio de um punho fechado, e a outra para realizar os movimentos. Um exemplo é mostrado na Figura 9, em que o punho é feito na mão esquerda e o robô segue os movimentos da mão direita.

Um dos dados passados pelo controlador é o parâmetro "*punho*", que pode estar *aberto* ou *fechado*, sendo *fechado*, um punho completamente fechado e *aberto*, uma mão estendida. Além do sinal para início dos movimentos, um punho fechado indica também a ação de pegar ou soltar um objeto. O movimento de fechar a mão para pegar um objeto já é natural e não necessita explicação. Para uma melhor identificação dos movimentos das mãos, porém, a mão deve estar estendida. Fazer movimentos complexos com a mão fechada traz resultados com grandes variações e ruídos, por isso foi definido que para pegar um objeto é feito um sinal, no entanto este sinal não precisa continuar sendo feito durante a movimentação, permitindo que a operação seja feita com a mão aberta. Um novo sinal

Algoritmo 1 Coleta das informações das mãos

```
1: Recebe as informações do frame
2: para cada mão em frameAtual faça
3:     se punho = fechado então
4:         se punhoFechado = Falso então
5:             punhoMão ← mão.tipo                ▷ Se é a mão esquerda ou direita
6:             punhoFechado ← Verdadeiro
7:             estáMovendo ← Verdadeiro
8:             frameInicial ← frameAtual
9:         senão
10:            se punhoMão ≠ mão.tipo então
11:                se estavaSegurando = Falso então
12:                    segurar ← Verdadeiro
13:                    mãoSegurando ← mão.tipo
14:                senão
15:                    segurar ← Falso
16:                fim se
17:            fim se
18:        fim se
19:    senão se punhoFechado = Verdadeiro e punhoMão = mão.tipo e punho = aberto então
20:        punhoFechado ← Falso
21:        estáMovendo ← Falso
22:    senão se punhoFechado = Verdadeiro e mãoSegurando = mão.tipo e punho = aberto então
23:        se estavaSegurando = Falso então
24:            estavaSegurando ← Verdadeiro
25:        senão
26:            estavaSegurando ← Falso
27:        fim se
28:    fim se
29:    se punhoFechado = Verdadeiro e punhoMão ≠ mão.tipo então
30:        movimentoLinear ← mão.translação
31:        rotação ← mão.rotação
32:    fim se
33: fim para
```

deve ser enviado para indicar a ação de soltar o objeto, sendo este sinal também definido como o fechamento de um punho, cabendo ao programa identificar o significado do punho em cada momento.

Desta forma, o primeiro dado a ser tratado é a formação do punho. Quando um *punho* é detectado, caso ele seja o primeiro, ou seja, o sinalizador *punhoFechado* não está marcado, o *frame* atual é definido como o *frame* inicial, que será usado como referência para os movimentos a partir deste momento. Se este *punho* não for o primeiro, ou seja, as duas mãos estiverem fechadas e o sinalizador *punhoFechado* estiver marcado, é indicado que o robô deve pegar ou soltar um objeto. A distinção entre o momento de pegar ou soltar um objeto é feita por meio do sinalizador *estavaSegurando*. Se este estiver marcado, o sinal indica a liberação do objeto. De outra forma, se não estiver marcado, o sinal indica a ação de pegar um objeto.

Quando há detecção de uma mão estendida, o código identifica qual mão é e modifica o sinalizador *estáMovendo* ou *estavaSegurando* de acordo com o momento do

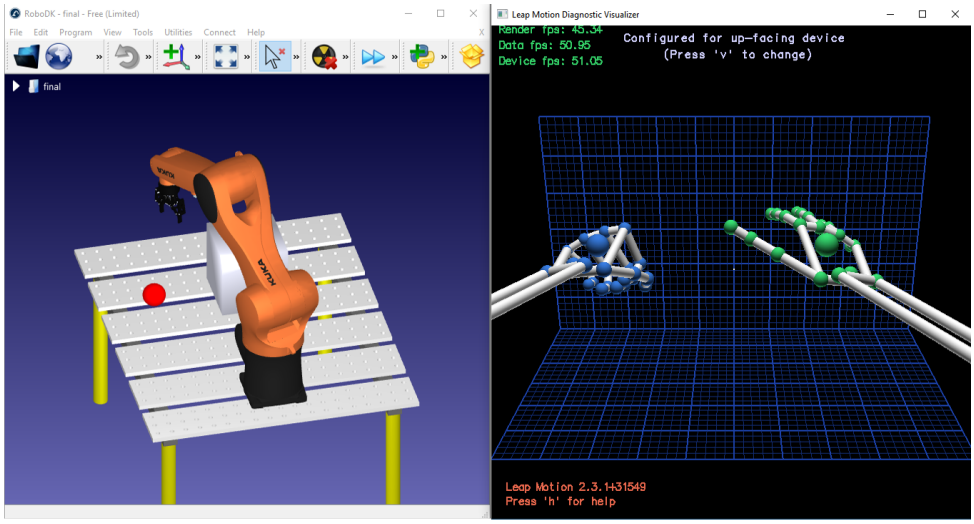

Figura 9: Visualização da tela com o robô sendo movimentado (esquerda) e o esquema das mãos que o movimentam (direita)

movimento.

Na situação de haver uma mão estendida e outra fechada, os dados de movimento da mão estendida são registrados para uso dos programas.

3.3.3 Modo repetição

O código que executa os movimento no robô, mostrado no Algoritmo 2, lê os sinalizadores marcados pelo código de leitura do sensor, envia os sinais ao robô e registra no arquivo. É este código também que faz a correspondência adequada entre as coordenadas usadas como referência pelo controlador do *Leap Motion* e as coordenadas de referência do robô e descarta variações muito pequenas na posição, considerando uma margem em que se considera a mão do usuário como parada.

3.3.4 Modo reprodução

O pseudo-código do terceiro programa pode ser visto no Algoritmo 3. Este programa faz a leitura do arquivo e envia os sinais identificados para o robô. No arquivo, cada linha representa uma ação a ser executado pelo robô, podendo ser a de segurar um objeto, soltá-lo ou realizar um movimento.

Algoritmo 2 Modo repetição dos movimentos

```
1: Move o robô para a posição inicial
2: enquanto Verdadeiro faça                                   ▷ Repete indefinidamente
3:     se Está terminado então Interrompe o loop
4:     fim se
5:     se segurar = Verdadeiro e segurando = Falso então
6:         Fecha a garra da ferramenta
7:         segurando ← Verdadeiro
8:         Registra o comando de fechamento no arquivo
9:     senão se segurar = Falso e segurando = Verdadeiro então
10:        Abre a garra da ferramenta
11:        segurando ← Falso
12:        Registra o comando de abertura no arquivo
13:    fim se
14:    se estáMovendo = Verdadeiro então
15:        Pega a posição e os ângulos do movimento
16:        Faz a correspondência entre as coordenada do Leap Motion e as do robô
17:        Descarta valores muito pequenos
18:        Adiciona o movimento à posição atual
19:        Move o robô para a nova posição
20:        Registra a nova posição no arquivo
21:    fim se
22: fim enquanto
```

Algoritmo 3 Modo de reprodução dos movimentos gravados

```
1: Lê o arquivo de dados gravados
2: Move o robô para a posição inicial
3: para cada linha em arquivo faça
4:     se linha ="SEGURAR" então
5:         Fecha a garra da ferramenta
6:     senão se linha ="SOLTAR" então
7:         Abre a garra da ferramenta
8:     senão
9:         Lê a posição de destino
10:        Move o robô para a nova posição
11:    fim se
12: fim para
```

4 Discussão dos Resultados

No início do experimento, o robô apresentava movimentos muito curtos, impedindo a continuidade e fluidez dos movimentos, o que levou à inclusão de um fator de multiplicação da distância percorrida, que foi ajustado empiricamente. Esta inserção, no entanto, fez com que o robô se movimentasse mesmo quando a mão do usuário aparentava estar parada. Isto se deve à dificuldade de permanecer com a mão completamente parada, sem sustentação, no ar. Diante disso, foi implementada uma margem de variação em que o robô não se move, descartando movimentos muito pequenos. Esta margem também foi ajustada de forma empírica, usando margens diferentes para os movimentos de translação, medidos em milímetro e em linha reta, e para os de rotação, medidos em graus.

Feita a calibração, foram realizados 60 ensaios, em sequência, seguindo o caminho definido na Seção 3.3.

Tanto durante os ensaios quanto posteriormente, utilizando o arquivo de dados salvos, foram observados e registrados 5 pontos:

- Movimentos involuntários - Quantidade de movimentos identificados como não correspondentes aos gestos feitos pelo usuário;
- Bola atravessada - Quantidade de vezes em que a bola foi atravessada pela ferramenta ou “afundada” na mesa ou na caixa;
- Ausência de identificação de punho fechado - Quantidade de vezes em que foi feito um movimento de fechar o punho, mas o gesto não foi detectado;
- Colisão - Se houve colisão ou não durante a execução;
- Mudanças de configuração - Quantidade de vezes em que o braço deu uma volta para mudar de configuração;

A Tabela 1 mostra o número de ensaios que apresentou cada um dos pontos observados, indicando também a que porcentagem do total este número representa. O número de ocorrências de cada evento pode ser visto no Apêndice B, ordenado pelo número do experimento.

Tabela 1: Presença dos problemas identificados nos ensaios

	Nº de ensaios que apresentam	Porcentagem do total
Movimentos involuntários	60	100,0%
Bola atravessada	60	100,0%
Ausência de identificação de punho fechado	55	91,7%
Colisão	26	43,3%
Mudanças de configuração	7	11,7%

Os movimentos involuntários apareceram em todos os ensaios, ainda que, em muitas vezes, tenham sido apenas pequenos movimentos em que a ferramenta voltou imediatamente ao caminho desejado. Mesmo com a atenuação deste efeito por meio dos parâmetros de calibração, estes movimentos são causados pela leitura de movimentos involuntários do usuário quando este fica com a mão no ar sem sustentação.

A bola também foi atravessada em todos os ensaios. No *RoboDK*, a identificação de uma colisão é seletiva, podendo ser escolhidos quais os pares de elementos são sólidos e representam uma colisão, porém o programa em execução é interrompido quando isto ocorre. Dito isto, a interação entre a bola e a ferramenta, assim como entre a bola e a mesa e entre a bola e a caixa, não foram marcadas como colisão e, consequentemente, estes elementos conseguiam atravessar uns aos outros. Durante a execução de cada ensaio, houve a preocupação em evitar objetos sendo atravessados, todavia, devido à representação do sistema em uma tela de computador, não era possível ter uma noção exata de profundidade e este problema persistiu.

A não identificação de punho fechado aconteceu em 91,7% dos ensaios pelo menos uma vez. Esta identificação é feita com base em um dos dados enviados pelo controlador do *Leap Motion*, sendo assim, é possível afirmar que o gesto não foi reconhecido devido a distorções na leitura da mão e dos dedos no processamento da imagem. Por este motivo, 51% dos gestos de punho feitos não foram reconhecidos. No entanto, 45,8% dos gestos não reconhecidos ocorreram em apenas 20% dos ensaios. Se estes fossem desconsiderados, a porcentagem de ensaios em que todos os gestos feitos são reconhecidos subiria para 28,3% e a quantidade de gestos não detectados cairia para 36,2% do total.

Conforme explicado nesta seção, o *RoboDK* permite a seleção de pares de elementos que podem ou não colidir. Neste trabalho, apenas a bola era isenta de colisões, para facilitar sua manipulação. Os demais itens (a garra, o braço do robô, a mesa e a caixa) geravam uma colisão quando entram em contato. Algumas vezes estas colisões aconteceram pelo mesmo motivo de a bola ser atravessada, a deficiência na visualização no simulador.

Outras vezes, movimentos involuntários acabavam por ocasionar uma colisão entre a garra e a mesa ou entre a garra e a caixa. 43% dos ensaios apresentaram uma colisão.

Em 11,7% dos ensaios, houve pelo menos uma volta para mudança de configuração. Uma mudança de configuração significa, por exemplo, o caso em que o robô muda de "cotovelo para baixo" para "cotovelo para cima" de um ponto ao outro no caminho. Esta alteração é definida por meio da cinemática inversa[1], feita pelo próprio *RoboDK*, que escolhe a configuração mais próxima da configuração corrente.

Além destes aspectos, também foram registradas as coordenadas da bola no ponto médio, sobre a caixa, e no ponto final. A distância média entre o ponto colocado e o alvo foi de 40,75mm. Separando os alvos, a distância média até o ponto final foi de 55,43mm, enquanto a distância média até o ponto intermediário foi de 26,08mm. Estas informações podem ser vistas na Tabela 2, sendo o ponto intermediário representado por Alvo 1 e o ponto final, por Alvo 2. A relação completa das coordenadas onde a bola foi deixada e distâncias em relação aos respectivos alvos pode ser encontrada no Apêndice A

Tabela 2: Distância média entre a posição definida e a realizada

	Distância média em relação ao alvo definido (mm)
Alvo 1	26,08
Alvo 2	55,43
Ambos	40,76

A experiência aponta que maiores aprimoramentos precisam ser feitos para que esta metodologia seja usada para aplicações que necessitam maior precisão e possuam maior sensibilidade a erros. O reconhecimento de gestos é uma tarefa complexa, que é dificultada quando há objetos se sobrepondo e pode ser bastante prejudicada pela rapidez dos movimentos da pessoa, o que pode ser observado pela alta taxa de gestos não reconhecidos no experimento. Outro aspecto que influencia na viabilidade da metodologia é a visualização dos movimentos e das posições assumidas pelo robô. No caso deste experimento, a visualização feita na tela de um computador foi prejudicial, gerando uma distorção entre a posição observada e a posição real do robô. A exibição em uma tela de computador permite apenas uma vista em duas dimensões, simulando a profundidade. O efeito é suficiente para gerar uma sensação de imagem tridimensional, porém não evita o ocasionamento da distorção relatada. Esta distorção pode ser observada claramente quando se pega a bola com a garra e muda o ângulo de observação do usuário, caso mostrado na

[1] A cinemática inversa consiste em determinar os ângulos das juntas a partir da posição da ferramenta.

Figura 10. Com este procedimento, diversas vezes era constatado que a bola não estava na posição que aparentava estar quando observada pelo ângulo inicial.

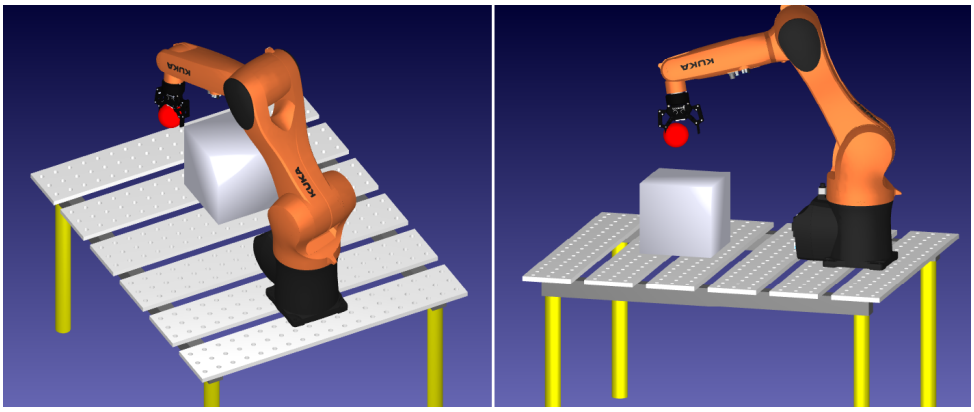

Figura 10: Vista inicial, com a bola aparentemente bem encaixada na garra, (esquerda) e vista lateral da mesma configuração, mostrando a diferença na posição da bola (direita)

5 Conclusões

Este trabalho apresentou uma revisão bibliográfica sobre a programação de robôs, sua simulação em *software*, sobre sensores de gestos e relatou a experiência do uso do *Leap Motion* para controle de um robô no ambiente simulado do *RoboDK*.

Uma visão geral sobre o simulador e o controlador utilizados são apresentados, assim como o desenvolvimento da programação realizada no experimento. A experiência gerou resultados satisfatórios, com o robô seguindo corretamente os comandos e movimentos feitos pelas mãos. Os dados coletados mostram, contudo, que ainda há espaço para melhorias. A implementação de parâmetros para ajuste da comunicação, mostrados no Capítulo 4, e sua calibração, feita empiricamente, melhorou o desempenho do robô, obtendo agilidade, responsividade e fluidez.

Para a avaliação dos resultados, os pontos observados no Capítulo 4 foram utilizados. A alta taxa de eventos negativos, como movimentos involuntários e não identificação dos gestos, nos leva a acreditar que esta solução ainda precisa de melhorias, principalmente para aplicações de alto risco ou precisão. A presença em mais de 90% dos ensaios de movimentos involuntários, não-identificação de punho fechado e bola atravessada é preocupante no sentido de viabilidade do método. Cabe ressaltar que grande parte dos movimentos involuntários foram gerados durante o fechamento da mão para fazer um punho ou no momento de sua abertura, de forma que o próprio gesto acabava por ocasionar problemas. Com a não identificação de punho ocorrendo em muitas ocasiões, o número de movimentos involuntários também ficou acima do que poderia ter sido. As mudanças de configuração do braço, presentes em menos de 12% dos ensaios pode ser mitigada fazendo a cinemática inversa manualmente, sem aproveitar da API do simulador. Já as colisões, da mesma forma que as bolas atravessadas, podem ser evitadas se uma melhor visualização puder ser feita, por meio de realidade virtual ou do uso de um manipulador real, que permitem a percepção da profundidade, levando a uma melhor precisão no posicionamento, o que espera-se que reduza em grande parte os problemas identificados neste experimento.

Diante do exposto, o experimento nos leva a crer que a metodologia utilizada, com a implementação de maiores aprimoramentos, pode ser aplicada na programação de manipuladores, assim seu estudo merece ser aprofundado, pois mostra um grande potencial e trará muitas vantagens.

5.1 Sugestões para Trabalhos Futuros

De acordo com o experimento realizado, podemos notar a necessidade de novos experimentos para tratar das dificuldades encontradas. A mesma metodologia, utilizada com a adição do uso de realidade virtual ou um robô real, pode chegar a reduzir consideravelmente os problemas, já que 2 dos 5 pontos observados no Capítulo 4 se devem à visualização dos movimentos do manipulador. Outro ponto de grande efeito nos resultados do trabalho foi a identificação faltosa de movimentos. Grande parte desse problema pode ser evitado com o uso de reconhecimento de voz, de forma que as mãos sejam utilizadas para realizar os movimentos do robô e a voz seja utilizada para dar comandos, como de início e fim de percurso. Uma sugestão para trabalhos futuros é uma abordagem completa, que englobe os gestos, a realidade virtual e o reconhecimento de voz.

Referências

ANGULO, C. Gesture based Human Multi-Robot Interaction. 2015. 2015. Citado na página 20.

BASSILY, D.; GEORGOULAS, C.; GÜTTLER, J.; LINNER, T.; BOCK, T.; MÜNCHEN, T. U. Intuitive and Adaptive Robotic Arm Manipulation using the Leap Motion Controller Gesture-based Human-Robot In- teraction Proposed Concept Supporting elderly and disabled people. 2014. p. 78–84, 2014. Citado na página 19.

BATISTA, R. M. d. P. Navigating Virtual Reality Worlds with Leap Motion Controller. 2016. 2016. Citado na página 19.

BERNARD, R.; DILLMAN, R.; HORMANN, K.; TIERNEY, K. **Integration of Robots into CIM**. [S.l.]: Springer Science & Business Media, 2012. 349 p. Citado na página 14.

GARBER, L. Moving Interfaces in a New Direction. 2013. p. 22–25, 2013. Citado na página 19.

GRZEJSZCZAK, T.; ADRIAN, Ł. Robot manipulator teaching techniques with use of hand gestures. 2015. 2015. Citado na página 20.

HAGIS, C. History of robots. 2003. n. May, 2003. Citado 2 vezes nas páginas 10 e 13.

HAYDEN, E.; ASSANTE, M.; CONWAY, T. An Abbreviated History of Automation and Industrial Controls Systems and Cybersecurity. **Sans**, 2015. n. August, p. 32, 2015. Disponível em: <https://ics.sans.org/media/An-Abbreviated-History-of-Automation-and-ICS-Cybersecurity.pdf>. Citado na página 10.

KUKA. **KR 6 R900 sixx (KR AGILUS)**. 201–? Acesso em: 04 de jan. 2017. Disponível em: <http://www.kuka-robotics.com/br/products/industrial_robots/small_robots-/kr6_r900_sixx/start.htm>. Citado na página 22.

LANDGRAF, M.; REITELSH, S.; HOFMANN, V.; ZIMBER, F.; EITH, F.; WIELAND, B.; YOO, I. S. Intuitive Gesture Control of Robots with a Sensor Arm Sleeve Based on Dielectric Elastomer Sensors. 2016. p. 114–119, 2016. Citado na página 20.

LEAPMOTION. **Coordinate Systems**. 201–? Acesso em: 04 de jan. 2017. Disponível em: <https://developer.leapmotion.com/documentation/csharp/devguide-/Leap_Coordinate_Mapping.html>. Citado na página 20.

LEAPMOTION. **Leap Motion - Solutions**. 201–? Acesso em: 04 de jan. 2017. Disponível em: <https://www.leapmotion.com/solutions>. Citado na página 19.

LORUSSI, F.; SCILINGO, E. P.; TESCONI, M.; TOGNETTI, A.; ROSSI, D. D. Strain Sensing Fabric for Hand Posture and Gesture Monitoring. 2005. v. 9, n. 3, p. 372–381, 2005. Citado na página 18.

MADALENO, P. D. T. Simulação e Programação Offline de Robôs Industriais Afectos a Tarefas de Lixagem. 2011. 2011. Citado 3 vezes nas páginas 14, 16 e 17.

MELO, A. R. L. L. de. **Um estudo sobre o mapeamento de gestos do Leap Motion para a Língua Brasileira de Sinais (Libras)**. Tese (Doutorado) — Universidade Federal de Pernambuco, 2015. Citado na página 18.

NUBIOLA, A. **Off-line Programming**. 2016. Acesso em: 03 de jan. 2017. Disponível em: <https://robodk.com/blog/>. Citado na página 17.

PAN, Z.; POLDEN, J.; LARKIN, N.; Van Duin, S.; NORRISH, J. Recent progress on programming methods for industrial robots. **Robotics and Computer-Integrated Manufacturing**, 2012. Elsevier, v. 28, n. 2, p. 87–94, 2012. ISSN 07365845. Disponível em: <http://dx.doi.org/10.1016/j.rcim.2011.08.004>. Citado 2 vezes nas páginas 15 e 16.

ROSA, G. M.; ELIZONDO, M. L. Use of a gesture user interface as a touchless image navigation system in dental surgery : Case series report. 2014. 2014. Citado na página 19.

SANTOS, J. B. dos. A Robótica Como Ferramenta no Ensino da Física. 2014. 2014. Citado na página 13.

SILVA, M. F. S. **Simulação e Programação Off-line de Robôs de Montagem**. 206 p. Tese (Doutorado) — Universidade do Porto, Porto, 1996. Citado 2 vezes nas páginas 10 e 15.

STURMAN, D. J.; ZELTZER, D. A Survey of Glove-based Input. 1994. v. 161941010, p. 30–39, 1994. Citado na página 18.

WEICHERT, F.; BACHMANN, D.; RUDAK, B.; FISSELER, D. Analysis of the Accuracy and Robustness of the Leap Motion Controller. **Sensors**, 2013. v. 13, n. 5, p. 6380–6393, 2013. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/13/5/6380/>. Citado na página 18.

WITTENBERG, G. Developments in offline programming: an overview. **Industrial Robot: An International Journal**, 1995. v. 22, n. 3, p. 21–23, 1995. ISSN 0143-991X. Disponível em: <http://dx.doi.org/10.1108/EUM0000000004186>. Citado 2 vezes nas páginas 15 e 17.

YANG, H.-d.; PARK, A.-y.; LEE, S.-w.; MEMBER, S. Gesture Spotting and Recognition for Human – Robot Interaction. 2007. v. 23, n. 2, p. 256–270, 2007. Citado na página 20.

YU, N.; XU, C.; WANG, K.; YANG, Z.; LIU, J. Gesture-Based Telemanipulation of A Humanoid Robot for Home Service Tasks. 2015. p. 1–5, 2015. Citado na página 20.

Apêndices

APÊNDICE A – Posições da bola e distância do alvo

A Tabela 3 mostra as coordenadas em que a bola foi colocada, assim como a distância em relação à posição pré-definida, sendo a posição intermediária representada por Alvo 1 e a posição final, por Alvo 2. O experimento 0 indica as posições definidas como destino para cada um dos alvos. Todas as distâncias são dadas em milímetros.

Tabela 3: Coordenadas em que a bola foi colocada e distâncias, em milímetros, em relação à posição pré-definida

Experimento	Alvo	x	y	z	Distância
0	1	-644,103	-26,544	280,447	
	2	-603,887	-332,404	36,819	
1	1	-634,978	-11,852	277,941	17,476
	2	-499,925	-395,751	114,861	144,608
2	1	-640,794	-20,105	272,282	10,912
	2	-569,184	-346,535	67,53	48,447
3	1	-655,269	0,64	276,661	29,631
	2	-510,58	-339,44	106,244	116,514
4	1	-648,882	-13,008	269,499	18,053
	2	-608,547	-311,693	47,768	23,886
5	1	-657,469	-13,787	265,677	23,655
	2	-586,827	-335,238	25,593	20,618
6	1	-668,014	-20,795	266,322	28,360
	2	-569,358	-320,188	50,853	39,223
7	1	-640,556	4,965	295,817	35,237
	2	-567,281	-337,216	60,14	43,669
8	1	-660,646	-25,441	290,182	19,226
	2	-607,035	-332,383	31,917	5,826
9	1	-638,332	-5,849	289,372	23,265
	2	-589,841	-339,341	52,599	22,236
10	1	-631,323	-13,953	280,72	17,943
	2	-618,058	-325,082	32,026	16,655

Continuação na próxima página

Tabela 3 – continuação da página anterior

Experimento	Alvo	x	y	z	Distância
11	1	-633,516	-23,858	291,05	15,222
	2	-552,287	-330,537	66,597	59,605
12	1	-619,073	-18,366	288,817	27,630
	2	-550,644	-310,651	62,638	63,045
13	1	-620,773	-26,074	268,219	26,345
	2	-548,366	-341,894	77,531	69,499
14	1	-614,597	-38,081	293,327	34,199
	2	-586,982	-327,945	16,246	26,998
15	1	-653,303	-20,321	267,233	17,262
	2	-535,911	-324,95	76,399	79,012
16	1	-622,421	-18,124	278,734	24,604
	2	-597,84	-333,235	34,754	6,444
17	1	-657,503	-13,018	264,864	20,006
	2	-577,678	-329,578	44,879	27,566
18	1	-643,629	-6,78	283,516	20,006
	2	-584,352	-322,871	27,499	23,651
19	1	-644,009	-14,033	274,18	13,993
	2	-575,884	-327,456	54,58	33,528
20	1	-638,509	23,056	275,559	50,153
	2	-618,104	-282,41	42,649	52,302
21	1	-648,972	-19,907	275,069	9,833
	2	-498,173	-343,863	104,141	125,853
22	1	-626,862	-16,7	269,753	22,550
	2	-481,236	-359,848	113,517	147,238
23	1	-635,726	-13,345	275,799	16,309
	2	-538,734	-334,693	96,366	88,295
24	1	-606,214	-28,909	296,147	41,081
	2	-572,512	-339,959	65,808	43,380
25	1	-607,344	-18,057	274,681	38,164
	2	-620,612	-321,003	57,363	28,840
26	1	-619,328	-21,534	296,571	29,981
	2	-513,243	-343,11	72,063	97,842
27	1	-692,175	-3,598	261,226	56,629
	2	-596,27	-333,35	48,114	13,656
28	1	-641,105	-13,704	279,74	13,204
	2	-591,599	-341,517	40,782	15,803

Continuação na próxima página

Tabela 3 – continuação da página anterior

Experimento	Alvo	x	y	z	Distância
29	1	-663,883	-13,615	276,935	23,890
	2	-504,62	-356,84	106,442	123,687
30	1	-642,868	-33,073	286,879	9,248
	2	-552,348	-325,862	58,847	56,430
31	1	-534,513	-28,307	343,667	126,530
	2	-560,983	-327,49	41,24	43,410
32	1	-626,249	-15,393	295,947	26,141
	2	-564,42	-331,982	55,637	43,726
33	1	-651,376	-19,346	289,265	13,508
	2	-573,5	-324,104	49,707	34,035
34	1	-628,966	-19,52	295,331	22,361
	2	-528,797	-337,025	61,857	79,289
35	1	-607,938	-16,861	284,427	37,650
	2	-583,68	-340,015	-4,719	46,815
36	1	-634,058	-12,325	271,065	19,776
	2	-507,997	-350,81	120,245	128,427
37	1	-645,15	-12,854	255,914	28,114
	2	-563,392	-333,025	51,28	43,004
38	1	-635,865	-15,783	264,272	21,102
	2	-543,881	-324,846	61,837	65,450
39	1	-619,404	-24,099	287,154	25,710
	2	-566,479	-318,98	52,825	42,846
40	1	-659,809	-1,73	266,071	32,697
	2	-517,139	-357,996	82,155	101,171
41	1	-650,546	-18,061	283,597	11,108
	2	-548,407	-345,097	69,893	65,826
42	1	-647,854	-20,292	275,049	9,072
	2	-595,459	-326,601	38,656	10,396
43	1	-643,947	-18,278	267,738	15,161
	2	-574,504	-345,928	46,26	33,696
44	1	-650,909	-17,365	268,519	16,518
	2	-584,624	-327,63	51,992	24,981
45	1	-663,166	-16,469	274,412	22,390
	2	-557,613	-329,62	59,707	51,700
46	1	-638,213	-11,597	270,363	18,968
	2	-565,834	-331,169	44,722	38,885

Continuação na próxima página

Tabela 3 – continuação da página anterior

Experimento	Alvo	x	y	z	Distância
47	1	-635,985	-25,807	284,837	9,258
	2	-560,309	-347,358	72,343	58,177
48	1	-675,705	-15,833	243,154	50,042
	2	-600,022	-321,056	31,142	13,264
49	1	-646,922	-14,115	270,688	16,052
	2	-526,606	-328,67	65,052	82,361
50	1	-661,369	-16,204	290,556	22,522
	2	-555,676	-336,855	49,234	49,982
51	1	-657,649	-12,443	256,858	30,639
	2	-614,624	-325,909	15,636	24,621
52	1	-662,089	-9,638	244,756	43,395
	2	-593,156	-322,789	25,832	18,120
53	1	-653,668	-12,66	277,792	17,068
	2	-563,283	-337,049	63,936	49,047
54	1	-583,471	-23,379	304,155	65,179
	2	-561,802	-316,98	52,679	47,546
55	1	-621,556	-22,873	273,824	23,785
	2	-508,514	-342,157	105,061	117,678
56	1	-624,797	-24,981	280,105	19,372
	2	-530,85	-317,212	69,327	81,375
57	1	-642,918	-8,122	265,507	23,748
	2	-577,493	-341,557	50,32	31,027
58	1	-641,267	-10,804	278,934	16,065
	2	-552,083	-304,722	43,565	59,122
59	1	-615,528	-21,164	295,646	32,810
	2	-513,555	-340,375	73,569	97,847
60	1	-642,209	-12,892	281,548	13,827
	2	-549,506	-353,438	88,393	77,843

APÊNDICE B – Dados dos Experimentos

Durante os experimentos, os dados coletados foram registrados e estão apresentados na Tabela 4, em que cada coluna apresenta a quantidade de vezes em que o evento ocorreu no experimento.

Tabela 4: Número de ocorrências de cada evento nos experimentos

Experimento	Movimentos Involuntários	Ausência de Identificação de Punho Fechado	Bola Atravessada	Colisão
1	9	0	4	0
2	6	2	3	0
3	6	0	4	1
4	8	12	5	1
5	24	2	6	0
6	10	4	3	0
7	6	3	4	1
8	6	3	2	1
9	24	12	4	0
10	13	4	4	0
11	8	4	5	0
12	16	5	5	0
13	7	4	2	1
14	12	4	5	0
15	9	3	3	1
16	6	5	5	0
17	11	6	8	0
18	5	1	4	0
19	12	8	5	1
20	8	3	7	0
21	12	4	4	0
22	10	2	2	0
23	6	3	3	0

Continuação na próxima página

Tabela 4 – continuação da página anterior

Experimento	Movimentos Involuntários	Ausência de Identificação de Punho Fechado	Bola Atravessada	Colisão
24	6	2	2	1
25	24	13	8	1
26	11	4	4	0
27	11	6	4	0
28	4	2	3	1
29	7	4	5	1
30	8	4	3	0
31	5	0	2	1
32	13	6	3	1
33	5	2	2	1
34	9	3	2	1
35	11	5	6	1
36	22	6	8	1
37	5	3	3	1
38	8	2	4	0
39	5	3	4	1
40	10	6	3	0
41	6	1	3	1
42	17	12	7	0
43	5	0	3	0
44	12	5	6	0
45	9	3	4	0
46	7	2	2	1
47	6	4	3	1
48	6	0	3	0
49	12	2	5	0
50	7	1	3	1
51	8	1	5	0
52	15	3	8	0
53	10	6	5	1
54	18	12	12	0
55	27	14	10	0
56	11	4	5	1
57	4	1	3	0
58	14	7	4	1

Continuação na próxima página

Tabela 4 – continuação da página anterior

Experimento	Movimentos Involuntários	Ausência de Identificação de Punho Fechado	Bola Atravessada	Colisão
59	5	1	4	0
60	7	7	3	0