Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
Programa de Pós-graduação em Engenharia de Software
Mestrado Profissional em Engenharia de Software

TeamBridge: Middleware para adaptação de games e controles de reabilitação motora

Alan Klinger Sousa Alves

Natal-RN
Abril 2018

Alan Klinger Sousa Alves

TeamBridge: Middleware para adaptação de games e controles de reabilitação motora

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Software da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Engenharia de Software.

Universidade Federal do Rio Grande do Norte – UFRN

Instituto Metrópole Digital – IMD

Programa de Pós-Graduação em Engenharia de Software

Orientador: Dr. Rummenigge Rudson Dantas

Natal-RN

Abril 2018

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Alves, Alan Klinger Sousa.
TeamBridge: Middleware para adaptação de games e controles de reabilitação motora / Alan Klinger Sousa Alves. - 2018.
113 f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte, Instituto Metrópole Digital, Programa de Pós-Graduação em Engenharia de Software. Natal, RN, 2018.
Orientador: Prof. Dr. Rummenigge Rudson Dantas.

1. Gameterapia - Dissertação. 2. Adaptador de dispositivos - Dissertação. 3. Comunicação entre dispositivos - Dissertação. 4. Exergaming jogos sérios - Dissertação. I. Dantas, Rummenigge Rudson. II. Título.

RN/UF/BCZM CDU 004:615.8

Elaborado por Ana Cristina Cavalcanti Tinôco - CRB-15/262

Alan Klinger Sousa Alves

TeamBridge: Middleware para adaptação de games e controles de reabilitação motora

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Software da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Engenharia de Software.

Trabalho aprovado. Natal-RN, 10 de abril de 2018:

Dr. Rummenigge Rudson Dantas
Orientador
Universidade Federal do Rio Grande do Norte

Dr. Aquiles M. Filgueira Burlamaqui
Examinador interno
Universidade Federal do Rio Grande do Norte

Dra. Tatiana Aires Tavares
Examinador externo
Universidade Federal de Pelotas

Natal-RN
Abril 2018

Este trabalho é dedicado à todos que já desejaram um mundo melhor.

Agradecimentos

Agradeço ao meu orientador Rummenige Rudson Dantas, que me fez enxergar detalhes que me eram invisíveis e confiou nas minhas ideias alternativas.

Agradeço aos meus pais que me ajudaram até aqui e souberam compreender o quanto era importante o meu trabalho, mesmo sem enxergar todo ele.

Agradeço à toda equipe TEAM e os demais envolvidos da UFRN, por todo o acolhimento e suporte.

Agradeço à todos dentro do IFRN e da UFRN que agiram de forma a nos dar esta oportunidade.

"O que as pessoas chamam de amor é apenas uma reação química que obriga os animais a se reproduzir. Isso te atinge de um jeito Morty, e então vai desaparecendo lentamente, deixando você encalhado em um casamento falho. Eu fiz isso, seus pais vão fazê-lo. Quebre o ciclo, Morty. Levante-se. Concentre-se na ciência.

(Rick Sanchez)

Resumo

A terapia ocupacional e a fisioterapia vêm ganhando nos últimos anos apoio da tecnologia aplicada em games. Dispositivos para VR tem sido utilizados juntamente com exergames para motivar o paciente a frequentar a terapia, entretendo-o ao mesmo tempo que realiza a terapia. Porém os consoles ainda não apresentam sensores capazes de identificar todos os movimentos necessários para os vários tipos de terapia, por exemplo, o Wii não é capaz de identificar se a mão do jogador estava fechada ou não. Dessa forma o TEAM (Laboratório de Tecnologias Educacionais, Assistivas e Multimídia), iniciou o desenvolvimento de tecnologias e exergames para aplicação em terapia física e ocupacional.

Durante o desenvolvimento de novos dispositivos de interação, foi observado que a cada novo hardware era necessário modificar o código-fonte dos games já produzidos ou criar um novo game para tal dispositivo. Esse problema caracteriza um forte acoplamento entre o hardware e os exergames, bem como retrabalho, consequentemente havia dispositivos completamente desenvolvidos, mas que não era possível utilizá-los com um exergame. Logo notou-se a necessidade de uma ferramenta para intermediar tais artefatos.

Este trabalho apresenta o TeamBridge, um middleware que tem por finalidade intermediar a comunicação entre dispositivos de interação para VR e exergames com foco em terapia física e ocupacional. Este middleware foi testado com um game comercial, provando que não há necessidade de modificação do código-fonte. Ele também passou por testes de performance, a fim de garantir que a experiência do jogador não seja comprometida. Ao final a ferramenta foi disponibilizada no GitHub.

Palavras-chave: Gameterapia. Adaptador de dispositivos. VRPN. Comunicação entre dispositivos. Exergaming jogos sérios.

Abstract

Occupational therapy and physiotherapy have been gaining support in the last few years of applied technology in games. VR devices have been used together with exergames to motivate the patient to attend the therapy, entertaining him at the same time as performing the therapy. But the consoles still do not have sensors capable of identifying all the movements required for various types of therapy, for example, the Wii can't identify whether the player's hand was closed or not. In this way the TEAM (Laboratory of Educational, Assistive and Multimedia Technologies) started the development of technologies and exergames for application in occupational therapy.

During the development of new input devices, it was observed that with each new hardware it was necessary to modify the source code of the games already produced or to create a new game for such a device. This problem characterizes a strong coupling between hardware and exergames, as well as rework, consequently there were fully developed devices, but it was not possible to use them with an exergame. Soon the need for a tool to intermediate these artifacts was noted.

This paper presents our middleware that aims to intermediate communication between VR input devices and exergames with a focus on occupational therapy. This middleware was tested with a commercial game, proving that there is no need to modify the source code. He also underwent performance testing to ensure that the player's experience is not compromised.

Keywords: Game Therapy. Device Adapter. VRPN. Cross device communication. Exergaming serious gaming.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API *Application Programmer Interface*

AR *Augmented reality*

ARTiFICe *Augmented Reality Framework for Distributed Collaboration*

AsTeRICS *Assistive Technology Rapid Integration & Construction Set*

ATAM *Architecture Tradeoff Analysis Method*

AVC Acidente vascular cerebral

CAC *Controller Application Communication*

DTW *Dynamic Time Warping*

FAAST *Flexible Action and Articulated Skeleton Toolkit*

GeMiNI *Generic Multi-Modal Natural Interface Framework for Videogames*

GoF *Gang of Four*

HCIˆ2 *Human-Centred Intelligent Human–Computer Interaction*

HTML *HyperText Markup Language*

IDE *Integrated Development Environment*

IGER *Intelligent Game Engine for Rehabilitation*

IMU *Inertial Measurement Unit*

IP *Internet Protocol*

IR *Infrared*

JS JavaScript

JSON JavaScript *Object Notation*

KHRD *Kinect-based system for ensuring home-based rehabilitation*

MR *Mixed Reality*

NITE *Natural Interaction Middleware*

NUI *Natural User Interface*

P/S *Publish Subscribe*

PITS *Paediatric Interactive Therapy System*

RV Realidade Virtual

SDK *Software Development Kit*

TCP *Transmission Control Protocol*

TOF *Time of flight*

UDP *User Datagram Protocol*

UFRN Universidade Federal do Rio Grande do Norte

UML *Unified Modeling Language*

USB *Universal Serial Bus*

USC *Institute for Creative Technologies - University of Southern California*

USEFIL *Unobtrusive Smart Environments for Independent Living*

VPN *Virtual Private Network*

VRPN *Virtual-Reality Peripheral Network*

Sumário

1 Introdução

Os exergames são jogos que exigem que o usuário se movimente para jogar (OH; YANG, 2010). Desde sua introdução à terapia, tanto em fisioterapia quanto em terapia ocupacional, foram realizados estudos sobre sua eficácia, gerando até mesmo estudos de classificação do uso de games para reabilitação (REGO; MOREIRA; REIS, 2010). Em alguns desses estudos já foi possível notar significância estatística sobre a melhora de pacientes (BÔAS et al., 2013) (BARCALA et al., 2011).

O Nintendo Wii alavancou a popularidade desse tipo de aplicação, pois permitiu o controle do game através de movimentos reais. Os pacientes fazem exercícios enquanto se divertem jogando. Algumas vezes a imersão é tão grande que os pacientes esquecem momentaneamente o desconforto ou a dor. Notou-se também que a variedade de perfis de jogadores ficou bem diversificada, pessoas de várias idades gostam de jogar no Nintendo Wii (REGO; MOREIRA; REIS, 2010).

Apesar de capturar diversos movimentos o hardware disponível ainda não tinha capacidade de capturar todos os dados relevantes, como o gesto de pinça ou agarrar objetos mão (MATHIOWETZ et al., 1985). Com isso o TEAM (Laboratório de Tecnologias Educacionais, Assistivas e Multimídia), pertencente à rede de laboratórios Natalnet, situado na UFRN (Universidade Federal do Rio Grande do Norte), iniciou com a pesquisa de desenvolvimento de dispositivos de interação para VR e exergames para e terapia física e ocupacional. A pesquisa foi desenvolvida em parceria com a base de pesquisa em TI aplicada à fisioterapia.

A pesquisa no TEAM também é voltada para hardware com menor custo, Kinect da Microsoft e o Leap Motion[1] podem ser utilizados (BIANOR; CAVALCANTI; DANTAS, 2017a), entretanto pretende-se obter o mesmo resultado com menos investimento na compra de equipamentos.

Como resultado da pesquisa, vários tipos de dispositivos foram criados (BIANOR; CAVALCANTI; DANTAS, 2017b) (SILVA et al., 2016) (SILVA et al., 2013). Cada novo dispositivo traz um novo protocolo de comunicação e outros tipos de dados. Consequentemente, um novo game deve ser desenvolvido ou os games já desenvolvidos têm que ser modificados para aceitar o novo dispositivo. Essa situação caracteriza um forte acoplamento entre os dispositivos e seus games, diminuindo a possibilidade de reutilização e dificultando a sua manutenção.

[1] Leap Motion - Hardware para captura de movimento de mãos desenvolvida para Realidade Virtual (RV) e Realidade Aumentada (RA).

Para a resolução desse problema, resolveu-se criar o TeamBridge, *middleware*[2] que permite a utilização dos dispositivos com os games já desenvolvidos de forma não invasiva, dispensando a modificação do código-fonte destes, de modo que a produção dos games leve em consideração apenas comandos do teclado ou mouse. Para cada dispositivo de interação, um *driver*[3] deve ser escrito e dependendo dos dados gerados pelo dispositivo, também pode ser necessário escrever um interpretador. Sendo assim ele é como uma ponte entre os dispositivos e os games, por esse motivo o nome TeamBridge.

O termo "não invasivo" pode possuir vários significados dependendo do contexto, não deve ser confundido com "Computação Invasiva" que segundo PRESMAN (2011, p. 705) "focaliza a integração de sistemas baseados em software em um ambiente mais amplo do que um PC". Neste trabalho "não invasivo" faz referência ao mesmo termo no contexto de criação de variabilidades para um software, onde tenta-se diminuir ao máximo a modificação do código-fonte original. Segundo Pohl, Böckle e Linden (2005, p. 64), uma variabilidade é modelada para customizar uma aplicação, ativando ou desativando uma funcionalidade, ajustando o artefato gerado.

1.1 Objetivo geral

O objetivo geral deste trabalho é definir uma arquitetura que permita o maior desacoplamento entre as ferramentas. Esse desacoplamento permitirá que qualquer dispositivo com *driver* para este projeto, possa controlar qualquer game sem a necessidade de criar um novo código de controle para cada game.

1.2 Objetivos específicos

- Conceituar a terapia com exergames.
- Detalhar os dispositivos e tecnologias utilizadas no projeto.
- Identificar possíveis abordagens para resolução do problema.
- Definir da arquitetura para desacoplamento entre os games e dispositivos, permitindo a utilização com games que possuam código-fonte fechado.
- Garantir que o *middleware* não atrapalhe a experiência do jogador.
- Criar uma documentação para a arquitetura desenvolvida neste projeto.
- Disponibilizar o código fonte do projeto.

[2] Middleware - Software que faz a mediação entre outro software e demais aplicações.
[3] *Driver* - Software que permite a utilização do dispositivo.

1.2.1 Detecção de movimentos incorretos

Até o presente momento, somente dispositivos de captura de pontos, como o Kinect ou Leap Motion, fornecem dados para identificar se o paciente está realizando um movimento de maneira incorreta, os outros dispositivos desenvolvidos pelo TEAM ainda não podem fornecer dados suficiente para chegar a tal conclusão.

1.3 Motivação

A principal motivação para o problema é o retrabalho para readaptar games que já foram produzidos. Em alguns casos a readaptação é difícil pois o código tem que ser estudado e até mesmo corrigido para novas versões do Unity, daí vem a preferência por uma arquitetura que não requisite a modificação do código fonte dos games. Outro problema observado foi o desenvolvimento de dispositivos que no final não puderam ser compatibilizados com os games já produzidos, resultando em um dispositivo que não possui nenhum game compatível. Da mesma forma também há games inteiramente desenvolvidos mas que não são mais utilizados porque o único dispositivo compatível ficou obsoleto.

1.4 Metodologia

Para o desenvolvimento deste trabalho foi necessário realizar um levantamento do estado da arte, encontrando pesquisas que tentaram solucionar o mesmo problema. Realizou-se uma pesquisa exploratória com base documental utilizando a ferramenta Google Acadêmico. Levou-se em consideração artigos e dissertações a partir do ano de 2008 até 2018. As opções de incluir patentes e citações foram desmarcadas.

Para formar as frases de busca 1 e 2 foram utilizados os termos *Framework*, *Middleware*, *Game engine* e Motor de game, com obrigatoriedade da presença das palavras *rehabilitation* e *input device* ou reabilitação e dispositivos de interação. A Tabela 1 mostra a frase de busca e a quantidade de artigos encontrados e descartados.

Após a pesquisa foi necessário realizar filtros para descartar artigos com pouca relevância para o trabalho. O primeiro filtro foi aplicado em artigos que não possuíam nenhuma relação com o trabalho proposto. Apesar da quantidade foi observado que muitos artigos estavam repetidos. Boa parte dos artigos estavam relacionados à aplicações mobile, sem nenhuma relação à terapia e sim à criação de games para a plataforma. Foi possível identificar outros assuntos como: representação de ambientes 3D, desenvolvimento de simuladores e aplicações de realidade virtual em geral.

Após o filtro inicial, foi feita a leitura dos artigos. Com a leitura foi possível verificar que alguns artigos ainda não estavam relacionados ao tema. Com isso, aplicou-se um segundo descarte. Esse descarte utilizou como filtro artigos que não possibilitavam o uso

de mais de um dispositivo de interação ou que apenas mostravam resultados obtidos em terapias com o uso de um dispositivo específico, não tratando do principal problema ao qual esta dissertação está relacionada. Apesar desse último filtro, alguns artigos descartados foram aproveitados no referencial teórico e tecnológico.

Com o intuito de obter algumas visões fora do contexto de reabilitação a mesma pesquisa foi realizada removendo as palavras chaves *rehabilitation* e reabilitação. Como o objetivo era apenas obter uma visão de outras aplicabilidades, somente 10 artigos foram selecionados da busca em inglês, a busca em português não obteve o mínimo de 10 artigos.

Apesar de não estar relacionado ao problema principal, porém, estar ligado à um objetivo específico, os artigos de Huang et al. (2012) e Su (2013) encontrados nessa busca, foram incluídos no capítulo de trabalhos relacionados.

Além dos artigos encontrados com essas frases de busca os trabalhos relacionados dos artigos lidos também foram observados, onde foi possível encontrar o *Flexible Action and Articulated Skeleton Toolkit* (FAAST) (SUMA et al., 2011) e o HCIˆ2 (SHEN; PANTIC, 2009).

Para embasamento do hardware a ser utilizado, também foram adquiridos os artigos do TEAM referentes aos dispositivos desenvolvidos. Assim foi possível obter um entendimento prévio do funcionamento dos dispositivos.

Tabela 1 – Resultado das frases de busca

#	Busca	Total	Filtro 1	Filtro 2	Obtidos
1	framework OR game engine OR middleware “rehabilitation” “input device”	846	800	39	7
2	framework OR motor de game OR middleware “reabilitação” “dispositivo de entrada”	360	82	3	1
3	framework OR game engine OR middleware “input device”	6770	6660	8	2
4	framework OR motor de game OR middleware “dispositivo de entrada”	384	381	1	2

Com o resultado de todos os artigos encontrados foram montados os capítulos de Referencial teórico e tecnológico e o de Trabalhos Relacionados. No capítulo de trabalhos relacionados tentou-se identificar, sempre que possível, os protocolos, tipos de hardware, games compatíveis, sistema operacional utilizado e a motivação para a escolha de cada um desses itens.

Com base nos trabalhos encontrados, foi definida uma arquitetura adequada à realidade dos dispositivos utilizados pelo TEAM, tanto em nível de hardware como protocolos.

Para o desenvolvimento do projeto seguiu-se uma metodologia tradicional de

desenvolvimento (PRESMAN, 2011, p. 40), com protótipos (SOMMERVILLE, 2011, p. 30) iniciais para verificar a viabilidade da arquitetura e o entendimento das tecnologias utilizadas.

Durante o processo de desenvolvimento uma documentação foi criada e está disponível no próprio repositório do GitHub[4] como um projeto público no endereço <https://github.com/klingerkrieg/Team-Bridge/wiki> e conta com instruções para instalação, configuração, compilação, diagramas da *Unified Modeling Language* (UML) 2.0 e o Diagrama de entidade e relacionamento para o modelo de banco de dados.

Para garantir uma maior manutenibilidade, o projeto contém testes unitários, conforme Sommerville (2011, p. 162) explica, o teste unitário é o processo de testar cada componente do programa, métodos ou classes. Esses testes são apenas chamadas para cada rotina com parâmetros diferentes. Para a construção dos testes foi utilizada a ferramenta de testes do Microsoft Visual Studio 2017.

Para fins de validação quanto ao objetivo de garantir o uso sem prejuízo no desempenho, foram realizados testes de carga em várias situações, inclusive aproveitando a ideia da comparação realizada por Shen e Pantic (2009) entre a ferramenta elaborada pelo autor e as já existentes, realizou-se também uma comparação entre o projeto proposto e o FAAST, que será abordado no capítulo de trabalhos relacionados. Detalhes dos testes estão descritos no capítulo 5.4.

1.5 Organização do trabalho

Este trabalho está organizado em 5 capítulos. O presente capítulo, Introdução, pretende apresentar e contextualizar o problema de investigação, os objetivos do projeto e metodologia adotada. O segundo capítulo, Referencial teórico e tecnológico, apresenta os principais conceitos abordados durante a pesquisa, seguido pelo capítulo de Trabalhos Relacionados onde serão apresentadas outras pesquisas que tentaram resolver o mesmo problema ou propuseram uma solução semelhante à proposição desse trabalho. O Quarto capítulo trata do projeto e da arquitetura desenvolvida para solução do problema. Em seguida o capítulo de Resultados apresenta as peças de softwares desenvolvidas e os testes realizados. Por último o capítulo de Conclusão, onde se encontram as considerações finais e possíveis trabalhos futuros.

[4] Github - Servidor onde é possível hospedar projetos versionados pelo Git. <https://github.com/>

2 Referencial teórico e tecnológico

Este capítulo apresenta os conceitos que serão e explorados, juntamente com os dispositivos utilizados nessa pesquisa. Será dada uma breve explicação do funcionamento de cada hardware e algoritmo que poderá ser utilizado.

2.1 Reabilitação

Segundo (WATERS, 1994) reabilitação é o processo de permitir ou facilitar a restauração de deficientes para que consigam se restabelecer fisicamente, socialmente e psicologicamente. Entretanto o autor finaliza com a frase “até o nível em que os pacientes estejam aptos ou motivados” (WATERS, 1994). Ou seja a reabilitação depende da motivação dos pacientes.

Quanto à aplicação, tanto este projeto quanto a pesquisa realizada no TEAM têm como foco a reabilitação motora. Além dessa existe a reabilitação cognitiva, onde o paciente treina a memória, atenção, concentração, raciocínio, resolução de problemas, linguagem, julgamento entre outros (REGO; MOREIRA; REIS, 2010). A reabilitação motora pode ser devido a um Acidente vascular cerebral (AVC), dano cerebral ou doença de Parkinson. Pode afetar membros superiores, inferiores e extremidades. É tratada com treino espacial, treino de equilíbrio ou até atividades do dia a dia.

Quando o foco é em tarefas do dia a dia é utilizada a terapia ocupacional, que é explicada por (REED; SANDERSON, 1999) como a aplicação de tarefas do dia a dia que tenham um propósito para o paciente e que ajudem ou permitam que ele consiga, recupere, melhore ou previna a perda de habilidades rotineiras. Tem como objetivo permitir que o individuo seja capaz de contribuir socialmente, culturalmente e economicamente. Como a terapia ocupacional é bem ampla, cada paciente é tratado de acordo com as suas necessidades, por exemplo, crianças com problemas de desenvolvimento podem ser ajudadas a aprender a escovar os dentes, vestir-se ou despir-se sozinhas (GUERZONI et al., 2008). Idosos com problemas de memória participam de atividades práticas que estimulem a memória e funções cognitivas como: desenhos livres e direcionados, leitura, jogos de lógica e raciocínio, filmes, pintura e socialização (SOARES et al., 2011), caça-palavras, palavras-cruzadas, participação em cursos e atividades de lazer indiretamente ligadas à memória, como costurar, viajar, realizar meditação, participar de atividades sociais e familiares e de atividades físicas, como também atividades instrumentais de vida diária como preparo de refeição, uso de meios de transporte, controle do orçamento financeiro (SATO; BATISTA; ALMEIDA, 2014).

2.1.1 Terapia com exergames

Bôas et al. (2013) realizou um estudo de caso para avaliar se a terapia com Wii trazia resultados. Participaram do estudo 3 pacientes de 6 a 10 anos de idade, duas com paralisia cerebral e uma com traumatismo craniano encefálico. Segundo o autor os resultados foram satisfatórios, uma vez que obteve-se até mesmo uma significância estatística em alguns pacientes.

No estudo do Barcala et al. (2011), foi realizada uma comparação entre a terapia convencional e com o uso do programa Wii Fit que é utilizado com o acessório Wii *Balance Board*, uma balança que captura o equilíbrio do jogador. Um total de 12 pacientes participaram do estudo e foram divididos em dois grupos. Foi obtido maior controle de equilíbrio em ambos os grupos, contudo o grupo que utilizou Wii Fit conseguiu diminuição da oscilação ângulo-posterior, enquanto que o grupo com tratamento convencional não obteve uma melhora nesse quesito.

Rego, Moreira e Reis (2010) sugerem que o sucesso da terapia com exergames está ligado ao fato da diversão proporcionada ao paciente, uma vez que durante esse tipo de terapia o paciente fica tão imerso que esquece o desconforto e a dor. Da mesma forma ele fica mais engajado em continuar com a terapia, pois torna-se uma tarefa mais agradável, uma vez que o maior problema da terapia são os exercícios repetitivos.

2.2 Desacoplamento e reuso

O problema do reuso é comum em games que precisam receber dados de vários dispositivos de interação, o trabalho de Santos, Carvalho e Bressan (2012), por exemplo, apesar de utilizar um *middleware*, ainda requer que o game seja compilado junto ao *middleware*. Caso este obtenha compatibilidade com um novo dispositivo o game deverá ser recompilado. Isso nos trás novamente ao problema de código fonte antigo que precisará ser estudado.

Vários padrões de projeto do *Gang of Four* (GoF)[1] tentam definir métodos para obter um maior desacoplamento da aplicação. A vantagem de poder trabalhar em uma parte do sistema de forma que não afete os trabalhos anteriores é tão importante que existem várias estratégias para tratar do assunto. Gamma (2009, p. 24-25) explica alguns padrões que garantem um melhor desacoplamento como: *Adapter*, *Bridge*, *Mediator* e *Proxy*.

Este projeto trata da intermediação entre dois sistemas, o hardware codificado para funcionar somente com um game e o próprio game que só funciona com o hardware específico. O fato do game ser feito para o hardware específico é o problema principal desse

[1] GoF - (“Gang of Four”) surgiu nos meados da década de 90 por quatro projetistas de software e que são aplicados, nos dias atuais, no desenvolvimento de sistemas dentro do paradigma orientado a objetos.

trabalho. Por este motivo o resultado esperado é a vantagem gerada pelo desacoplamento: obter um sistema mais coeso, extensível e flexível.

No entanto é importante definir como se dará essa mediação entre o hardware e os games, como a ideia é fazer essa adaptação da forma menos invasiva o possível foi optado pelo desenvolvimento de um *middleware* e não um *framework*.

2.2.1 Por que um *middleware* e não um *framework*

Buschmann et al. (1996) explicam que um *framework* é um software parcialmente completo. Ele definirá uma arquitetura padrão para uma família de softwares, provendo os blocos básicos para construção. Ele também deve definir locais para adaptações e funcionalidades específicas. Para Pree (1995) *frameworks* são adequados para domínios onde várias aplicações similares são criadas várias vezes a partir do zero. O autor concorda com a ideia de que é uma estrutura base para criação de outros sistemas. No caso o TeamBridge não possui uma estrutura de base para outros sistemas, ele será usado mediando outros sistemas. O TeamBridge poderia ter adotado uma estrutura de *framework* caso tivesse sido construído como um *plugin* para o Unity, porém essa arquitetura tornaria-se mais invasiva, exigindo que o game fosse desenvolvido com o framework, e possivelmente dificultaria a atualização dos games para outros dispositivos de interação, já que poderia requerer a recompilação dos exergames. Um possível ponto positivo dessa estrutura seria uma maior praticidade ao executar o game, uma vez que não seria necessário iniciar um servidor e um cliente do TeamBridge, bastaria iniciar o exergame.

Para entender a necessidade de um *middleware* é preciso observar alguns de seus conceitos, segundo Schmidt e Buschmann (2003), *middleware* é um software que pode aumentar a reusabilidade ao prover uma estrutura padrão para as tarefas mais comuns como armazenamento, fila de mensagens, multiplexação ou controle de concorrência. Com essa definição o TeamBridge pode ser considerado um *middleware* por ter como um dos objetivos o reuso dos games e transformar tipos específicos de estruturas para controle dos games em uma única estrutura padrão que já é utilizada, o uso de mouse e teclado. Dentre as tarefas comuns oferecidas pelo TeamBridge estão o armazenamento das informações capturadas durante a sessão de terapia e por utilizar o *Virtual-Reality Peripheral Network* (VRPN) ele também herda os benefícios de uma conexão confiável uma vez que é tolerante à falhas.

Para Bakken (2002), *middleware* é uma classe de software desenvolvida para ajudar a gerenciar a heterogeneidade em sistemas distribuídos. Neste trabalho está previsto o uso do Leap Motion, Kinect V1, Kinect V2, NEDGlove e PhysioHappy, cada um desses dispositivos tem uma forma diferente de se acessar os dados, a ajuda do TeamBridge está em reduzir o número crescente de formas de acesso aos dispositivos para apenas uma.

2.3 RV - Realidade Virtual

A Realidade Virtual, termo com crescente uso na atualidade já é alvo de pesquisas desde os anos 70 como em *Responsive Environment* do Krueger (1977) apresentado na figura 1. Somente a partir dos anos 80 com o avanço tecnológico, a Realidade Virtual (RV) passa a ser mais acessível, deixando de ser viável somente para grandes empresas e instituições de pesquisa (NETTO; MACHADO; OLIVEIRA, 2002). Atualmente é possível encontrar vários dispositivos de Realidade Virtual de baixo custo, ou até mesmo utilizar smartphones como dispositivos de saída para RV (BRAGA; MOTA; COSTA, 2016).

Figura 1 – Videoplace em funcionamento

Fonte: Krueger, Gionfriddo e Hinrichsen (1985)

Para Latta e Oberg (1994) RV é criar e inserir o participante em um ambiente, fazê-lo sentir-se como se estivesse em outro lugar. Segundo Aukstakalnis e Blatner (1992 apud NETTO; MACHADO; OLIVEIRA, 2002) A RV permite a visualização e manipulação de representações extremamente complexas. “é um paradigma pelo qual usa-se um computador para interagir com algo que não é real, mas que pode ser considerado real enquanto está sendo usado ” (HAND, 1996 apud NETTO; MACHADO; OLIVEIRA, 2002).

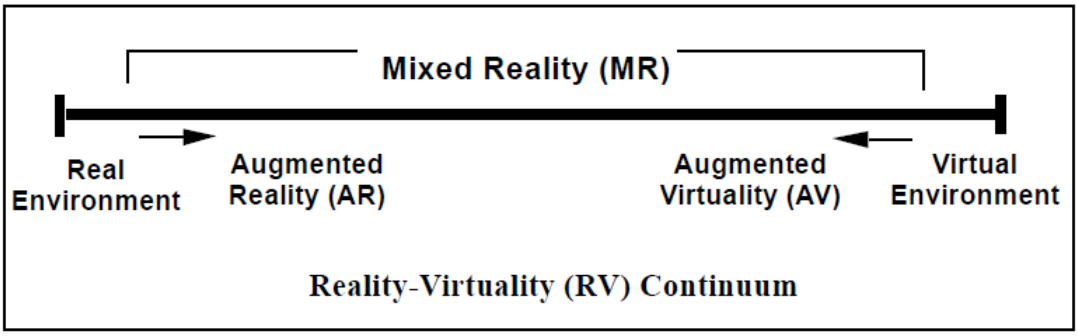
Aplicações de RV podem ser definidas em dois níveis, imersivas e não imersivas (LESTON, 1996). Segundo Robertson, Card e Mackinlay (1993) o conceito mais comum para RV envolve a imersão completa, com algum dispositivo de saída que proporcione uma visão completa do ambiente 3D, normalmente montado na cabeça do usuário, podendo ser acompanhado de luvas ou dispositivos para captura das mãos, mas também existe a versão alternativa da RV, a não imersiva onde o ambiente 3D é apresentado em um monitor e a entrada pode ser um mouse e teclado. Com isso ainda é possível obter vários níveis híbridos de RV, pode-se aumentar o grau de imersão a medida que se adiciona periféricos.

Ainda em relação aos níveis existentes de RV, Milgram et al. (1995) propôs a representação constante na figura 2, onde é possível observar uma mudança gradativa do conceito de *Augmented reality* (AR) para RV. Onde o extremo esquerdo estão presentes tecnologias que atuam sobre o ambiente real, apenas modificando-o, na direita um ambiente completamente virtual e no meio ele caracteriza como *Mixed Reality* (MR), onde entram

todas as hibridizações entre os conceitos. Segundo o autor existem as aplicações MR podem ser classificadas em 3 dimensões:

- Alcance do conhecimento de mundo: Se o ambiente possui pouca influência computacional ou se foi totalmente modelado.
- Fidelidade da reprodução: Desde gráficos apenas com contorno de objetos à definição com alta fidelidade.
- Extensão da metáfora da presença: Diretamente ligado à imersão, se está sendo utilizado um monitor, uma tela larga ou um dispositivo montado na cabeça.

Figura 2 – RV Continuum



Fonte: Milgram et al. (1995)

Dessa forma os hardwares analisados nesse projeto podem ser considerados como dispositivos de interação para RV, ainda que não haja necessidade de montar algum dispositivo de saída na cabeça do usuário.

2.4 Dispositivos de interação

Esta sessão irá apresentar os dispositivos que foram avaliados nesta dissertação com uma análise sobre o seu funcionamento, sempre tentando ressaltar pontos como: Linguagem utilizada, modo de conexão com o PC e princípio de funcionamento.

2.4.1 NED Glove

De acordo com Murthy e Jadon (2009) sistemas de realidade virtual possuem basicamente dois modos de captura de dados: visão computacional ou luva de dados. A visão computacional se baseia em técnicas não invasivas, normalmente com câmeras. Luva de dados são dispositivos que utilizam sensores mecânicos ou ópticos que capturam

o quanto o dedo está flexionado e traduzem para sinais elétricos, com isso é possível determinar em qual pose a mão se encontra, normalmente depende de cabos conectados ao computador. Uma das luvas de dados desenvolvida pelo TEAM é a NED[2] *Glove* (SILVA et al., 2013), equipamento de baixo custo, comparado a outros equipamentos semelhantes como a DG5-VHand[3], que chega a custar US$ 600,00 enquanto que o projeto do TEAM ficou em média US$ 300,00, essa luva utiliza JavaScript (JS) e C# e é conectada via *Universal Serial Bus* (USB), a saída gerada é algo semelhante à "760,730,831,720,710, -10.8,20.2,180.0", onde cada número representa um dedo e o quanto ele está flexionado, a figura 3 apresenta a versão publicada.

Figura 3 – NED-Glove

Fonte: Silva et al. (2013)

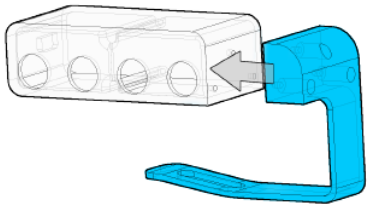
2.4.2 PHYS.IO

O PHYS.IO (SILVA et al., 2016) é outra luva de dados desenvolvida pelo TEAM. Esse dispositivo utiliza sensores capacitivos para captura dos dados, possui um design que permite ao usuário a utilização de forma prática, não há a necessidade de vestir, basta encaixar o dispositivo na mão, calibrar e utilizar. Para se comunicar com um PC, primeiramente ele precisa se conectar com um aplicativo no smartphone via Bluetooth, em seguida esse aplicativo envia para o computador os dados no formato JavaScript *Object Notation* (JSON) via *User Datagram Protocol* (UDP), esse protocolo foi escolhido para obter o melhor desempenho possível. Um dos projetos futuros é o desenvolvimento de uma

[2] NED - Acrônimo para NeuroErgoDesign
[3] DG5-VHand - <http://www.dg-tech.it/vhand3/products.html>

API para comunicação com a *Game Engine*[4] Unity 3D. É possível visualizar um esquema do dispositivo na figura 4.

Figura 4 – PHYS.IO



Fonte: Silva et al. (2016)

2.4.3 PhysioHappy

Com o foco de obter o maior baixo custo possível, foi desenvolvida a PhysioHappy (BIANOR; CAVALCANTI; DANTAS, 2017b), uma luva de dados construída apenas com pano, elástico, velcro, garrafa PET e um mouse. Os dados gerados pelo dispositivo são apenas os cliques do mouse, limitando-se à quantidade de botões existentes, mantém a conexão natural do mouse, via USB, entretanto ele interpreta quando o usuário flexiona para um lado ou para o outro, pode ser adaptado para diversas partes do corpo como: joelho, cotovelo, punho ou tornozelo. A figura 5 mostra o dispositivo vestido na mão para captura do movimento do punho.

Figura 5 – PhysioHappy

Fonte: Bianor, Cavalcanti e Dantas (2017b)

[4] *Game Engine* - Motor de games, ambiente que possui diversas ferramentas já prontas para desenvolvimento de games

2.4.4 Leap Motion

Este projeto também prevê a utilização do Leap Motion (BIANOR; CAVALCANTI; DANTAS, 2017a), outro dispositivo de interação RV. Este utiliza visão computacional, não requer que o usuário vista ou segure qualquer equipamento. Apesar de ser um equipamento proprietário, o TEAM o utiliza nas pesquisas sobre reabilitação.

Figura 6 – Leap Motion

Fonte: <http://burke.weill.cornell.edu/research/research-equipment/leap-motion>

A figura 6 apresenta o dispositivo e uma demonstração do seu uso, com a captura das mãos sendo representada no monitor.

O Leap Motion utiliza câmeras infra-vermelho para observar uma área em torno de 1 metro, conecta-se ao computador via USB e fornece uma *Application Programmer Interface* (API) para acesso aos dados. Desenvolvida com foco para captura de mãos e dedos, a própria *Software Development Kit* (SDK) já realiza o reconhecimento dos gestos mais importantes, o de fechar e abrir a mão e o gesto de pinça, dessa forma, estas informações serão reaproveitadas pelo projeto. O Leap Motion possui uma acurácia maior do que o Kinect, obtendo uma precisão de 0,7mm, valor que não pode ser atingido com o Kinect que possui 1,5cm de precisão (WEICHERT et al., 2013).

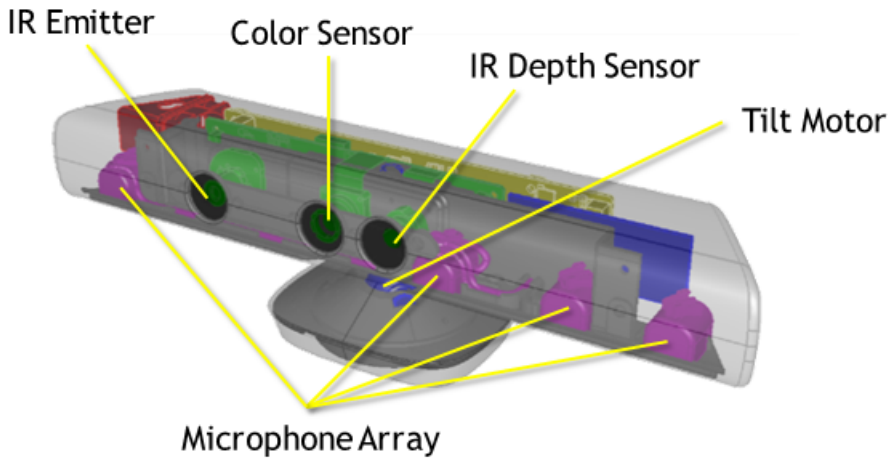
2.4.5 Microsoft Kinect

Lançado em 2010, o primeiro Kinect foi desenvolvido pela pela PrimeSense e Microsoft. Capaz de identificar movimentos de uma pessoa dinamicamente em tempo real, ele fornece um *stream* de imagens 2D em cores, um *stream* de imagens 3D com profundidade e um *stream* de pontos em um plano 3D referentes ao esqueleto do usuário, cada ponto possui informação sobre juntas como: mão, pulso, cotovelo, ombro, entre outros (LUN; ZHAO, 2015). Inicialmente projetado para utilização com o Xbox e sem possibilidade de conexão com computadores comuns, apesar disso pouco tempo após o seu lançamento a empresa Adafruit[5] lançou um concurso para desenvolvimento de um

[5] Adafruit - <http://www.adafruit.com/>

driver para o Kinect poder ser acessado através de um computador comum. Hector Martin foi o vencedor desse concurso iniciando com o desenvolvimento do Open Kinect, sendo o primeiro *driver* disponível para PC (GILES, 2010 apud CARVALHO, 2013). Após um certo tempo a PrimeSense disponibilizou os *drivers* oficiais do Kinect, o OpenNI, permitindo o acesso a mais dados do que o *driver* anterior. Rapidamente o uso do Kinect se estendeu a outras áreas: games, saúde, educação, treinamento, realidade virtual, robótica, linguagens de sinais e outros (LUN; ZHAO, 2015).

Figura 7 – Kinect Hardware



Fonte: <https://msdn.microsoft.com/en-us/library/jj131033.aspx>

O Kinect atua como um sensor de profundidade. Como pode ser visto na figura 7, nele estão presentes uma câmera normal, um emissor e um receptor de *Infrared* (IR). A forma de descobrir a profundidade é diferente na versão para Xbox 360 para o Kinect v2.

A tecnologia do Kinect v1 foi desenvolvida pela PrimeSense, a profundidade de cada pixel é calculada pela triangulação, normalmente são utilizadas duas câmeras para facilitar a triangulação, porém foi utilizada uma técnica para permitir esta descoberta com apenas um emissor de infravermelho e um sensor de profundidade. O emissor envia luzes com padrões predefinidos para o campo de visão, pela observação dos padrões o sensor pode inferir a linha do emissor IR até o pixel com o padrão, calculando a distância vertical entre o emissor IR, sensor de profundidade e o pixel usando trigonometria. Esse esquema apresenta falhas na medição de profundidade, para que possa funcionar o padrão de cada pixel deve ser único, por consequência, a câmera só consegue identificar 1 em cada 20 pixeis com fidelidade, os demais são interpolados. O problema se agrava quando há presença de luz forte (LUN; ZHAO, 2015).

O cálculo de profundidade utilizado pelo Kinect v2 é diferente, foi desenvolvido pela Canesta, que foi adquirida pela Microsoft em 2010 (KOLAKOWSKI, 2010), trata-se de

calcular a profundidade a partir do tempo que a luz infravermelha parte e retorna ao sensor, técnica conhecida como *Time of flight* (TOF), semelhante a um sonar (CARVALHO, 2013). Para obter um melhor resultado, é utilizado um contador de tempo que sincroniza o emissor IR e o sensor de profundidade. O emissor de IR é periodicamente ligado e desligado e a saída do sensor de profundidade é enviada para duas portas diferentes, A e B, quando a luz está ligada e desligada, respectivamente. A porta A possui a luz do emissor e a luz ambiente, B contém somente a luz ambiente. Após subtrair B de A é obtido somente a luz refletida do emissor IR, com isso a profundidade é calculada com mais precisão (LUN; ZHAO, 2015).

Outra mudança importante é que o Kinect *Runtime* v1 podia ser usado somente por uma aplicação e múltiplos Kinects poderiam ser controlados por uma única aplicação, esse software poderia ter todo o controle sobre o dispositivo, resolução, cor e *frames* de profundidade. No Kinect v2 o Kinect *Runtime* foi elevado ao nível de serviço de sistema, para facilitar o uso de coleta de dados pelo Kinect para múltiplas aplicações, como *tradeoff*[6] a aplicação perdeu o controle de resolução e não é mais possível utilizar vários sensores em um mesmo computador (LUN; ZHAO, 2015).

A mesma quantidade de sensores foi mantida, conquanto os sensores são mais robustos, o Kinect v1 não consegue reconhecer os dedos da mão (CAMERON et al., 2011), o v2 consegue diferenciar o polegar dos outros dedos, gerando dois pontos a mais, somando com mais um ponto no pescoço, em resumo o Kinect v1 consegue reconhecer 2 pessoas com 20 pontos cada uma, o v2 reconhece até 6 pessoas com 25 pontos (LUN; ZHAO, 2015). Conecta-se ao PC através da USB. Atualmente já existem 4 versões do Kinect porém somente duas para PC, o uso de uma SDK incompatível fará com que o hardware não seja reconhecido pelo computador. É possível o desenvolvimento com C++, C# ou Visual Basic. A tabela 2 resume os requisitos de SDK de cada versão do Kinect. Como o Kinect V2 requer Windows 8 ou superior o TeamBridge conterá uma versão sem suporte ao V2 para ser utilizado no Windows 7.

Tabela 2 – Versões do Kinect x SDK

Versão	SO	SDK
Kinect para Xbox 360	Windows 7	SDK 1.7 Dev. Tool Kit 1.7.0
Kinect para Windows	Windows 7	SDK 1.7 Dev. Tool Kit 1.7.0
Kinect para Xbox One	Windows 8	SDK 1.8/2.0 Dev. Tool Kit 1.8.0
Kinect para Windows v2	Windows 8	SDK 1.8/2.0 Dev. Tool Kit 1.8.0

Fonte: <http://www.instructables.com/id/How-to-Connect-a-Kinect/>

[6] *Tradeoff* - São decisões arquiteturais que afetam pelo menos dois atributos de qualidade do sistema.

O *Open Natural Interaction* (OpenNI), é um *framework* escrito em C++ que visa melhorar a compatibilidade de dispositivos de interface natural disponibilizando uma API e permitindo que as aplicações façam uso dos recursos de voz e gestos para interação sendo independente do sensor utilizado, funciona nas plataformas Windows, Linux e Mac OS X. Serve de base para outros softwares como o *Natural Interaction Middleware* (NITE). O NITE provê o reconhecimento de alguns gestos simples como: levantar mão, clique e onda (Prime Sense, 2011). O Kinect v2 também possui reconhecimento de gestos de mão como: fechada ou aberta (LUN; ZHAO, 2015).

2.4.6 Resumo dos dispositivos

A tabela 3 resume o tipo de conexão dos dispositivos envolvidos, a quantidade de protocolos e a heterogeneidade dos dados. O tipo *Button*, é apenas pressionado ou não, o *Analog* é ser uma faixa de valores e o *Tracker* é uma posição em um espaço tridimensional.

Tabela 3 – Resumo dos dispositivos utilizados pelo TEAM

Dispositivo	Conexão	Possui SDK	Tipo de dados
NEDGlove	USB	Não	Analog
PHYs.IO	Bluetooth	Não	Analog
PhysioHappy	USB	Não	Button
Leap Motion	USB	Sim	Tracker/Analog
Kinect v1	USB	Sim	Tracker
Kinect v2	USB	Sim	Tracker/Analog

2.5 VRPN - Virtual-Reality Peripheral Network

O VRPN é um protocolo de rede para dispositivos de interação para realidade virtual e independente de hardware. Conforme Taylor II et al. (2001) explica o VRPN foi desenvolvido para facilitar o uso e montagem de um laboratório de realidade virtual:

- Em laboratórios com múltiplas telas que requisitam dados de diversos periféricos de interação, é necessário reorganizar as máquinas de acordo com o equipamento ou passar cabos de vários dispositivos até o servidor.
- Alguns dispositivos de RV precisam ser reiniciados sempre que são reabertos.
- Diferentes dispositivos RV possuem diferentes interfaces, apesar de fornecerem basicamente as mesmas informações, além disso alguns requerem conexões especiais ou possuem *drivers* somente para um sistema operacional.
- Realidade virtual requer o mínimo de latência e é preciso saber quando que os eventos ocorreram.

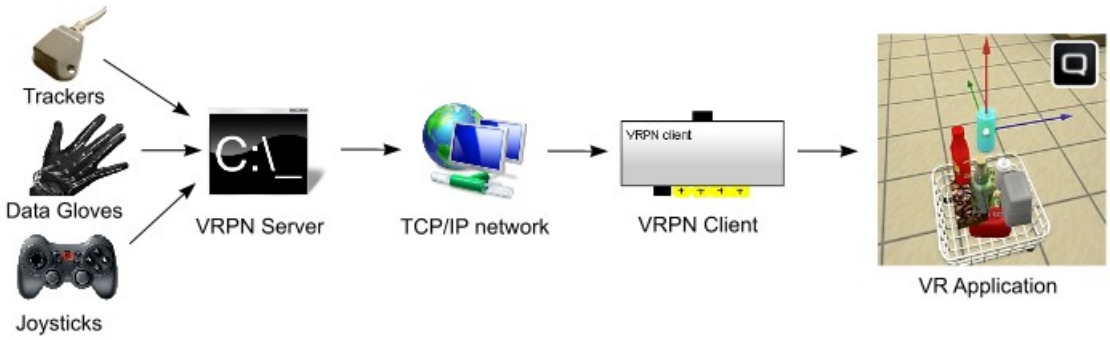
Alguns dos benefícios são:

- Acesso a uma variedade de dispositivos RV através de uma interface comum e extensível.
- Hora do evento em todas as mensagens de periféricos.
- Sincronização da hora entre servidores e clientes em diferentes máquinas.
- Múltiplos periféricos conectados simultaneamente.
- Reconexão automática em caso de falha.

Apesar do código inicial do VRPN ter sido iniciado em 1998, o código atual é compatível com Visual C++ 6.0 até o Visual Studio 2017, é aberto e encontra-se no endereço <https://github.com/vrpn/vrpn>, atualmente na versão 7.33. O código é escrito em C/C++ e possui pequenas implementações escritas em Java e Python que aproveitam o código em C/C++.

Quanto à arquitetura, os dispositivos de interação são conectados em servidores, podendo existir mais de um dispositivo conectado ao mesmo servidor. Enquanto que o cliente se conecta aos servidores e passa a receber todo o fluxo dos inputs repassando-os para uma aplicação de RV, a figura 8 apresenta esse esquema. Apesar de ser um protocolo para rede, servidor e cliente poderão estar na mesma máquina física.

Figura 8 – Funcionamento do VRPN.



Fonte: <http://3dvrm.com/vrpn/>

Em relação ao desempenho, segundo Taylor II et al. (2001) o VRPN consegue em alguns casos, ser melhor do que se o dispositivo estivesse conectado localmente, devido à melhorias realizadas em *drivers* específicos. É possível inclusive permanecer com o desempenho dentro do mínimo exigido para uso com RV que de acordo com Holloway (1997) é de 1mm de deslocamento para cada 1ms de latência.

3 Trabalhos relacionados

Este capítulo irá abordar o resultado da pesquisa bibliográfica para o entendimento do estado da arte. Aqui será possível compreender como outros trabalhos tentaram resolver problemas semelhantes ao discutido neste projeto, quais resultados eles obtiveram e sempre que possível qual foi a estratégia, hardware e algoritmos utilizados.

3.1 *Flexible Action and Articulated Skeleton Toolkit* (FAAST)

Middleware desenvolvido pelo *Institute for Creative Technologies - University of Southern California* (USC) (SUMA et al., 2011). Permite que o usuário realize gestos que podem ser configurados para acionar determinados comandos de teclado ou mouse. Desse modo, *Flexible Action and Articulated Skeleton Toolkit* (FAAST) consegue adaptar games para receber indiretamente inputs de sensores como Kinect. Atualmente encontra-se na versão 1.2 e não possui código aberto.

Conforme Suma et al. (2011) explica, o FAAST utiliza o OpenNI SDK juntamente com o NITE que são respectivamente uma biblioteca para operar com dispositivos de captura de movimentos corporais e um *middleware* que fornece um reconhecimento básico de gestos. Apesar do NITE, foi necessário implementar a captura de mais poses, uma vez que o reconhecimento oferecido pelo *middleware* não era suficiente. É compatível com o Kinect e o PrimeSensor[1]. Com ele é possível regular o limite dos ângulos da pose, resultando no aumento ou diminuição da sensibilidade de captura dos gestos e tem como um diferencial a possibilidade de alterar configurações sem a necessidade de reiniciar o software. Movimentos de braço, perna, andar estacionado, pular e até os movimentos básicos reconhecidos pelo NITE são computados pelo FAAST, convertidos para comandos de teclado ou mouse e enviados para o sistema operacional. O autor explica que foi implementado e incorporado um VRPN para o FAAST, mas não foi possível encontrar o motivo pelo qual foi preciso implementar um novo VRPN, no lugar de utilizar o já existente, apesar disso, nos testes de desempenho realizados neste trabalho foi verificado que o servidor do FAAST possui vantagem sobre o VRPN oficial, estes testes estão descritos na sessão 5.4.

O FAAST tem como trabalhos futuros a implementação de aprendizado de máquina para gravar novos gestos. Também pretende-se implementar a compatibilidade com outros controles como o Wii Mote ou PlayStation Move (SUMA et al., 2011). Atualmente a versão do FAAST para o Kinect V2 ainda está no estágio experimental.

[1] PrimeSensor - Câmera de profundidade desenvolvida pela PrimeSense

Devido a ideia do FAAST utilizar o VRPN com sucesso, o projeto proposto neste trabalho também utilizou esse protocolo, entretanto, ao contrário do FAAST, não foi necessário codificar um novo VRPN, o protocolo oficial foi utilizado, uma vez que ele possui código aberto e tem a vantagem de já ser compatível com pelo menos 60 dispositivos controladores (QUALISYS; GIRAULT; BOGER, 2016), entre eles o Wii Mote citado no parágrafo anterior. Como o Kinect não possui *driver* dentro do VRPN oficial, esse *driver* foi desenvolvido dentro do projeto do TeamBridge, mas também é possível utilizar o servidor do FAAST em conjunto com o cliente do TeamBridge, isso porque ambos utilizam o mesmo protocolo.

3.2 *Intelligent Game Engine for Rehabilitation* (IGER)

A fim de se obter as vantagem de reuso, Pirovano et al. (2013) propôs uma *game engine* voltada para terapia motora, foram definidas funcionalidades básicas que uma *game engine* voltada para terapia deve ter, dentre elas:

- Eficácia - Como foi realizada junto com especialistas em terapia, os movimentos disponíveis para interação com os games são eficazes para a reabilitação.
- Exergames configuráveis - Os games desenvolvidos são totalmente configuráveis pelo terapeuta, tanto os exercícios que serão realizados na sessão, quanto a dificuldade ou velocidade de execução.
- Monitoramento - A *game engine* monitora a execução dos movimentos realizados pelo paciente, normalmente esses movimentos são monitorados pelo terapeuta, porém a *game engine* utiliza sensores que além de detectar movimentos errados, poderão alertar ao paciente com sinais visuais: ícones, mensagens e alertas sonoros.
- Avaliação - A *game engine* é capaz de gravar e reproduzir todos os movimentos realizados pelo paciente durante a sessão. Isso é importante para que o terapeuta possa avaliar a evolução do estado do paciente durante a terapia.
- Abstração dos dispositivos de interação - Cada paciente possui uma dificuldade específica que só pode ser tratada com determinado exercício, por sua vez cada exercício só pode ser tratado com determinado tipo de dispositivo, dessa forma, a *game engine* deve estar preparada para se adaptar. Como resultado, a *Intelligent Game Engine for Rehabilitation* (IGER) é capaz de funcionar com Sony PlayStation Eye, Microsoft Kinect, Wii Balance Board, Omni Phantom, Novint Falcon[2], Tyromotion

[2] Omni Phantom e Novint Falcon - Dispositivos hápticos que possibilitam a simulação de manipulação de objetos no computador. O usuário move uma parte do dispositivo e ele envia as informações do movimento para o computador.

Tymo[3] e o Moticon OpenGo Insoles[4]. Para melhor entendimento a figura 9 apresenta o Omni Phantom e o Novint Falcon e a figura 10 apresenta o Tyromotion tymo e o Moticon OpenGO Insoles.

- *Natural User Interfaces* (NUIs) - NUIs são mais fáceis de aprender e intuitivas, elas tornam invisíveis a camada de comunicação com o aplicativo, utilizando microfones e câmeras os menus da *Game Engine* poderão ser manipulados por gestos ou voz.

- Terapeuta virtual - Como parte do objetivo de aumentar a motivação, foi incluído um terapeuta virtual que ajuda na utilização da execução dos exergames, podendo até parabenizar e incentivar o paciente durante suas novas conquistas.

Figura 9 – Omni Phantom e Novint Falcon

Fonte: Delft Haptics Lab (2014) e Artificial Intelligence Laboratory (2006)

Figura 10 – Tyromotion tymo e Moticon OpenGO Insoles

Fonte: Reha Stim (2014) e Krishnamurthy (2013)

[3] Tyromotion Tymo - Dispositivo com sensores para captura da postura, com foco para aplicações em terapia

[4] Moticon OpenGo Insoles - Possui sensores para captura de pressão, peso, equilíbrio e movimento é utilizado como sola do tênis.

Conforme Pirovano et al. (2014) explica, o monitoramento do IGER funciona da seguinte forma, cada monitor possui 3 propriedades, a ação associada, o intervalo de valores do correto até o incorreto e o grau de severidade dessa regra, de 0 a 4 onde 0 é aceitável que o paciente erre ou 4 onde não deve haver possibilidade de erros, esse valor depende do tipo de reabilitação. No início de um tratamento é comum que o paciente erre por não conseguir realizar o movimento. É aplicada a lógica difusa no intervalo de valores e no grau de severidade, o limite de valores irá gerar 4 classes (*Ok, Risky, Bad, Wrong*), o grau de severidade se baseia em 5 classes já existentes, *Silent, Log, Warning, Error, Shutdown*. Para descobrir o grau do alarme a ser acionado é aplicada a seguinte fórmula, onde x é o valor gerado pelo paciente, "x min" e "x max" é o intervalo de valores do aceitável até o inaceitável e "y min" e "y max" é o intervalo do alarme, de 0 até o grau de severidade escolhido.

$$y = y\,\text{min} + \frac{(y\,\text{max} - y\,\text{min})}{x\,\text{max} - x\,\text{min}}(x - x\,\text{min})$$

Com o resultado da equação o alarme gerado é enviado ao agregador de alarmes, que irá dar ou não um feedback ao paciente. O feedback pode variar de acordo com o nível da severidade, caso seja *Silent*, o sistema irá ajustar os parâmetros para permitir tais erros, a medida que a severidade aumenta o ajuste nos parâmetros é reduzido. Quando o alarme atinge o nível *Error* o game é pausado e o terapeuta virtual aparece para dar instruções. No nível máximo, *Shutdown* o sistema é finalizado, pois o paciente pode estar realizando um exercício de forma perigosa e uma mensagem é enviada para o hospital entrar em contato com o paciente.

A IGER é baseada na *game engine* Panda 3D e possui algumas funções de baixo nível[5] programadas em C++. Ela faz parte do projeto REWIRE (BORGHESE et al., 2012), e com essa *Game Engine* foi desenvolvido um conjunto de 6 de mini-games para o projeto REWIRE: *The Balloon Popper game*, *The Fruit Catcher game*, *The Scare Crow*, *The Hay Collect game*, *The Mix Soup game* e *The Animal Hurdler game*.

Possui três objetivos muito semelhantes ao presente projeto, permitir uso de diversos equipamentos com os games, manter um histórico do desempenho dos pacientes para que eles possam ser acompanhados por um terapeuta, mesmo que à distância e detectar e alertar quando movimentos incorretos são realizados, no entanto de forma menos abrangente que o IGER.

Essa *Game Engine* alimenta um banco de dados que é acessado através de uma plataforma Web, só então os terapeutas tem acesso às atividades realizadas pelos pacientes. O escopo desse projeto conterá o envio dos dados gerados pelos pacientes para um

[5] Funções de baixo nível - São procedimentos que operam perto da camada do sistema operacional, como entrada e saída de dados dos dispositivos de hardware

repositório de dados, no entanto a plataforma Web e os relatórios não farão parte desse projeto.

O escopo da arquitetura proposta neste trabalho é menor do que a do IGER, não é pretendido criar uma nova *Game Engine* com diversas funcionalidades. Há pontos do projeto IGER onde não há possibilidade de reaproveitamento, como a funcionalidade de configurar a dificuldade do game. O responsável por desenvolver essa funcionalidade é o próprio desenvolvedor do game, logo não é possível inserir no escopo do presente projeto algo que é de responsabilidade de outra equipe. Alguns pontos já alcançados com o IGER fazem parte do escopo desse projeto, conquanto de forma menos invasiva. O IGER requer que os games sejam desenvolvidos com ele, a proposta é que qualquer game possa ser utilizado e as funcionalidades de multi-hardware, alertas de movimentos incorretos e gravação da evolução sejam automaticamente incorporadas.

3.3 Arcabouço para Construção de games Ubíquos (uOS *Plugin*)

Trata-se de um trabalho de mestrado realizado por Santos (2016). Esse trabalho tem como foco a criação de uma estrutura para facilitar o desenvolvimento de games ubíquos para reabilitação. Conforme o autor menciona, games ubíquos tem várias definições, todavia sempre há um ponto em comum, "a inserção de elementos do mundo real interagindo e confundindo-se com o mundo virtual".

Conforme mencionado no trabalho, o desenvolvimento de sistemas ubíquos já possui algumas dificuldades inerentes, como a heterogeneidade de dispositivos computacionais e canais de comunicação. Necessidade de serem escaláveis, devem manter a funcionalidade e tempo de resposta mesmo com uma quantidade considerável de dispositivos presentes. Devem ser tolerantes à falhas. E por fim, espera-se que as aplicações sejam adaptáveis à mudanças no ambiente e seguras lidando com questões de privacidade, acessos indevidos ou não autorizados.

Como o foco do trabalho estava na reabilitação, também foram criadas estruturas que salvam dados para poder inferir o progresso do paciente e permitir uma futura análise pelo profissional de saúde.

Para o desenvolvimento do software foi utilizado um *middleware* já existente, o uOS, é um software que já lida com diversas questões de computação ubíqua e permite a sua extensão através de *plugins*. Para a validação da ferramenta um novo game foi projetado. O autor menciona que a adaptação para games comerciais requer um *wrapper* que iria mediar a interface do seu projeto e os games. Na validação, além de gestos simples e complexos para a interação com o game, também foram testados hardwares diferentes, foi utilizado um dispositivo Android, teclado e sensores *Inertial Measurement Unit*s (IMUs), são sensores que conseguem capturar a sua própria rotação.

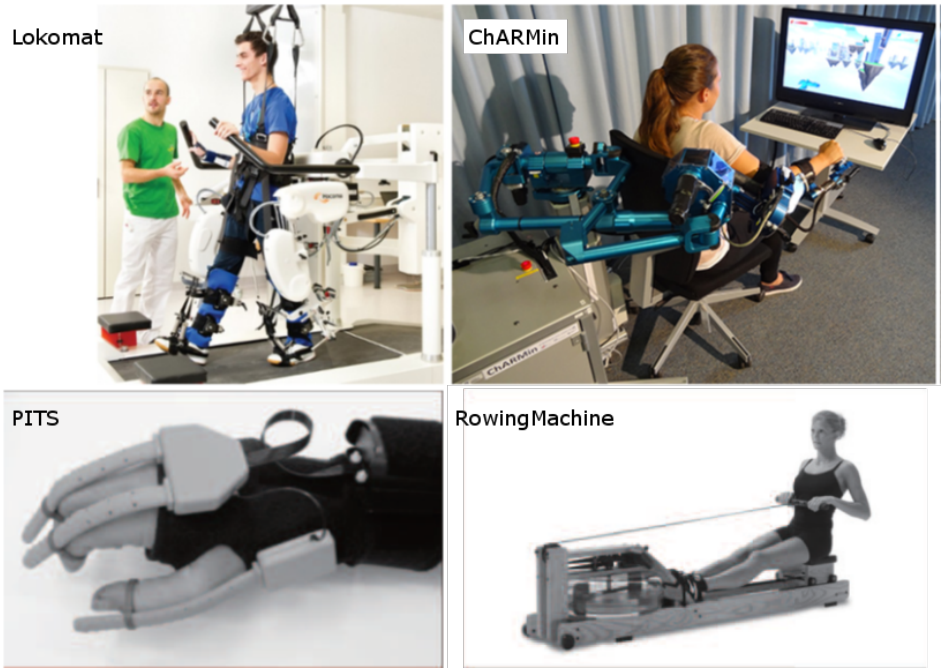
Em comparação ao trabalho deste capítulo o projeto proposto nesta dissertação leva a vantagem de já ser compatível com game desktop sem a necessidade de criação de qualquer outra estrutura, contudo o *plugin* relatado como trabalho futuro, sobre a criação de um *wrapper* para permitir a compatibilidade com outros games, poderia ser um *wrapper* único e generalista, de forma que não fosse necessário criar uma nova peça de software para cada game. Concluindo, o projeto como consta no trabalho ainda requer a modificação no código-fonte do game.

3.4 RehabConnex

O RehabConnex é um *middleware* para conexão entre games e diferentes tipos de controles, com o intuito da utilização em terapia e exercício físico.

Segundo Martin-Niedecken et al. (2015), esse *framework* foi testado com alguns games do Rehab Games, uma coleção de games para reabilitação, como *Gabarello* 1.3 e 2.0, *Magic Castle*, *Tornalino* e *FlyVinci*. Foram testados diferentes tipos de equipamentos como *Lokomat*, ChARMin, *Paediatric Interactive Therapy System* (PITS), *RowingMachine* e câmeras com software de reconhecimento. A figura 11 demonstra esses equipamentos.

Figura 11 – Dispositivos compatíveis com o Rehabconnex.



Fonte: Airindo (2014), Sensory-Motor Systems Lab (2013) e Martin-Niedecken et al. (2015)

O *Gabarello* 1.3 e 2.0 foram originalmente construídos para serem controlados com o *WaterRower*, um equipamento de remo, mas através do *framework* ambos os games

foram adaptados para funcionar com o *Gait Robot Lokomat* e o PITS, que funcionam como uma esteira e uma luva respectivamente. O RehabConnex utiliza conexões TCP/IP[6] para intermediar os dispositivos com os games porque essa é a saída nativa dos dispositivos, apesar de usar um protocolo TCP/IP não foi mencionado o protocolo VRPN nesse artigo. Nos projetos do TEAM nenhum dispositivo possui conexões de rede, existe somente USB e Bluetooh, contudo é importante considerar que pode-se desenvolver um equipamento que use TCP/IP, portanto essa dissertação já prevê essa situação.

3.5 HCIˆ2

O HCIˆ2 (SHEN; PANTIC, 2009) é uma ferramenta que torna mais fácil a interação Multimodal Humano-Computador. Essa interação trata-se de conseguir realizar comandos na máquina através de vários canais: voz, gestos, expressões faciais. Apesar do sentido de interação multimodal ser bem amplo essa ferramenta atualmente não trabalha com voz, somente com dispositivos de câmera.

O software possui uma interface gráfica simples que pode ser vista na figura 13, nela pode-se escolher os *inputs*, qual módulo ou classe irá lidar com aquele *input* e qual será o comando executado no game. Apesar de existir uma interface gráfica, também está disponível uma SDK para C++ documentada. Possui compatibilidade com Câmeras web comuns, Kinect, *Tobii Eye Tracker*[7] e está disponível apenas para Windows.

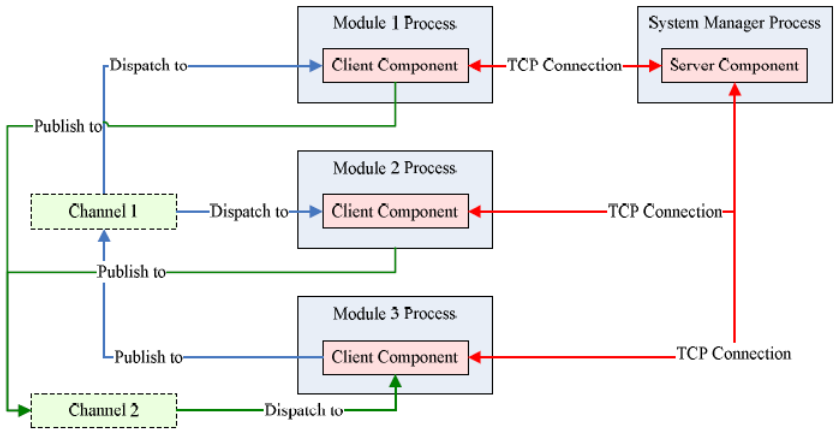
Na figura 12 autor explica que a arquitetura está dividida em duas partes, módulos e Sistema Gerenciador de Processos, os módulos irão operar diretamente com os dispositivos de interação ou com os games, utilizam a arquitetura *Publish Subscribe* (P/S), enquanto que o segundo irá apenas gerenciar a saída e entrada desses módulos.

Conforme Shen e Pantic (2009) a escolha da linguagem C++ se dá por conta da necessidade do baixo uso de recursos, item obrigatório para esse tipo de software. Com essa linguagem será possível obter um desempenho melhor em todos os aspectos. Para a parte crítica do sistema, a comunicação entre os módulos, no lugar do *Transmission Control Protocol* (TCP) foi utilizado compartilhamento de memória, apesar da dificuldade e necessidade de se implementar um protocolo só para comunicação entre processos o ganho com desempenho é 10 vezes sobre o TCP. No gerenciamento dos módulos como as mensagens são menores e em menor quantidade foi utilizado TCP pela facilidade do uso.

Shen, Shi e Pantic (2011) realizaram um teste de desempenho, comparando a ferramenta proposta com outras especializadas em troca de mensagens, como Pscylone[8] e Apache ActiveMQ[9], para avaliação de desempenho. Aproveitando a ideia do teste de

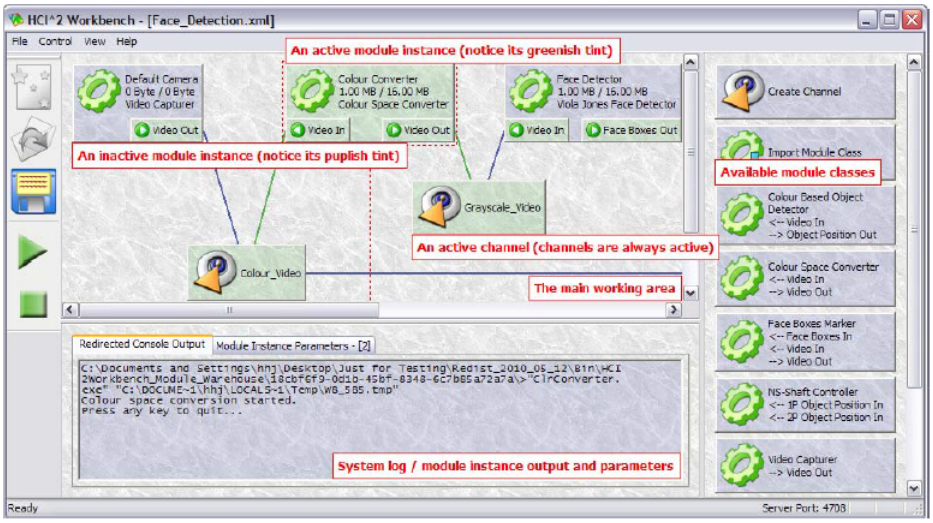
[6] TCP/IP - É um conjunto de protocolos de comunicação entre computadores em rede
[7] Tobii Eye Tracker - Hardware que lê a posição no monitor onde os olhos do usuário estão focando
[8] <http://www.cmlabs.com/psyclone/>
[9] <http://activemq.apache.org/>

Figura 12 – Arquitetura HCI^2.



Fonte: Shen e Pantic (2009)

Figura 13 – Interface gráfica HCI^2.



Fonte: Shen, Shi e Pantic (2011)

desempenho a ferramenta gerada neste trabalho foi comparada com o FAAST e testada em outras situações. Os detalhes do teste estão descritos na sessão 5.4.

Segundo os autores, além do teste de desempenho também foi realizada uma validação com um game desenvolvido para funcionar com um teclado comum, porém o HCI^2 possibilitou realizar a adaptação para ele funcionar com uma câmera de baixo custo e um software para rastreamento de objetos. O jogador move o marcador para direita ou para a esquerda fazendo com que o personagem siga a direção indicada. A interação Multimodal Humano-Computador requer dados fora do escopo desse projeto, voz e expressões faciais não serão captadas por nenhum dispositivo para reabilitação e os

gestos nem sempre serão capturados na sua íntegra, logo este projeto não se enquadra no tópico de interação Multimodal Humano-Computador.

3.6 Games sérios para reabilitação

Omelina et al. (2012) propôs uma arquitetura que permite a utilização de diversos controladores diferentes para um game. Pode-se usar uma câmera de captura de movimentos como o Kinect, e se surgir a necessidade pode-se mudar para uma câmera de outro fabricante, Sony PlayStation Eye e o game continuará funcionando.

O ponto principal dessa arquitetura é a possibilidade de configuração do modo de game pelo fisioterapeuta sem a necessidade de programação. Define-se com quais movimentos o game será controlado, configurando, por exemplo, que somente movimentos com o tronco irão mover o personagem e ele irá atacar com os movimentos de braço.

Uma das justificativas para a criação desse projeto é que, devido a existência de diversos mini games de baixo custo, os terapeutas precisam ter uma atenção maior na escolha do game ideal, uma vez que os games para reabilitação normalmente são codificados para atenderem diversos exercícios, normalmente não permitindo uma configuração do quanto que o paciente terá que se mover. Com esse pensamento, foi construída a ferramenta que permite que qualquer game possa ser controlado com o exercício desejado.

Segundo (PERRY et al., 2011 apud OMELINA et al., 2012) "games sérios precisam satisfazer necessidades especiais para serem úteis na reabilitação", com isso a surge a ideia de configurar o movimento em si, não basta o movimento ser reconhecido, mas também deve-se poder ajustar o game e o sensor para as necessidades do paciente. Deve ser possível configurar ângulo dos movimentos e desafios. Aproveitando essa ideia, as configurações do TeamBridge foram construídas para sempre que possível permitir ajustes. O terapeuta não escolhe apenas "mão fechada", ele pode definir o quanto a mão precisa estar fechada para executar a ação, da mesma forma ele pode definir vários níveis ao fechar a mão, possibilitando que quando a mão esteja fechada ao máximo outra tecla seja pressionada em conjunto, fazendo com que o personagem execute a tarefa com mais intensidade.

A ferramenta foi testada com 4 games do ICT4Rehab: *Flying Simulator*, *HitTheBoxes*, *WipeOut* e *PickThemUp*. Embora essa ferramenta atinja alguns dos objetivos propostos neste projeto, ela não possui código livre, nem disponibiliza informações úteis para sua replicação, o nome da ferramenta não foi mencionado no artigo, tampouco a lista de dispositivos compatíveis.

3.7 PlayMancer

O PlayMancer é um projeto para criação de um *framework*, com o objetivo de diminuir os custos de produções de games para reabilitação e ao mesmo tempo aumentar a velocidade de criação. Seu objetivo assemelha-se à motivação do desenvolvimento do TeamBridge, fornecer módulos para recebimento de *input* dos dispositivos controladores e interface para comunicação em rede. Usa o protocolo TCP/IP para conexão entre a aplicação cliente e servidor.

Para o PlayMancer está previsto o uso de reconhecimento de voz, imagem e até batimentos cardíacos e respiração, todavia não é foco do projeto a utilização de dispositivos para captura de movimentos (KALAPANIDAS et al., 2008).

O autor utilizou a abordagem de criar um *framework* para adaptar os games, portanto, a adaptação torna-se invasiva, é preciso modificar o código-fonte do game para sua realização.

3.8 *SmartGlove* e Kinect

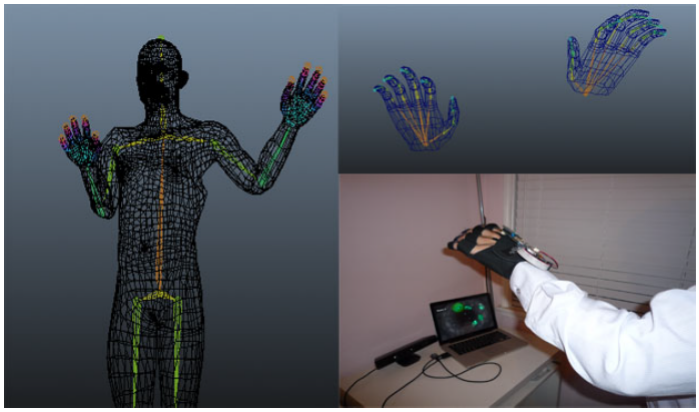
Huang et al. (2012) propôs uma luva de dados para uso na terapia. Essa luva permite o treinamento com movimento de pinça. Sua proposta se dá a partir do momento que o Kinect e outros dispositivos não são capazes de identificar com precisão os dedos da mão. Nessa pesquisa ele uniu o Kinect à luva de dados, obtendo um total de 30 pontos, desses, 20 são originais do Kinect.

Também foi relatado um problema que pode ocorrer quando se utiliza dois dispositivos para captura de dados, o sinal da Wireless tende a ser absorvido pelo corpo, quando um desses dispositivos utiliza esse meio de comunicação os dados transmitidos podem atrasar ou se perderem, causando uma falta de sincronização entre os dois dispositivos de captura podendo resultar em uma quebra ou falha na aplicação. Sua luva foi desenvolvida para re-sincronizar caso ocorra algum erro. O problema relatado também pode ser resolvido com o VRPN, os dados enviados com esse protocolo sempre são acompanhados de um *timestamp*, dessa forma, caso o dado se atrase ele não será considerado, somente quando o *timestamp* estiver no intervalo esperado a aplicação irá reconhecer esses dados, em caso de perda de conexão o VRPN reconecta automaticamente.

Nessa pesquisa também foi desenvolvida uma aplicação para visualizar os dados gravados durante a sessão de terapia. A figura 14 mostra a aplicação em funcionamento.

O autor ainda relatou que encontra dificuldade para medir o espaço entre pares de dedos ajacentes, porém esse requisito não é importante para o reconhecimento do movimento de pinça.

Figura 14 – Aplicação para visualização dos dados da sessão de terapia.



Fonte: Huang et al. (2012)

3.9 KHRD

O *Kinect-based system for ensuring home-based rehabilitation* (KHRD) foi proposto por Su (2013), o problema relatado é que os pacientes tinham dificuldade de se locomover até o local de tratamento, então passaram a realizar a terapia em casa com o Kinect, entretanto o treinamento na presença do terapeuta podia ser corrigido, enquanto que em casa essa correção não era possível. A solução proposta foi a utilização de aprendizado de máquina, mais precisamente o algoritmo *Dynamic Time Warping* (DTW) com lógica *fuzzy* para identificar movimentos que fugiam do padrão. A lógica *fuzzy* foi aplicada porque a lógica tradicional não servia para resolver tal problema, uma vez que trabalha-se com informações imprecisas, muitas vezes o terapeuta irá corrigir o paciente com frases como: "Está rápido demais", "O braço está muito alto". Seguindo essas instruções, o paciente poderá corrigir seu movimento abaixando devidamente o braço ou pode continuar errando por ter abaixado muito pouco. Diferente da lógica comum que permite apenas 0 e 1 a lógica *fuzzy* permite que existam vários valores entre 0 e 1, o paciente pode estar executando o movimento muito errado, ou apenas um pouco fora do comum.

Para essa pesquisa o paciente teve sua sessão de terapia realizada em uma clínica com supervisão do terapeuta, gravada pelo KHRD. Quando o paciente foi realizar uma sessão em casa o sistema avaliou dois fatores:

- Diferença na trajetória - O caminho percorrido por cada ponto.
- Variação na velocidade - O tempo que o paciente levou para completar cada exercício.

O resultado foi satisfatório, o KHRD conseguiu identificar 80,01% dos movimentos incorretos que o fisioterapeuta identificou.

A versão atual do TeamBridge não conta com identificação de movimentos incorretos de forma tão precisa. As únicas correções feitas pelo TeamBridge são gestos configurados pelo terapeuta, como: “levantou o braço alto demais”, “inclinou muito o corpo para o lado, podendo perder o equilíbrio”, e se o gesto está rápido demais. Caso ocorra algum movimento indesejado é possível exibir um alerta para o paciente.

3.10 GEMINI

Proposta pelo Teófilo, Nogueira e Silva (2013), a *Generic Multi-Modal Natural Interface Framework for Videogames* (GeMiNI) é um *framework* que segue o mesmo conceito de conversão de inputs adotado pelos demais trabalhos. Para seu desenvolvimento três tipos de hardware foram levados em consideração: Kinect, para detecção de poses; microfones para captura de voz e Nintendo *Nunchucks* para captura de botões auxiliares. O autor ressalta que seu diferencial é que não está restrito a um dispositivo de interação específico, todavia não foi mencionado um teste com outro dispositivo da mesma classe.

O sistema conta com gravação e reconhecimento de gestos, apesar disso nem todos os gestos são reconhecidos perfeitamente. Não há nenhuma menção no artigo quanto ao algoritmo de aprendizagem utilizado.

De acordo com os resultados, o trabalho conseguiu transformar gestos e voz para comandos no game comercial *The Elder Scrolls V: Skyrim*. A escolha desse game se deve ao complexo sistema de combate e interação.

3.11 ARTiFICe

O *Augmented Reality Framework for Distributed Collaboration* (ARTiFICe) (MOSSEL et al., 2013) é um *framework* voltado para construção de aplicações de realidade virtual ou aumentada. Em sua construção foi utilizado o protocolo VRPN para captura dos dados de *tracking*. O autor utilizou o FAAST para servir os dados de *tracking* gerados pelo Kinect. O trabalho proposto nesta dissertação também utilizou o FAAST durante o desenvolvimento.

Esse *framework* foi integrado ao Unity, ou seja, ele não gera eventos no sistema operacional, apenas facilita a integração de diversos dispositivos ao Unity. O ponto negativo dessa arquitetura é que o software não pode ser utilizado com jogos em outras game *engines*.

Nos resultados apresentados uma aplicação de RV foi jogada com o Kinect e óculos 3D enquanto outro usuário jogou a mesma aplicação com um mouse 3D em um monitor comum. Outro ponto apresentado nos resultados foi a utilização do *framework* para desenvolver uma aplicação de treinamento para pacientes com próteses, demonstrando outros fins para esses tipos de projetos.

3.12 CAC *Framework*

Esse trabalho faz parte do projeto *Unobtrusive Smart Environments for Independent Living* (USEFIL) (DEMOKRITOS, 2017). Esse projeto tem como objetivo a criação de novas tecnologias capazes de auxiliar a independência de idosos. O *Controller Application Communication* (KONSTANTINIDIS et al., 2015) teve como foco a implementação de um padrão para transportar e consumir os dados gerados por diversos dispositivos.

É compatível com: Kinect, Controles do Wii, NeuroSky MindWave, dispositivos Android e outros.

A grande vantagem desse *framework* é que ele trabalha com o conceito de serviços. Com ele é possível servir os dados gerados pelo Kinect e o Wiimote, ao mesmo tempo e ainda tornar mais fácil para aplicações em qualquer hardware consumirem esses serviços. É possível, por exemplo, consumir os dados servidos por esse *framework* dentro de uma aplicação mobile ou até mesmo uma Smart TV. Para a comunicação utiliza-se *websockets* e *web services* RESTFUL [10].

O CAC pode agir de forma semelhante ao VRPN, onde os dispositivos podem ser conectados a um ou mais servidores e esses servidores se registram em um servidor principal para fornecer os dados através deste.

Os dados são passados no formato JSON para a aplicação final. Para isso é necessário que dentro do código-fonte da aplicação, existam chamadas para se conectar ao servidor do CAC e consumir o JSON recebido para realizar ações.

A Figura 15 mostra uma aplicação Web sendo executada em uma Smart TV, consumindo dados gerados por um Kinect e WiiMote. Ao lado a mesma aplicação sendo utilizada em um *tablet*.

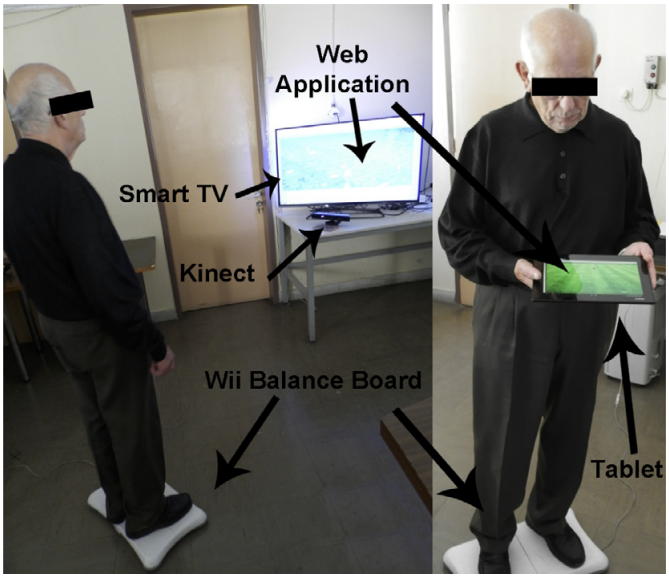
A ideia de fornecer os dados como serviço pode ser aplicada no TeamBridge, bastando a criação de um módulo com esse fim. Como não fazia parte do escopo desta dissertação, essa ideia foi registrada nos trabalhos futuros.

3.13 Outros trabalhos

Na pesquisa documental também foram encontrados outros trabalhos que utilizam o mesmo princípio de receber um input e gerar outro de teclado ou mouse. Dentre eles destaca-se o trabalho do FERREIRA (2014), onde o Kinect é utilizado para a implementação de uma lousa virtual, com o propósito de auxiliar em aulas expositivas. Gestos são realizados e a aplicação converte para entradas esperadas pela a aplicação. Em outro trabalho, o autor

[10] RESTFUL - RESTful é um sistema que utiliza o padrão REST <https://www.w3.org/2001/sw/wiki/REST> que são serviços que respondem à chamadas HTTP do tipo POST e GET comuns, entre outros.

Figura 15 – Um usuário do CAC exemplificando seu uso com diversos hardwares.



Fonte: (KONSTANTINIDIS et al., 2015).

Morales et al. (2013) apresenta o *Assistive Technology Rapid Integration & Construction Set* (AsTeRICS), um *middleware* construído com o objetivo de facilitar a acessibilidade de pessoas com diversas limitações físicas. Ele permite o uso de diversos dispositivos, inclusive de eletroencefalograma, para o uso de um computador. Seguindo a mesma linha Santos (2008) propôs uma aplicação capaz de traduzir sinais de sensores para comandos de mouse ou teclado. Esses trabalhos têm como destaque a simplificação de um dispositivo que pode ser utilizado, por exemplo, por usuários tetraplégicos, permitindo acessibilidade com aplicativos computacionais.

O Kinteract foi desenvolvido por Melo (2012), possui o mesmo princípio de converter dados do Kinect para *inputs* de teclado ou mouse, ainda assim esse trabalho está restrito ao ambiente Linux e ao dispositivo Kinect. Seu uso foi com foco em criar uma interface mais simples para idosos. O TeamBridge poderia ser utilizado para todos esses casos. Por esse motivo o módulo de compatibilidade deverá ser independente do módulo de terapia, permitindo sua utilização para outros fins.

3.14 Comparação entre os trabalhos

Com o objetivo de apresentar uma visão ampla entre os trabalhos relacionados e o TeamBridge, as similaridades entre as ferramentas foram listadas na Tabela 4, a compatibilidade com os dispositivos está listada na Tabela 5. O trabalho 3.6, games sérios para reabilitação, não está presente nas tabelas por não apresentar detalhes técnicos. O

trabalho 3.8, *SmartGlove* e Kinect, não consta nas tabelas por não ser um software para facilitar a inclusão de diversos dispositivos. Na tabela, o tópico "Não invasivo" é relativo à não exigência de modificação no código-fonte dos games. Já o tópico "Comunicação Assistida" está relacionado ao uso de alguma estrutura que lide com problemas comuns de comunicação, como falhas ou latência.

Tabela 4 – Comparativo entre as ferramentas apresentadas.

	Não invasivo	Comunicação Assistida	Persistência de dados	Monitoramento de gestos	*Open Source*
FAAST	X	VRPN	-	-	-
IGER	-	-	X	X	-
uOS *Plugin*	-	uOS	X	X	-
RehabConnex	X	-	X	-	-
HCI^2	X	-	-	-	-
PlayMancer	-	-	X	-	-
KHRD*	-	-	X	X	-
GeMiNI	X	-	-	-	X
ARTiFICe	-	VRPN	-	-	-
AsTeRICS	X	-	-	-	X
Kinteract	X	-	-	-	-
CAC Framework	-	Web Service	X	-	-
TeamBridge	X	VRPN	X	X	X

Legenda: * = Compatível com apenas um dispositivo.

Tabela 5 – Compatibilidade das ferramentas apresentadas.

Ferramenta	Dispositivos compatíveis
FAAST	KinectV1 e KinectV2
IGER	Sony PlayStation Eye, Kinect, Wii Balance Board, Omni phantom, Novint Falcon, Tyromotion Tymo e Moticon OpenGo Insoles
uOS Plugin	Dispositivo Android, teclado e Sensores IMU.
RehabConnex	Lokomat, ChARMin, PITS, RowingMachine e câmeras com software de reconhecimento
HCI 2	Câmeras WEB, Kinect e Tobii Eye Tracker.
PlayMancer	Voz, imagem, batimentos cardíacos e respiração. (Dispositivos não informados)
GeMiNi	Kinect, Microfones e Nintendo Nunchucks
ARTiFICe	Kinect e Mouse 3D
CAC Framework	Kinect e WiiMote.
TeamBridge	KinectV1, KinectV2, NEDGlove, PhysioHappy, Leap Motion, Teclado e Mouse.

4 Desenvolvimento do projeto

Este capítulo apresenta detalhes técnicos sobre o desenvolvimento do projeto, tais detalhes relacionam os interessados (*stakeholders*), requisitos funcionais e não funcionais, modelos, diagramas e padrões aplicados.

4.1 *Stakeholders*

- STK01 - Equipe do TEAM: A equipe desenvolve dispositivos de interação e games, dando andamento para a pesquisa de terapia com exergames no TEAM.
- STK02 - Equipe de terapeutas da Universidade Federal do Rio Grande do Norte (UFRN): Aliados à equipe do TEAM, os terapeutas aplicam o que foi desenvolvido com seus pacientes além de direcionar a pesquisa sobre terapia com exergames, possuem o papel de cliente do projeto.
- STK03 - Pacientes: Irão interagir diretamente com os dispositivos de interação e os games, mesmo assim para eles o atual projeto será invisível, possuem o papel de usuário final.

4.2 Requisitos

Conforme PRESMAN (2011, p. 151) as características operacionais do software são frutos da etapa de análise de requisitos, essas características estão relacionadas à interface, restrições e até o que o software deve atender. São as necessidades básicas que devem ser elaboradas durante a fase de concepção, levantamento e negociação, todas essas fases fazem parte da engenharia de requisitos.

4.2.1 Requisitos funcionais

Para Sommerville (2011, p. 73) os requisitos funcionais descrevem o que o sistema deve fazer, são os serviços que o sistema deve oferecer ou a forma dele reagir a entradas específicas, pode-se até conter detalhes do que o sistema não deve fazer.

Para alcançar os objetivos desse trabalho, foram definidos os seguintes requisitos funcionais:

- RF01 - Deverá existir um arquivo de configuração onde estarão mapeados os comandos de entrada e de saída.

- RF02 - Deverá existir um conversor que a partir dos dados lidos no arquivo de configuração, receba os inputs dos servidores e converta para os comandos desejados na máquina do cliente.
- RF03 - Deve ser possível habilitar o armazenamento dos dados gerados pelos dispositivos em um banco de dados durante a sessão.
- RF04 - Deve ser possível reproduzir uma seção com os dados armazenados.
- RF05 - Deve ser possível exibir mensagens de alerta mesmo que o game esteja sendo reproduzido em tela cheia, podendo vir acompanhado de um efeito sonoro.
- RF06 - Deverá haver uma configuração para alertar o cliente quando ele estiver com uma inclinação do corpo em um ângulo a ser configurado ou quando estiver realizando gestos muito rápidos.
- RF07 - Quando um dispositivo de *tracker* detectar uma pessoa o aplicativo cliente deverá gerar um feedback visual.

4.2.2 Requisitos não funcionais

Os requisitos não funcionais normalmente definem restrições aos serviços ou funções do sistema, podem ser restrições de *timing*, no processo de desenvolvimento e até normas. É comum que os requisitos não funcionais se apliquem a todo o sistema (SOMMERVILLE, 2011, p. 73).

A fim de descrever os requisitos não funcionais já preocupado com os riscos de cada requisito, também serão descritos os pontos de sensibilidade e pontos de conflito (*tradeoffs*) quando existirem, conforme Pinto et al. (2013) explica, existem vários métodos de avaliação de arquitetura de software baseados em cenários e atributos de qualidade, um desses é o *Architecture Tradeoff Analysis Method* (ATAM), que gera relatórios com tais pontos:

- Ponto de sensibilidade - Representa uma decisão arquitetural envolvendo um ou mais elementos arquiteturais críticos para satisfazer um atributo de qualidade. Conforme o exemplo citado: “o nível de confidencialidade em uma rede privada virtual *Virtual Private Network* (VPN) pode ser sensível ao número de bits usados para encriptação.” (PINTO et al., 2013)
- Ponto de conflito (*Tradeoff*) - São decisões arquiteturais que afetam pelo menos dois atributos de qualidade. Exemplo do artigo: “alterar o nível de encriptação poderia ter um impacto tanto na segurança quanto no desempenho do sistema.” (PINTO et al., 2013)

Os requisitos não funcionais desse projeto são:

- RNF01 - O sistema deverá utilizar o protocolo VRPN para o caso da necessidade de interligar os dispositivos via rede. **Ponto de sensibilidade**: A manutenibilidade do sistema poderá ser afetada, pois programadores não familiarizados deverão estudar o protocolo.

- RNF02 - Toda parte crítica do projeto deverá ser em C++ por conta do desempenho da linguagem. É importante que o usuário não sinta diferença ao jogar com ou sem o *middleware*, para isso o software não poderá ter baixo desempenho ao receber e converter os comandos. ***Tradeoff***: A manutenibilidade poderá ser afetada, pois a linguagem é mais complexa para programadores iniciantes. Já a eficiência é afetada positivamente, pois softwares em C++ normalmente possuem um desempenho melhor que nas demais linguagens.

- RNF03 - O sistema deverá estar versionado e documentado. **Ponto de sensibilidade**: Manutenibilidade. Um código documentado e versionado tem uma manutenção facilitada.

- RNF04 - O módulo de compatibilidade não poderá ser dependente do módulo de terapia, podendo ser compilado sem a sua presença. **Ponto de sensibilidade**: Modificabilidade. Torna-se mais fácil adaptar o *middleware* para outros fins.

- RNF05 - Devido à maioria dos games estarem disponíveis apenas para Windows o sistema poderá ser compatível apenas com este sistema operacional. ***Tradeoff***: Afeta a portabilidade, uma vez que o sistema será dependente do Windows, e a manutenibilidade, gerando retrabalho quando for necessário um refatoramento para executar em outros sistemas operacionais.

- RNF06 - Deve haver um projeto para teste unitário do código desenvolvido. **Ponto de sensibilidade**: Testabilidade, é recomendado que a cada novo gesto programado crie-se um novo caso de teste.

- RNF07 - Para obtenção de maior desempenho o registro dos dados gerados pelos dispositivos deverá ser local, os dados serão enviados para um servidor externo somente quando o utilizador desejar, dispensando a necessidade de conexão com a internet para realizar a sessão, como também há uma menor utilização da banda. **Ponto de sensibilidade**: Eficiência, o banco de dados demonstrou ser um gargalo na execução do software, não utilizá-lo irá afetar positivamente a eficiência.

- RNF08 - O projeto deverá oferecer uma classe abstrata que facilite o desenvolvimento de novas opções de ações que podem ser realizadas pelo TeamBridge. **Ponto de**

sensibilidade: Modificabilidade. Torna-se mais fácil adaptar o *middleware* para outros fins.

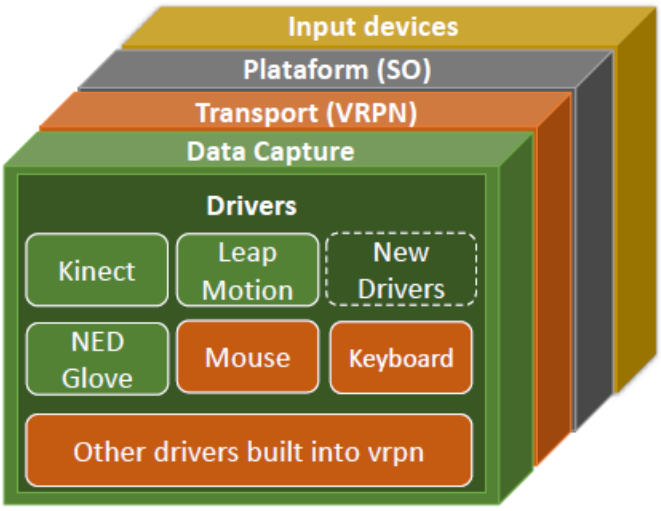
4.3 *Application programming interface* (API)

A *Application programming interface* (API) é uma interface padrão para que os desenvolvedores possam utilizar outros softwares (JACOBSON; WOODS; BRAIL, 2011). No caso do TeamBridge os desenvolvedores poderão utilizar sem a necessidade de realizar qualquer tipo de programação, basta a configuração dos gestos e inicializar o software. Porém também será possível estender o *middleware* criando novos *drivers* para outros dispositivos como também, permitir que o *middleware* realize outras ações além de chamar comandos do teclado ou mouse. Tais extensões serão abordadas nas sessões 4.6.3 e 4.6.5.

4.4 Arquitetura

A figura 16 apresenta o lado servidor. Temos a camada de transporte onde opera sob o VRPN e a Data Capture, responsável por capturar os dados dos dispositivos de interação. O lado servidor não realiza nenhum processamento em cima dos dados, apenas os recolhe e disponibiliza para o cliente. Como utilizamos o VRPN oficial alguns drivers já estavam prontos, como o do mouse e teclado. O Physiohappy utiliza o driver do mouse, para todos os demais foram criados novos drivers.

Figura 16 – Arquitetura em camadas do lado servidor.

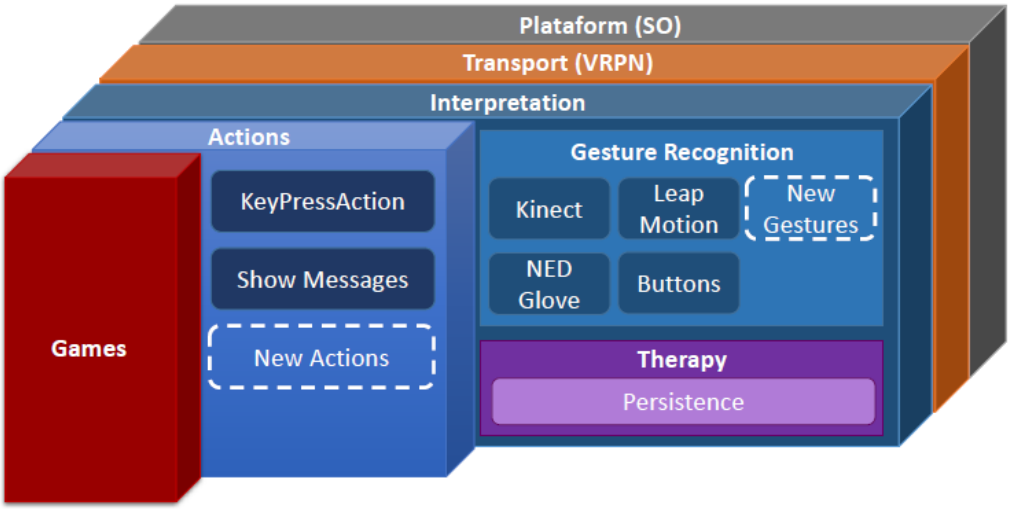


Fonte: Elaborado pelo autor.

O lado cliente demonstrado na figura 17 possui a maior parte do código. Na camada de Interpretação temos um conjunto de classes responsáveis pela interpretação dos dados

e o módulo *Therapy*, responsável por salvar todos os dados que chegam. Os dados são salvos antes de serem interpretados, devido a isso eles estão representados na camada *Interpretation*. Há a possibilidade de compilar o TeamBridge sem o módulo de terapia, isso pode ser usado quando houver a necessidade da utilização do TeamBridge para outros fins. A camada *Actions* possui as possíveis ações que o TeamBridge pode executar no sistema operacional, atualmente só existe a possibilidade de acionar uma tecla ou mostrar uma mensagem na tela, porém o *middleware* pode ser estendido para realizar outras ações como por exemplo: executar uma aplicação, acionar um comando através da rede, disponibilizar um *webservice*. Por fim, a última camada conterá o game que irá receber os inputs.

Figura 17 – Arquitetura em camadas do lado cliente.



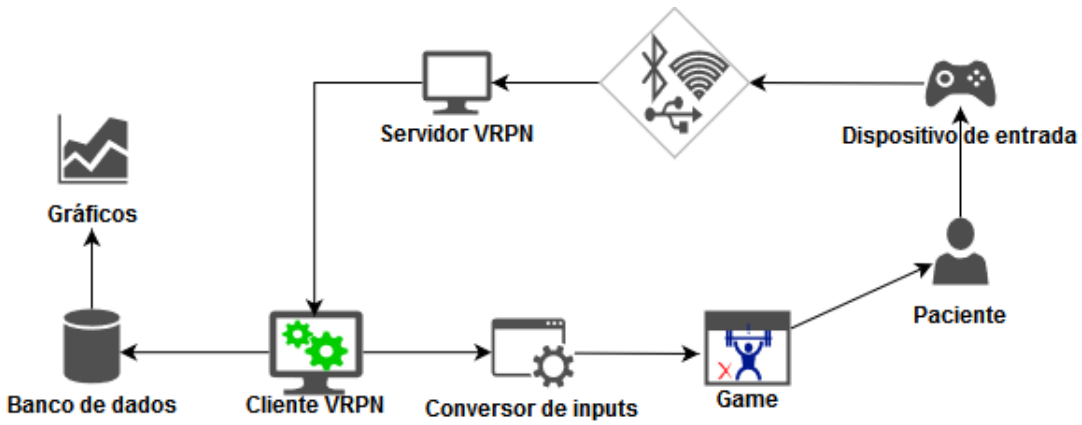
Fonte: Elaborado pelo autor.

4.5 Fluxo de dados

Como apresentado na figura 18 os dispositivos de interação estarão ligados ao servidor onde estarão os *drivers* VRPN para os dispositivos já construídos ou utilizados pelo TEAM, como o Kinect, Leap Motion e NEDGlove. Os dados serão enviados para o cliente VRPN que irá interpretá-los, gerando eventos no sistema operacional. Ele conterá uma lista de gestos que poderão ser configurados para acionamento de comandos. Além de realizar a tradução dos comandos, o cliente irá fornecer um meio para salvar o desempenho ao executar os movimentos, permitindo a análise da progressão dos pacientes ao longo das sessões.

A configuração do mapeamento consta em um arquivo de texto do tipo JSON, pode-se configurar algo como: flexionar punho aciona o botão esquerdo do mouse, enquanto que ao fechar a mão a tecla espaço é acionada.

Figura 18 – Fluxo de dados no TeamBridge.



Fonte: Elaborado pelo autor.

Caso um novo dispositivo de interação seja desenvolvido basta criar um *driver* VRPN e uma classe no lado cliente para reconhecer os dados desse dispositivo. Dependendo do dispositivo pode ser que somente o *driver* seja suficiente.

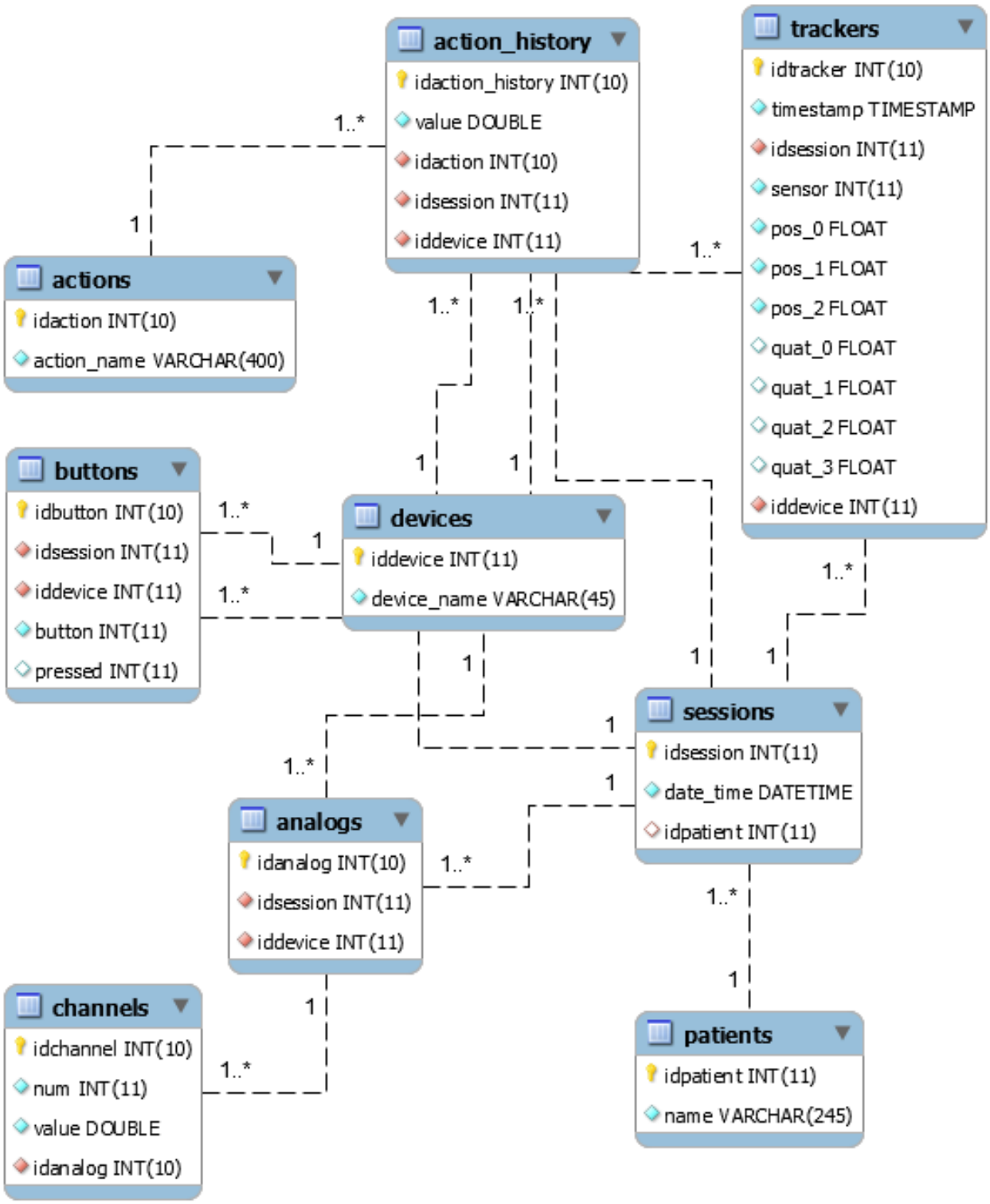
4.6 Modelos e diagramas

Ainda segundo PRESMAN (2011, p. 151) a modelagem dos requisitos resulta em alguns modelos, os modelos utilizados neste trabalho serão: modelos de dados, modelos orientados a classes, modelo de componentes e modelos orientados a fluxos. Será utilizada a UML 2.0 para todos os modelos, exceto no modelo de dados.

4.6.1 Modelos de dados

Os modelos de dados representam o domínio de informações para o problema. Para esse modelo será utilizado o diagrama de entidade e relacionamento. A figura 19 apresenta o diagrama de entidade e relacionamento, com esse diagrama é possível observar a estrutura do banco de dados do sistema. PRESMAN (2011, p. 163) explica: “O diagrama entidade-relacionamento (*entity-relationship diagram*) trata das questões e representa todos os objetos de dados introduzidos, armazenados, transformados e produzidos em uma aplicação.”.

Figura 19 – Diagrama de entidade e relacionamento do TeamBridge.



Fonte: Elaborado pelo autor.

Como esse trabalho lida com 3 tipos de dados do VRPN, é necessário uma tabela

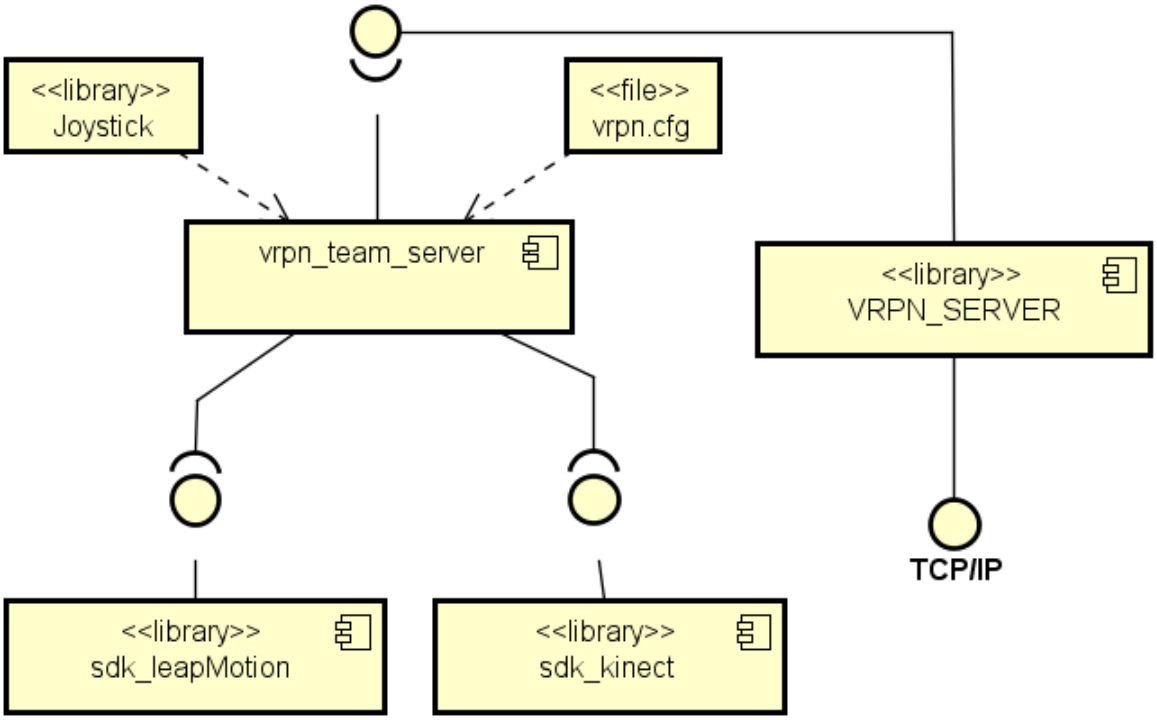
para cada tipo de dado, *trackers*, *analog* e *button*, pois cada um tem sua especificidade. As tabelas *action* e *action_history* irão guardar informações como: O ângulo máximo que o paciente conseguiu executar durante a sessão; A força máxima ao fechar a mão.

A tabela *tracker* conterá a gravação fiel da sessão, com todas as posições reportadas pelo sensor. Poderá ser usada para reprodução da sessão em um ambiente 3D. Assemelhando-se à tabela *tracker* temos a *analog* e *button* que também guardam tudo que foi reportado durante a sessão para seus respectivos tipos de dados.

4.6.2 Diagrama de componentes

Procurando dar inicialmente uma visão geral da arquitetura da solução, foi gerado um diagrama de componentes que detalha apenas alguns pontos do sistema, mostrando as conexões existentes entre os componentes.

Figura 20 – Diagrama de componentes representando o lado servidor do TeamBridge.

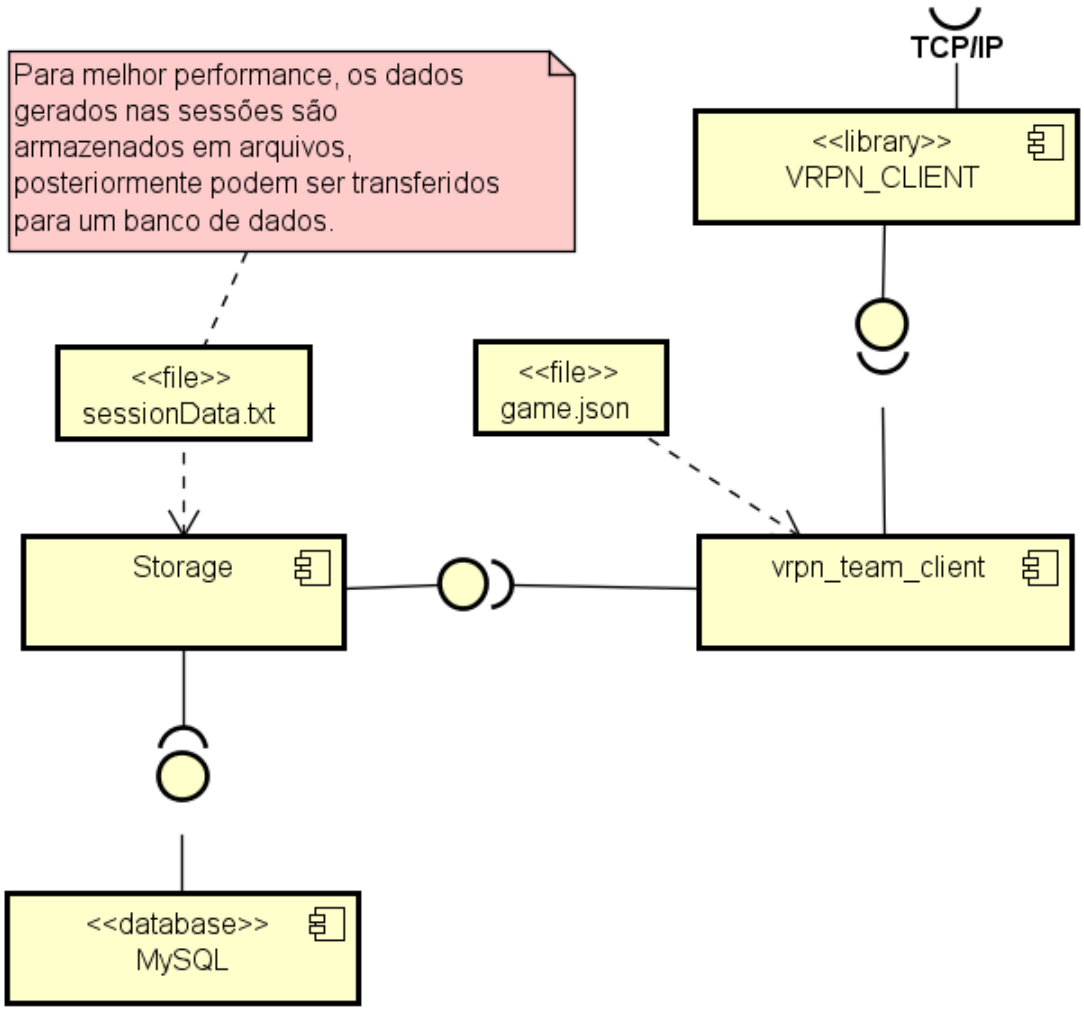


Fonte: Elaborado pelo autor.

Como demonstrado na Figura 20, o servidor irá ler um arquivo chamado vrpn.cfg, onde estarão todas as configurações do servidor. Além disso, utilizará qualquer biblioteca que seja requisito para os dispositivos físicos. Ela irá ler os dados gerados pelo dispositivos sem fazer nenhuma interpretação desses dados, em seguida irá chamar os meios nativos do VRPN para servir esses dados via TCP/IP.

Na Figura 21 o modo de conexão nativo do VRPN é utilizado. O cliente poderá se conectar a diversos servidores, possibilitando que mais de um dispositivo possa ser lido ao mesmo tempo, inclusive dispositivos do mesmo tipo sem criar conflitos. No lado cliente consta o acesso ao banco de dados para armazenar os dados da sessão, no entanto os dados são salvos em arquivo de texto por motivo de melhor desempenho. As configurações dos gestos estarão em um arquivo de extensão json.

Figura 21 – Diagrama de componentes representando o lado cliente do TeamBridge.



Fonte: Elaborado pelo autor.

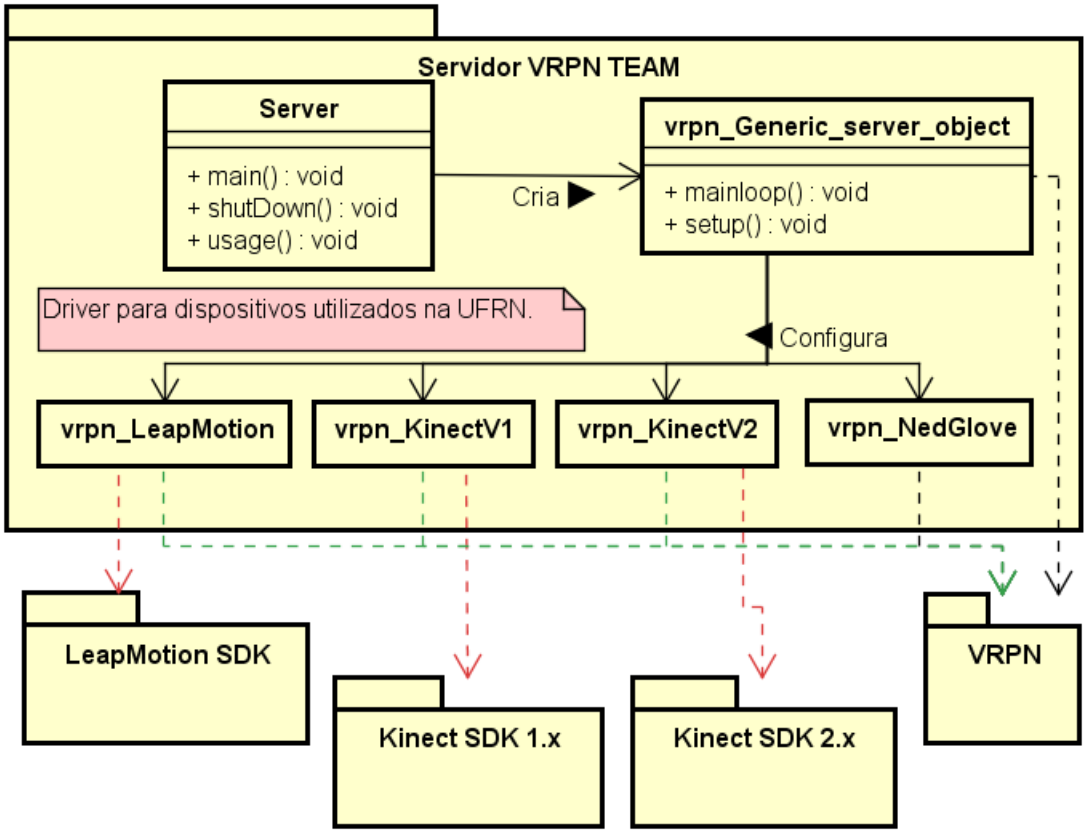
4.6.3 Diagrama de classes do lado servidor

Representam classes orientadas a objetos (atributos e operações) e a maneira por meio da qual as classes colaboram para atender aos requisitos do sistema. Sommerville (2011) explica que os diagramas de classe são usados para definir a estrutura estática de

classes em um sistema e suas associações, podendo ser expressos em diferentes níveis de detalhamento.

O diagrama apresentado na Figura 22 detalha a implementação do lado servidor do TeamBridge. O VRPN possui diversos *drivers*, para obter uma melhor organização foram apresentados apenas os *drivers* que foram desenvolvidos nessa pesquisa. Estes herdarão classes do VRPN, mas também poderão herdar ou utilizar classes de outras bibliotecas como o *driver* do Leap Motion e do Kinect que estendem e utilizam classes de suas respectivas SDKs. Como o código para utilizar o Kinect V1 é diferente do Kinect V2, foi necessária a criação de duas classes independentes, para utilizar basta configurar o vrpn.cfg e informar qual versão do Kinect será utilizada.

Figura 22 – Lado servidor do TeamBridge.



Fonte: Elaborado pelo autor.

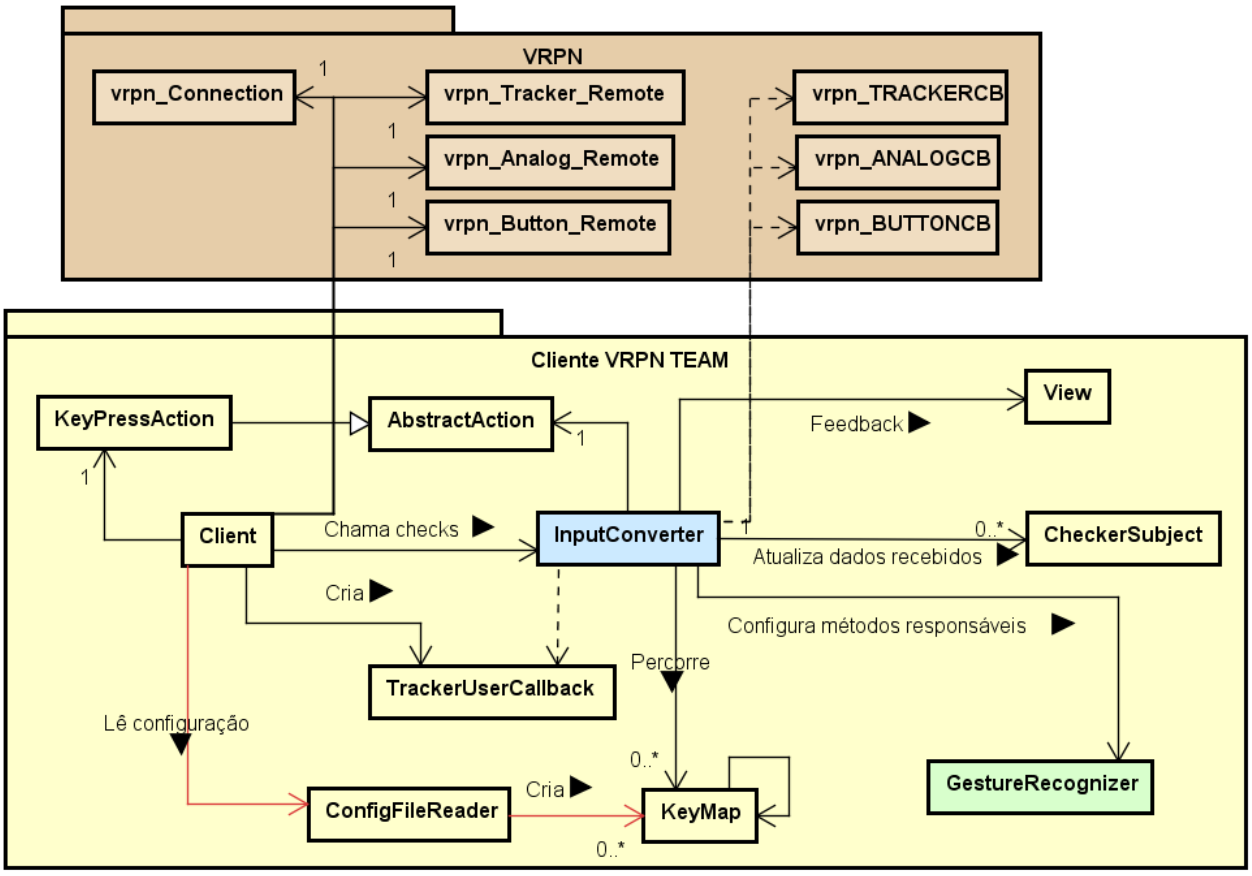
Evitando refazer algoritmos já existentes, a identificação da mão fechada tanto pelo Kinect V2 como pelo Leap Motion, foi reaproveitada diretamente das suas respectivas SDKs. Essa informação foi transformada em dados do tipo *Analog*, o driver para o Kinect V2 entrega, por exemplo: 2, 3, os valores representam respectivamente a mão esquerda e direita, o 2 significa que a mão aberta e o 3 fechada. Também pode ser encontrado os valores, 0, 1, e 4, que são respectivamente, desconhecido, não seguido e gesto de laço

(MSDN, 2014). No caso do Leap Motion os valores são mais variados pode ser, 2.5, 30, onde o primeiro é mão fechada ou aberta, quanto menor o valor mais aberta e o segundo é a distância entre o indicador e o polegar, para captura do movimento de pinça.

4.6.4 Diagrama de classes do lado cliente

Devido ao tamanho do diagrama, o lado cliente foi dividido em duas partes, como também os atributos e operações foram ocultadas. Apenas métodos essenciais para o entendimento serão apresentados nas figuras seguintes. A primeira parte apresentada na Figura 23 mostra a interação entre as classes desenvolvidas nessa pesquisa e as classes do VRPN, a conversão de comandos na classe *InputConverter* (azul) e a interpretação dos dados dos gestos na classe *GestureRecognizer* (verde) serão apresentadas em outra figura. A linha em vermelho é o primeiro passo a ser executado, a leitura do arquivo de configuração e a criação da tabela de comandos a serem convertidos. As classes no pacote VRPN (marrom) são do código oficial do VRPN.

Figura 23 – Interpretação e conversão dos comandos enviados pelo VRPN.



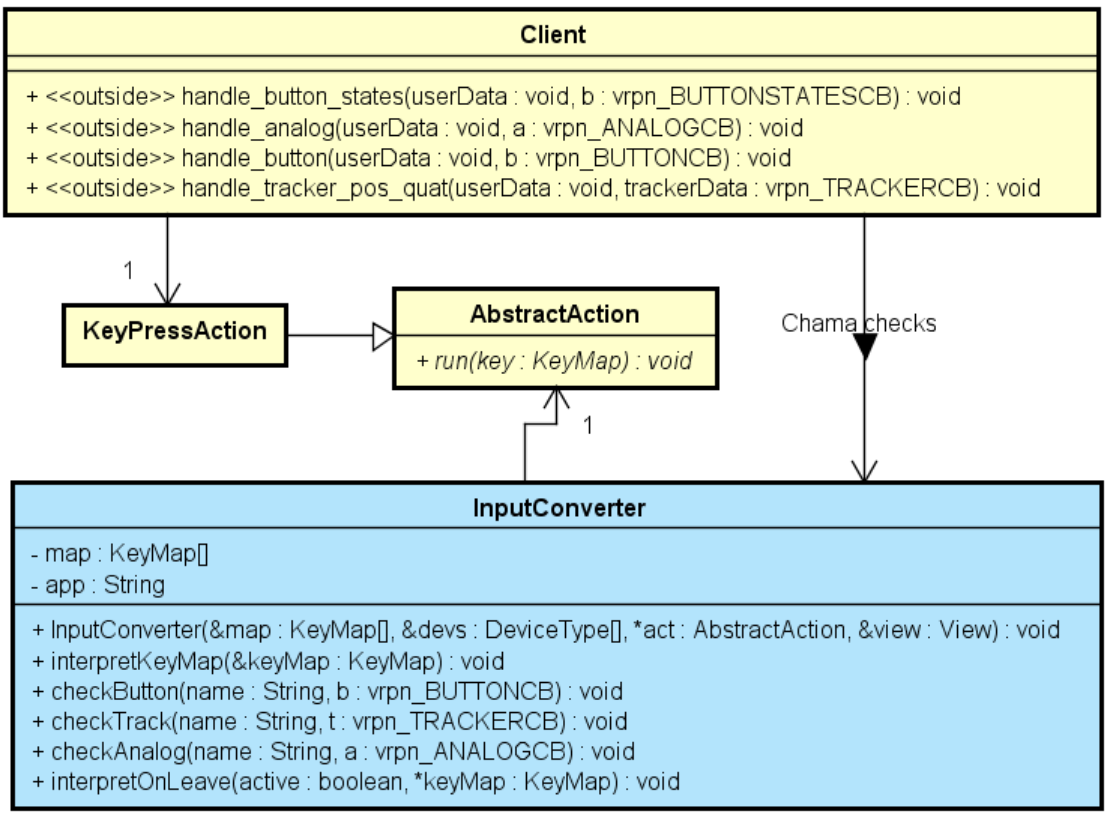
Fonte: Elaborado pelo autor.

4.6.5 Diagrama de classes do Conversor de Input

A figura 24 mostra a interação entre a classe *Client* e a *InputConverter*. Por conta da arquitetura do VRPN os métodos que recebem as mensagens VRPN não podem estar dentro de uma classe, por isso alguns métodos da classe *Client* estão com o esteriótipo *outside*, para este projeto significa que essas funções estão fora da classe, mas presentes no mesmo arquivo. Ao receber uma mensagem da rede essas funções chamam os métodos *check* da classe *InputConverter* passando a mensagem recebida.

Ainda na figura 24 é possível observar o artifício utilizado para possibilitar uma fácil adaptação deste projeto para outros fins, no sentido de acionar outros comandos no lugar de comandos de mouse ou teclado. Para isso foi criada uma classe abstrata *AbstractAction*. O *InputConverter* espera uma classe que estenda essa classe abstrata e implemente o seu método *run*. O *InputConverter* com auxílio do *GestureRecognizer* irá identificar o gesto realizado e chamar o método run da *Action*. Para esse projeto a *Action* é uma *KeyPressAction* ou seja, uma classe que possui métodos para gerar eventos de pressionar teclas. Em outros projetos a *Action* poderá acionar outro software, ou disparar algum comando pela web, basta sobrescrever o método *run*.

Figura 24 – Classes InputConverter e AbstractAction.



Fonte: Elaborado pelo autor.

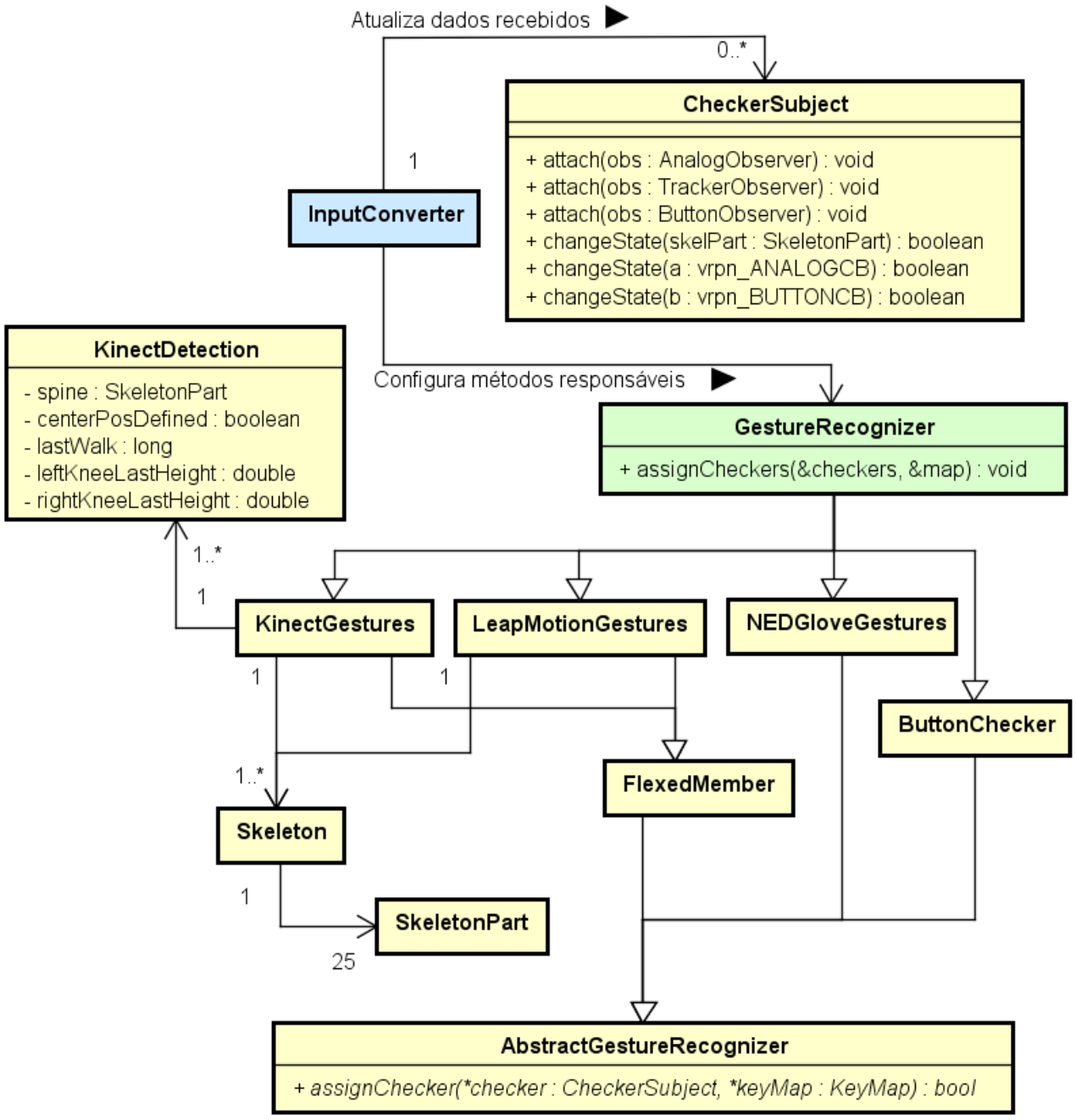
Com o intuito de obter uma maior manutenibilidade foi aplicada uma adaptação do padrão *Observer*. Neste padrão de projeto existe uma classe *Subject* que armazena as objetos que serão notificados quando houver alguma modificação. Os objetos notificados são denominados *Observers*, tais objetos devem implementar uma interface de atualização, ou seja, um método *update* que será chamado pelo *Subject* (ERICH et al., 2000, p. 274-283). Na modificação presente neste projeto, no lugar de termos várias classes observadoras foram armazenados métodos observadores, essa escolha teve como objetivo, manter todos os métodos de um dispositivo na mesma classe.

Na inicialização da classe *InputConverter* o método *GestureRecognizer.assignCheckers* é chamado. Esse método por sua vez chama o *assignChecker* de cada classe que o *GestureRecognizer* estende. Dentro destes o *attach* do *CheckerSubject* é chamado passando o método responsável por verificar aquele mapeamento.

O *GestureRecognizer* funciona como um centralizador que estende todas as classes responsáveis por reconhecer gestos, os métodos observadores estão presentes em cada uma dessas classes estendidas.

A figura 25 apresenta uma classe para cada dispositivo, cada classe possui vários métodos observadores. Nesse caso o *InputConverter* conterá um mapa com um *CheckerSubject* para cada dispositivo configurado, então quando o *InputConverter* receber uma mensagem ele chamará o *CheckerSubject.changeState* específico daquele dispositivo, este por sua vez terá 3 listas de armazenamento de métodos, uma para cada tipo de dado enviado, essa separação permite que os métodos observem apenas o "assunto" desejado, isso é necessário pois alguns dispositivos possuem métodos que observam "assuntos" distintos. A aplicação desse padrão reduziu significativamente o tamanho da classe *InputConverter* que passou algo em torno de 480 linhas para 130, além de deixar a responsabilidade de configuração dos métodos responsáveis pelos gestos nas próprias classes dos dispositivos, por exemplo, dentro da classe Kinect há a configuração do método responsável por verificar se o braço está levantado, antes dessa modificação todas essas configurações ficavam no *InputConverter*.

Figura 25 – Padrão *Observer* no *CheckerSubject*.

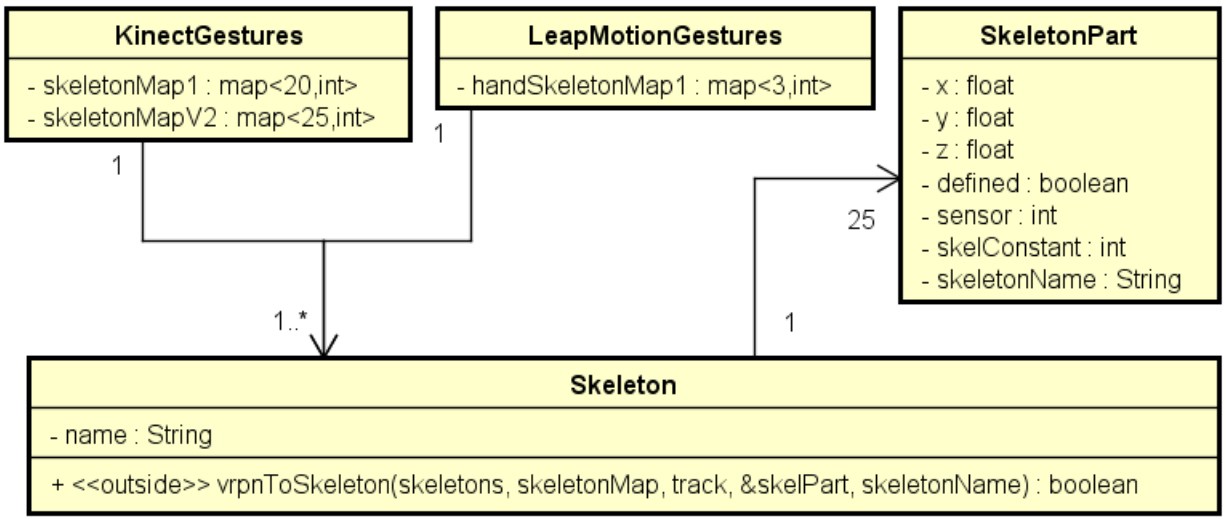


Fonte: Elaborado pelo autor.

Quando os dados de *tracking* são passados de uma SDK para o VRPN, a sua organização se perde, um dado que a SDK fornece pelo nome, por exemplo, *leftHand*, torna-se um número. O número utilizado depende da quantidade e disposição de pontos no dispositivo. Com a classe *Skeleton* foi possível criar um padrão para reorganizar e identificar esses dados de *tracking*. Cada classe responsável por identificação de gestos, com tipo de dado *tracking*, contém um mapa onde são armazenados os *Skeletons* do

dispositivo. Dessa forma cada endereço contém um único mapa, impossibilitando que um dispositivo influencie em outro. Por exemplo: Tracker0@localhost terá um *Skeleton* dentro de *KinectGestures*, Traker1@localhost também é do Kinect, mas é uma segunda pessoa sendo capturada, sendo assim, esta terá direito a outro *Skeleton*. Ainda para facilitar a organização, cada *Skeleton* pode conter até 25 partes menores, que são as *SkeletonPart*: Cabeça, pescoço, mão esquerda, polegar esquerdo. Para utilizar essa estrutura, todas as classes que trabalham com *Skeleton* terão um mapa indicando qual sensor é responsável por cada *SkeletonPart*. Com isso é possível trabalhar dentro das classes de reconhecimento com o nome do *SkeletonPart* e não com o número do sensor, que pode mudar de dispositivo para dispositivo, é possível observar isso no trecho de código 4.1. A função vrpnToSkeleton é responsável por ler o dado de *tracking* do VRPN e atualizar o *Skeleton* ao qual ela pertence, basta passar o mapa correto para essa função. Essa estrutura está apresentada na figura 26.

Figura 26 – Classe *Skeleton*.



Fonte: Elaborado pelo autor.

Com essa tática é possível usar a mesma classe de reconhecimento de gestos, para reconhecer dados de dois dispositivos diferentes, bastando criar um novo mapa para associar cada sensor à sua respectiva parte do esqueleto. Não é preciso preencher todas as partes do esqueleto no mapa, basta aquelas que serão utilizadas. O Leap Motion, por exemplo, utiliza apenas cotovelo, punho e mão para identificar o gesto de flexionar punho.

O trecho de código 4.1 mostra uma comparação dentro do *detectRightHandTop* para identificar se aquele *SkeletonPart* é o esperado para iniciar a verificação do gesto, caso não seja ele descarta e retorna -1 para informar que foi descartado. No método *detectHandXPos* ele usa o *hipCenter* do *Skeleton* que possui o mesmo nome do *SkeletonPart* que ele recebeu,

com isso é garantido que um *Skeleton* não influenciará em outro, e torna-se possível restaurar uma estrutura semelhante à da SDK antes de ser transformada para o VRPN.

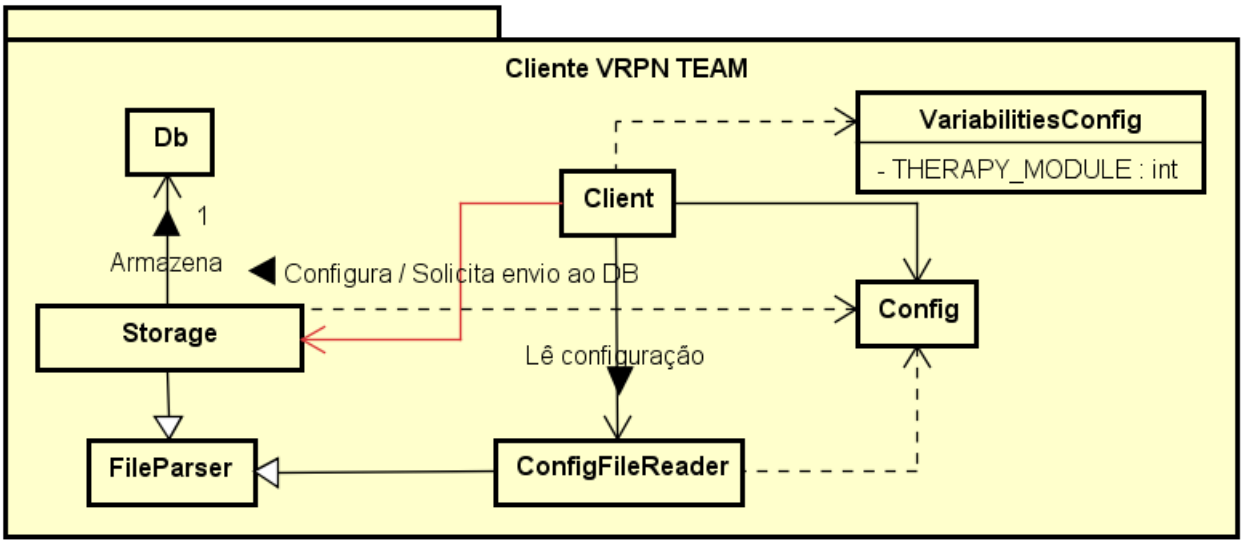
Código 4.1 – Realizando comparação com o SkeletonPart

```
int KinectGestures::detectRightHandTop(SkeletonPart skelPart, KeyMap * keyMap) {
        if ( skelPart.skelConstant == SKELETON_HAND_R ) {
                return detectHandXPos(skelPart, keyMap);
        }
        return -1;
}

int KinectGestures::detectHandXPos(SkeletonPart skelPart, KeyMap * keyMap) {
        if ( skeleton[skelPart.skeletonName].hipCenter.defined == false ) {
                return -1;
        }
}
```

A segunda parte do lado cliente constante na Figura 27 exibe a persistência dos dados gerados. Como essa funcionalidade pertence ao módulo de terapia, o sistema poderá ser compilado sem este código. Essa variabilidade está realizada através de uma constante no preprocessador do C++, constando na classe *VariabilitiesConfig*. Essa classe é apenas um arquivo com a constante do preprocessador definida ou comentada, habilitando ou desabilitando essa funcionalidade.

Figura 27 – Classes de persistência do cliente do TeamBridge.



Fonte: Elaborado pelo autor.

Uma vez que o código esteja compilado para utilizar essa funcionalidade a classe *Client* poderá solicitar que o *Storage* salve as mensagens VRPN recebidas. O *Storage* por sua vez irá salvar em arquivo de texto por motivo de melhor desempenho, uma vez que em um pequeno teste com o Kinect foram geradas mais de 9 mil linhas por segundo.

Quando o utilizador solicitar, o *Storage* poderá realizar a transferência dos dados dos arquivos de texto para o banco.

Os apêndices A e B mostram as figuras 45 e 44 com os detalhes das classes do projeto.

Esse trabalho foi produzido na *Integrated Development Environment* (IDE) Microsoft Visual Studio 2017, com Microsoft Visual C++ 2017, não é garantida a compatibilidade do projeto com IDEs mais antigas, isso se deve à utilização de bibliotecas disponíveis apenas na versão 2017 do Microsoft Visual C++.

5 Resultados

Neste capítulo serão apresentadas as ferramentas criadas durante este trabalho, bem como as validações e testes realizados.

Todo código produzido foi versionado através do Git[1] e enviado para o GitHub, estando disponível como um projeto público, possibilitando que outras pessoas possam baixar o código e realizar modificações.

5.1 Editor de configuração

Para configurar os gestos que serão convertidos para *inputs* é utilizado um arquivo JSON. O trecho de código 5.1 mostra uma simples configuração do gesto de levantar a mão acima da cabeça:

Código 5.1 – Exemplo de configuração JSON

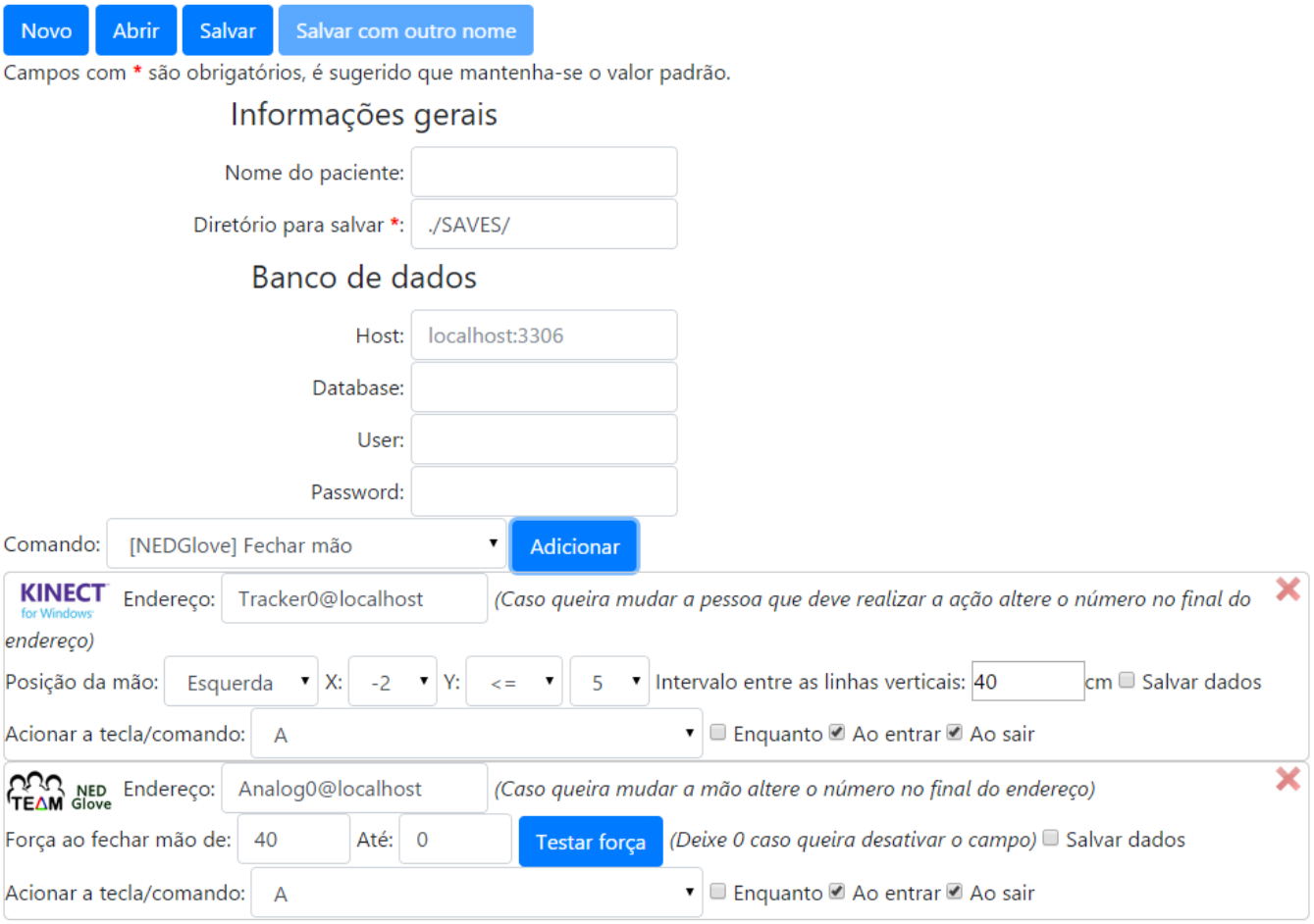
```
{
  "common": {
    "patientName": "",
    "saveDir": "./SAVES/",
    "host": "",
    "database": "",
    "user": "",
    "passwd": ""
  },
  "keys": [{"divClass": "handTop",
      "devType": "kinect",
      "dev": "Tracker0@localhost",
      "key": "KINECT_RIGHT_HAND_TOP",
      "x": 0,
      "coordinateMod": "<=",
      "y": 4,
      "handWidthInterval": 0.4,
      "toKeyDown": "A",
      "toKeyUp": "A"}]
}
```

A parte “common” possui configurações gerais como banco de dados e paciente. Seu preenchimento só é necessário quando for preciso exportar as informações para o banco de dados. O bloco de “keys” conterá as várias configurações possíveis.

[1] Git - Sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte.

Para facilitar a criação ou edição desses arquivos um pequeno projeto foi desenvolvido. O Node Webkit [2] v0.11.5 foi escolhido para esse projeto, por trabalhar com arquivos HTML e JS, permitindo programar facilmente uma interface flexível que se adapta à quantidade de comandos configurados, a figura 28 mostra a interface da aplicação. É importante observar que para o desenvolvimento desse projeto não houve condições para a variabilidade do módulo terapêutico, ou seja o JSON gerado por essa aplicação sempre conterá propriedades relativas ao módulo de terapia, ainda assim essas propriedades simplesmente serão ignoradas, caso o cliente do TeamBridge tenha sido compilado sem esse módulo.

Figura 28 – Editor de configuração do TeamBridge.



Fonte: Elaborado pelo autor.

Nas informações gerais só existe um campo obrigatório, “Diretório para salvar”, é onde serão guardados os arquivos txt que serão gerados durante a sessão, é recomendável manter o local que já vem preenchido, porém, pode ser modificado para qualquer outro.

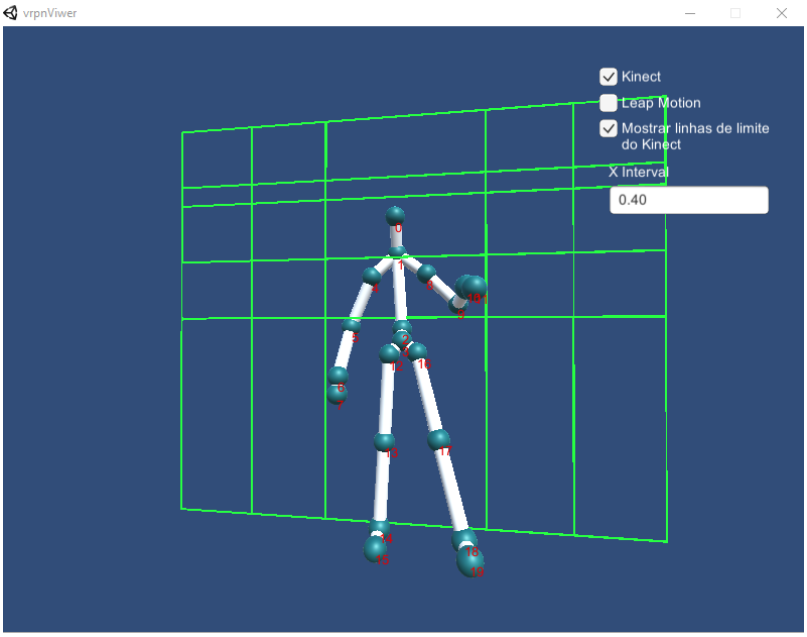
[2] Permite a criação de aplicativos desktop com *HyperText Markup Language* (HTML) e JS https://nwjs.io/

Em seguida vem as informações de banco de dados. Só será necessário preencher estas informações quando houver necessidade de exportar os arquivos txt para o banco de dados.

Para adicionar um novo gesto, basta escolhê-lo na caixa “Comando” e clicar em “Adicionar”, uma nova caixa será adicionada somente para a configuração desse gesto. Cada gesto pode conter configurações diferentes. Na figura 28 existe a configuração para posição da mão com o Kinect e Fechar mão com a NEDGlove. No caso da NEDGlove e outros como o Leap Motion, há configurações como “força ao fechar a mão de X até Y”, isso permite a adaptação da terapia para a necessidade do paciente, permitindo que a luva aceite uma força menor do que o habitual, como também permite que caso o usuário consiga fechar mais do que o normal a luva acione outro comando.

5.2 VRPN *Viewer* - Visualizador 3D

Figura 29 – Visualizador 3D dos dados transmitidos via VRPN.



Fonte: Elaborado pelo autor.

Também como um projeto secundário foi criado com a Unity 3D na versão 5.5.0f3[3] o VRPN *Viewer*, tem um objetivo específico: exibir os pontos que estão sendo enviados através do VRPN em um ambiente 3D. Com isso é possível visualizar uma sessão que foi salva em arquivo ou visualizar em tempo real o que está sendo transmitido através do VRPN. A figura 29 mostra a sua utilização exibindo dados de um Kinect, todavia ela também pode ser utilizada para apresentar a visualização do Leap Motion. As linhas

[3] Unity 3D - *Game Engine* para desenvolvimento de games <https://unity3d.com/>

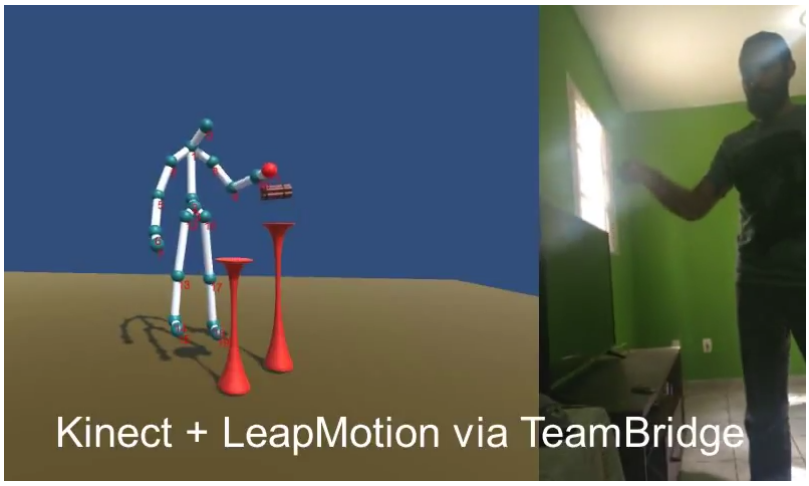
verdes na figura 29 são as áreas que podem ser configuradas para acionamento de teclas. Essas linhas são exibidas apenas para demonstração, no editor de configuração há uma explicação mais detalhada sobre como configurar.

Esse aplicativo não faz a leitura direta do arquivo de texto onde estão salvas as sessões, para isso existe um subprojeto na solução do Visual Studio denominado VRPN *Simulator* que é capaz de fazer a leitura de um arquivo de texto e simular a utilização de um dispositivo de *tracking* via VRPN.

Essa aplicação também representa a possibilidade da utilização dos dados gerados pelo VRPN dentro dos games. Através do projeto UnityVRPN[4] é possível facilmente ler as mensagens geradas pelo VRPN, isso possibilita games que utilizem a pose real do jogador, ainda assim sem a necessidade de código para verificar se ele está com o braço levantado ou até mesmo a captura da mão fechada através de outro dispositivo, uma vez que todo esse trabalho pode ser feito pelo TeamBridge.

A figura 30 é parte de um vídeo[5] que mostra o Kinect sendo utilizado junto com o Leap Motion, em uma aplicação Unity modificada a partir do VRPN *Viewer*. O TeamBridge identifica se a mão está fechada, acionando uma tecla. Quando a tecla for pressionada a aplicação Unity deixa a mão vermelha e faz com que o objeto que está sendo "segurado" acompanhe a posição da mão.

Figura 30 – União do Kinect com Leap Motion via TeamBridge



Fonte: Elaborado pelo autor.

A união de dispositivos é interessante pois um pode suprir as deficiências do outro, como mostra o trabalho de Huang et al. (2012), onde uma luva foi utilizada para suprir a

[4] <https://github.com/Laremere/unityVRPN>
[5] União do Kinect com Leap Motion via TeamBridge - <https://www.youtube.com/watch?v=7x8uCQZMkXA>

deficiência do Kinect V1, que não consegue capturar as mãos. Outro ponto relevante é que o desenvolvimento de games baseados no VRPN pode manter a compatibilidade dos jogos com os novos dispositivos, uma vez que se faça um game para Kinect, porém baseado em VRPN ele funcionará com qualquer dispositivo semelhante ao Kinect que possua driver VRPN.

5.3 Games compatíveis

Essa sessão irá mostrar a execução do TeamBridge com um game comercial um exergame do TEAM, algumas adaptações que foram realizadas em um exergame já produzido e os detalhes de outro exergame que ainda está em desenvolvimento pelo TEAM.

5.3.1 Mount and Blade

Figura 31 – Mount and Blade sendo jogado com o Kinect através do TeamBridge

Fonte: Elaborado pelo autor.

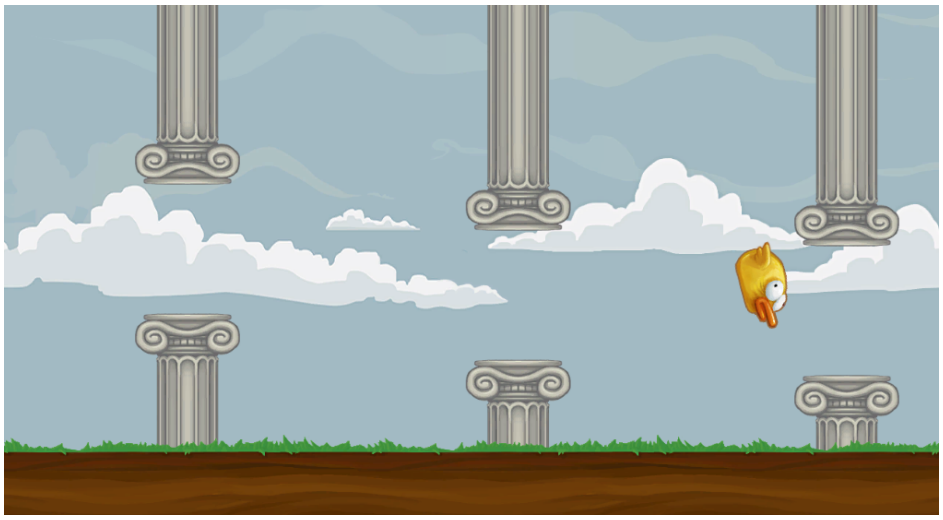
Para comprovar que não há necessidade de alterar o código-fonte dos games para utilização com o TeamBridge, foi realizado um teste com o game comercial Mount and Blade. Esse game foi escolhido por não ter nenhuma compatibilidade com o Kinect ou qualquer outro dispositivo de RV, além de permitir a execução de vários comandos diferenciados, como por exemplo: ataque pela direita, ataque pela esquerda, ataque por cima, defesa com escudo, andar e outros. Com esses comandos foi possível fazer uma analogia aos gestos realizados com o Kinect. O TeamBridge consegue capturar a marcha estacionária, logo esse gesto foi utilizado para fazer o personagem andar. Ao girar o corpo

para o lado o personagem também irá girar. Para o comando de defender com o escudo, basta levantar a mão esquerda na altura do peito. Os comandos de ataque requerem que o jogador posicione a mão direita do lado esquerdo, direito ou em cima da cabeça, para definir a direção do golpe, em seguida o jogador deve levar a mão direita à sua frente na altura do peito, fazendo com que o golpe seja lançado. A figura 31 mostra o game sendo jogado com o TeamBridge, essa figura é parte de um vídeo[6].

5.3.2 Fat Bird

O exergame Fat Bird, demonstrado na figura 32 foi desenvolvido pelo TEAM e já possuía compatibilidade com o Leap Motion e o PhysioHappy. Com o PhysioHappy o game pode ser jogado flexionando o punho para baixo ou para cima, no Leap Motion só é possível jogá-lo abrindo e fechando a mão. No entanto através do TeamBridge também foi possível jogá-lo com o Kinect flexionando o punho, no caso foi configurado para flexionar para baixo, bem como também foi possível utilizar o Leap Motion, também configurando para flexão de punho. É importante ressaltar que a flexão de punho não é identificada naturalmente pela SDK do Leap Motion tampouco pela do Kinect, essa configuração só é possível através do TeamBridge.

Figura 32 – Fat Bird



Fonte: Elaborado pelo autor.

[6] Jogando Mount and Blade através do TeamBridge - <https://www.youtube.com/watch?v=hQgXh6EPxDw>

5.3.3 Kayak Supremo

Figura 33 – Kayak Supremo

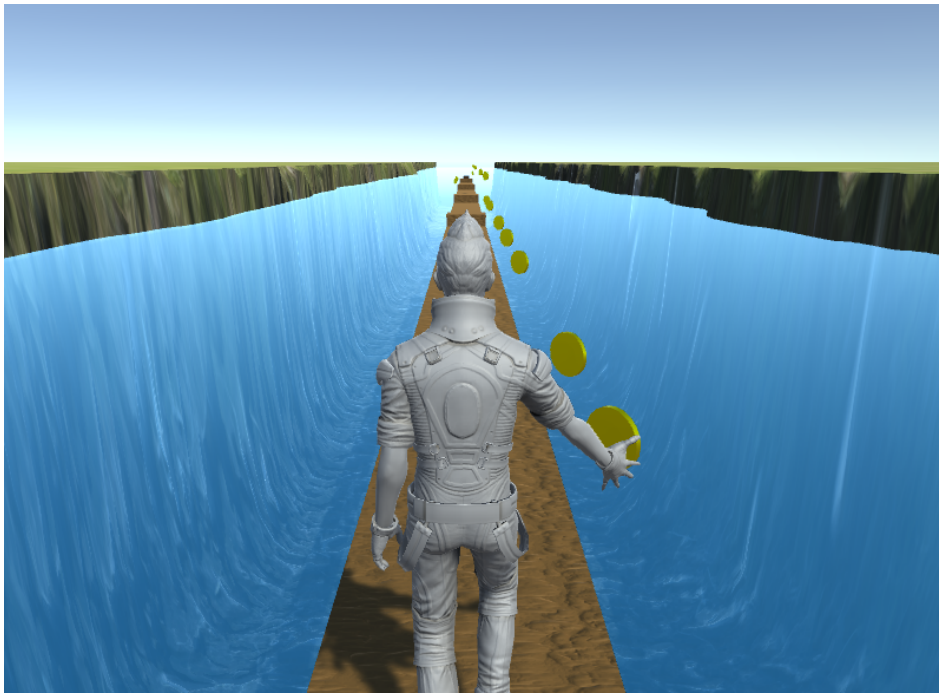
Fonte: SBGames (2013).

O Kayak Supremo (SBGames, 2013) demonstrado na figura 33 teve o seu código fonte modificado para ser compatível com o TeamBridge. Esse game foi desenvolvido pelo Team e só era possível ser jogado com a NEDGlove. A adaptação consistiu apenas em permitir entradas de teclado. Foram criados 4 comandos com as teclas Q, A, E e D. onde Q e A levam o barco para a esquerda e E e D levam para a direita, porém A e D fazem o barco ir mais rápido para o lado. Essa diferença de velocidade é para interpretar quando a luva está pouco ou muito fechada. Os gestos capturados pelo TeamBridge foram construídos para serem configurados com faixa de valores. Como o exemplo deste caso a NEDGlove foi configurada para acionar a tecla A caso a força ao fechar a mão seja de 30 até 50. O editor de configuração também permite o teste da força para que a pessoa que esteja configurando saiba qual faixa inserir, basta conectar a luva e clicar no botão “testar força”, com isso a força capturada pela luva será exibida na tela.

5.3.4 VirtuAlter

O VirtuAlter exibido na figura 34 está sendo desenvolvido pela equipe do TEAM para ser jogado com o Kinect, faz parte de um projeto de doutorado em andamento na UFRN que envolve reabilitação com exercícios de equilíbrio para idosos. Apesar de

Figura 34 – VirtuAlter



Fonte: Elaborado pelo autor.

estar sendo construído para o Kinect não possui nenhum código específico para esse hardware, sequer é necessário o driver para o Kinect na máquina que irá rodar o game, apenas na máquina que terá o servidor do TeamBridge. Dessa forma esse game pode ser desenvolvido sem a necessidade do desenvolvedor conhecer qualquer outra SDK externa ao Unity, possibilitando que ele foque apenas no desenvolvimento do game.

Atualmente é possível jogá-lo apenas com teclado ou com o Kinect V1 e Kinect V2 através do TeamBridge. Com esse tipo de arquitetura será possível modificar o TeamBridge para funcionar com qualquer sensor semelhante ao Kinect como o PrimeSensor ou SonyPlaystation Eye e o VirtuAlter automaticamente já estará compatível com tais sensores através do TeamBridge.

Esse game tem uma especificidade em relação aos demais, o jogador terá que subir e descer de degraus físicos, isso fará com que o personagem suba ou desça dos degraus que eventualmente aparecerão no game. Espera-se que seja montada uma sala onde haverá de 1 a 3 degraus para o paciente interagir, onde ele deverá subir e descer o primeiro degrau e já ficar posicionado para o próximo.

Até o momento esse gesto só pode ser reconhecido com o Kinect, porém futuramente o hardware poderá ser substituído inteiramente por um ou vários dispositivos mais simples, sua compatibilidade poderá ser garantida com o TeamBridge.

Durante os testes com o Kinect foram observadas algumas limitações, eventualmente o dispositivo não capturava os dados corretamente quando o jogador descia ou subia o degrau. Ao andar na frente do Kinect foi identificado que a altura do usuário mudava conforme se aproximava ou distanciava do Kinect e isso atrapalharia o TeamBridge a identificar quando o usuário sobe ou desce o degrau. Diante desse problema tentou-se identificar uma região onde essa distorção fosse mínima.

Para identificar a região ideal foi montado um teste onde o Kinect foi posicionado em uma mesa com 75cm de altura, em um primeiro momento a câmera foi ajustada para ficar completamente nivelada e repetiu-se o teste aumentando a inclinação da câmera em aproximadamente 4°. O Kinect V1 possui opção via software para essa configuração, o Kinect V2 precisa ser configurado manualmente.

Para obtenção dos valores da altura o TeamBridge foi configurado para imprimir constantemente a altura capturada no comando de "degrau", com isso ele imprime a altura do ponto acima do peito do usuário. Esse teste também pode ser feito sem o TeamBridge, basta imprimir o eixo Y do ponto *Shoulder center*. É importante observar que anteriormente o TeamBridge utilizava o ponto da cabeça do usuário, porém quando testado o ponto do *Shoulder center* a área útil aumentou e a chance do ponto continuar na visão da câmera quando em cima do degrau é maior, pontos muito a baixo como *Hip center* apresentaram uma área pior do que quando o ponto era na cabeça.

Em todos os testes procurou-se uma área onde a variação da altura fosse de no máximo 0,5, ou seja procurava-se a posição onde a altura indicada fosse maior, por exemplo: 70,0 e o usuário andava para frente até encontrar uma posição onde a altura fosse de no mínimo 65,0 em seguida o usuário andou para trás até encontrar o mesmo valor de 65,0, isso porque notou-se que quando o usuário se aproximava muito do dispositivo a altura diminuía drasticamente, o mesmo acontecia se ele se afastasse demais. As áreas onde a variação ficou em torno de 0,5 foram marcadas e medidas, a tabela 6 apresenta todas as medições realizadas.

Tabela 6 – Medições da região ideal para o comando de degrau com o Kinect

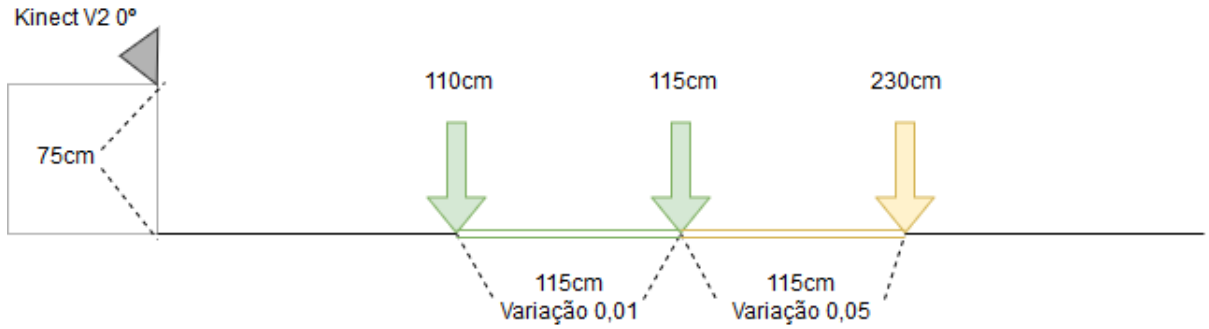
Versão	Ângulo	Ponto inicial	Área
Kinect V1	0°	125cm	66cm
Kinect V1	4°	110cm	150cm
Kinect V1	8°	110cm	115cm
Kinect V2	0°	110cm	230cm
Kinect V2	4°	110cm	80cm

De acordo com resultados apresentados na tabela 6, para o Kinect V1 a melhor configuração foi com o valor de 4°, conseguindo uma área de 150cm, já com o Kinect V2 a melhor configuração foi ele completamente nivelado, conseguindo uma área de 230cm. Ainda foi observado no Kinect V2 que nos primeiros 115cm a altura tem uma variação de

no máximo 0,01, nos próximos 115cm a variação chega a 0,05, essa primeira área pode ser utilizada para algum gesto que requeira uma maior precisão. A figura 35 ilustra as medidas encontradas na configuração ideal do Kinect V2. Com o Kinect V1 não foi identificada uma área com tão pouca variação.

Com essas informações é possível utilizar um degrau que possua 11,5cm de altura, essa altura será lida pelo Kinect como um valor em torno de 0.12. Sendo assim deve-se configurar o TeamBridge com a sensibilidade de 0.07, já que quando o usuário subir o degrau o valor mínimo na região ideal será a altura do degrau 0.12 menos a variação máxima 0.05.

Figura 35 – Medidas do teste da região ideal com o Kinect V2



Fonte: Elaborado pelo autor.

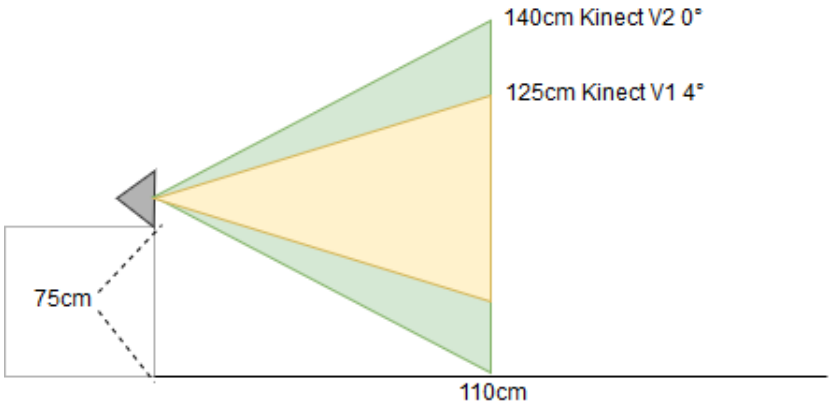
Apesar do Kinect reconhecer pessoas em uma área a partir de 80cm até 4m, segundo segundo a própria Microsoft (MSDN, 2012) existe uma região recomendável que vai de 120cm até 3.5m. Entretanto conforme as medidas tiradas nesse teste a área útil para um gesto que requer bastante precisão é bem menor.

Como a região ideal fica a 110cm do Kinect e tem 230cm, só é possível utilizar satisfatoriamente 3 degraus apesar de possuírem 30cm de largura. Espera-se que o usuário tenha um espaço adequado de 30cm para subir e mais 30cm para descer, sendo assim um degrau sozinho ocupará 90cm, dois degraus posicionados um após o outro ocuparão 150cm já que o espaço para descer de um degrau poderá ser o mesmo espaço para subir no próximo, finalmente 3 degraus em sequência irão demandar 210cm. Com o Kinect V1 a maior área útil foi de 150cm, possibilitando o uso de no máximo 2 degraus. Uma alternativa é utilizar somente um degrau onde o usuário deverá subir de frente e descer de costas.

Também foi observado outro problema, quando o usuário está em cima de um degrau muito próximo ao dispositivo porém ainda na área ideal é possível que o ponto do peito do usuário saia da visão do dispositivo, isso causará flutuações na captura da altura. Devido a isso foram medidas a altura máxima que ficará dentro da visão do dispositivo

aos 110cm. No Kinect V1 ajustado com 4° essa altura é de 125cm no Kinect V2 com 0° essa altura é de 140cm, a figura 36 ilustra essas medições. Logo a amplitude da câmera do Kinect V2 é melhor para esse fim. Apesar de ser possível utilizar o Kinect V1 para esse projeto recomenda-se o Kinect V2 por conta da sua precisão.

Figura 36 – Amplitude da visão do Kinect



Fonte: Elaborado pelo autor.

5.4 Testes de desempenho

Segundo Meier et al. (2007) o teste de desempenho tem como objetivo determinar a capacidade de resposta (*responsiveness*), a taxa de transferência (*throughput*), a confiabilidade (*reliability*) e/ou escalabilidade (*scalability*) de um sistema em uma determinada carga de trabalho. Ainda conforme Meier et al. (2007) explica, normalmente é feito para identificar gargalos em um sistema, estabelecer uma *baseline*[7] para testes futuros, fornecer informações que ajudem na melhoria de desempenho, determinar a conformidade com os objetivos e requisitos de desempenho e/ou ajudar os *stakeholders* a tomarem decisões relacionadas à qualidade geral da aplicação.

Existem dois tipos de teste de desempenho, testes de carga (*Load testing*) voltado para validar as características de desempenho em situação de produção; e testes de estresse (*Stress testing*), onde o sistema é testado em condições além das previstas em produção como: memória limitada, espaço em disco insuficiente ou falha no servidor. (MEIER et al., 2007)

Para este trabalho será realizado um teste de carga com os seguintes objetivos:

- Comparação com outro software semelhante, no caso o FAAST.

[7] *Baseline* - Dados de referência para testes futuros.

- Validar o TeamBridge para uso, em relação ao tempo de resposta para ler um gesto do dispositivo e traduzir para um comando de teclado ou mouse.
- Comparar o desempenho dos dispositivos em diversos cenários.
- Identificar gargalos na leitura ou interpretação dos dispositivos.

Meier et al. (2007) enumera as atividades necessárias para executar um teste de desempenho:

- **Atividade 1. Identificar o ambiente de teste** - Identificar hardware, software no ambiente de teste e/ou em produção. O software proposto deve ser leve o suficiente para operar em computadores portáteis. Para os testes foram utilizados dois computadores, o primeiro com processador i5-5200 8GB de memória RAM e Windows 10 e o segundo com processador de 3.4ghz (AMD Athlon II X2 B28) com 4GB de memória RAM e Windows 7.
- **Atividade 2. Identificar os Critérios de Aceitação de Desempenho.** - Consiste em identificar o tempo de resposta, o rendimento, os objetivos e restrições de utilização de recursos. Segundo estudos realizados pelo Michotte (2017), eventos que tenham um *delay* de no máximo 56ms são reconhecidos pelo homem como um único evento, esse estudo foi publicado em 1946 e ficou conhecido como *launching effect*. Logo o delay inferior aos 56ms é o critério de aceitação para o teste, já que é o tempo ideal para que não se sinta o atraso na execução dos comandos.
- **Atividade 3. Planejar e desenhar os testes.** - É preciso identificar os cenários chave. Cada dispositivo compatível com o TeamBridge já trata-se de um cenário, a combinação entre esses dispositivos representam outros cenários e essa combinação pode ser modificada no sentido de possibilitar o uso de um ou dois computadores para a combinação de dispositivos.
- **Atividade 4. Configure o ambiente de teste** - Consiste na preparação do ambiente de teste com os softwares necessários para monitorar o hardware, é importante lembrar de remover aplicativos que possam influenciar no teste como anti-vírus. A memória utilizada somando a aplicação cliente e servidor chegou a ocupar 15mb de RAM, por este motivo o objetivo não consiste em verificar o processamento ou uso da memória, e sim o desempenho do software.
- **Atividade 5. Implementar o Design de Teste.** - Deve-se implementar os testes de desempenho. Diante da falta de algum software para esse tipo de teste um pequeno software com essa função também foi construído, a sessão de 5.4.1 explica o funcionamento desse software.

- **Atividade 6. Execute o teste.** - Finalmente deve-se executar e monitorar os testes.

- **Atividade 7. Analise os resultados, o relatório e refaça o teste.** - Por fim, é preciso realizar uma análise dos dados consolidando-os. Caso necessário deve-se reexecutar os testes. Para apresentar os dados serão utilizados gráficos de dispersão, que mostrarão a média, o mínimo e máximo do tempo de resposta. Ao final foi gerada uma tabela com todos os testes realizados, o desvio padrão e o escore médio, este último é a quantidade percentual dos dados que obedecem os limites do desvio padrão. Para garantir uma maior precisão na média da população (IBM, 2016) os testes em que o escore padrão ficou abaixo de 68% foram repetidos.

Para a verificação de significância entre as comparações, será utilizado o teste T pareado. Segundo Wainer et al. (2007), o teste T pareado pode ser utilizado para comparar dois conjuntos de dados, por exemplo, para verificar a diferença entre o tempo de execução de um programa e de outro. O autor ainda explica que o teste T pareado é mais forte que o teste T comum, por conta disso ele foi escolhido para este trabalho.

5.4.1 Software para o teste de desempenho

Para realizar o teste de desempenho, foi criado um projeto que tem como função receber diversos *inputs* de teclado e contar o tempo entre cada um. O objetivo é identificar quanto tempo será gasto para que o software leia os dados do dispositivo e entregue uma entrada de teclado.

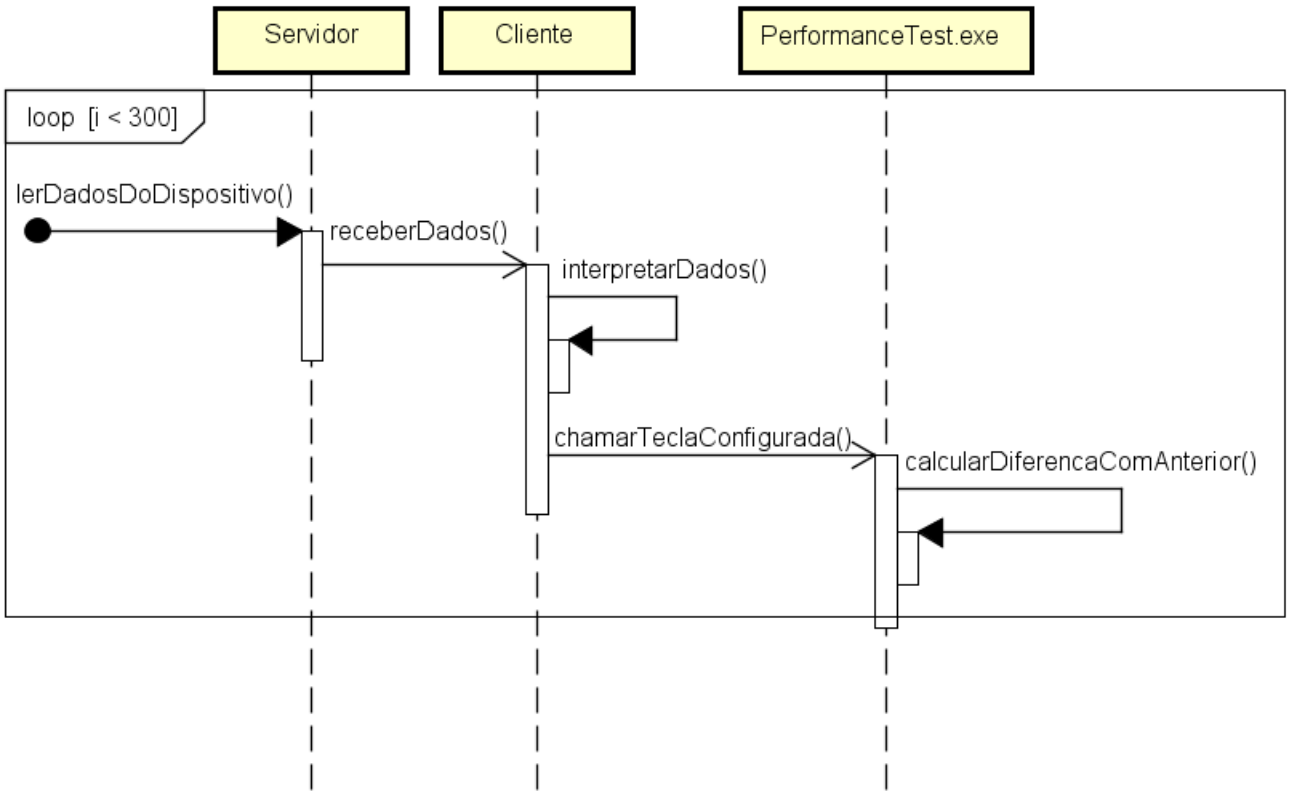
O diagrama de sequência na figura 37 mostra quais são os passos que são contabilizados, ou seja, o tempo contabilizado será a soma do tempo gasto para ler o dispositivo, enviar para o cliente, interpretar os dados e gerar a saída de teclado. Meier et al. (2007) recomenda uma amostra de pelo menos 100 medidas, porém a medição foi realizada com 300 iterações, essa quantidade foi escolhida por levar em torno de 10 segundos com o Kinect, permitindo que oscilações na rede possam ser observadas. A aplicação calcula a média, mediana e desvio padrão do tempo, além de armazenar o intervalo mínimo e máximo entre um input e outro.

O projeto foi construído em C++ e também está disponível no GitHub para ser utilizado em testes futuros.

5.4.2 Aplicação do teste de desempenho

A fim de validar o tempo ideal de 56ms conforme o critério de aceitação, resolveu-se testar um teclado USB comum. Para realizar essa validação, basta executar o projeto de teste e pressionar uma tecla qualquer até o fim do programa. Ao final ele irá gerar

Figura 37 – Diagrama de sequência dos passos contabilizados



Fonte: Elaborado pelo autor.

todas as medidas utilizadas. É importante observar que não pode haver interrupção no pressionamento da tecla, caso isso ocorra o teste deverá ser refeito.

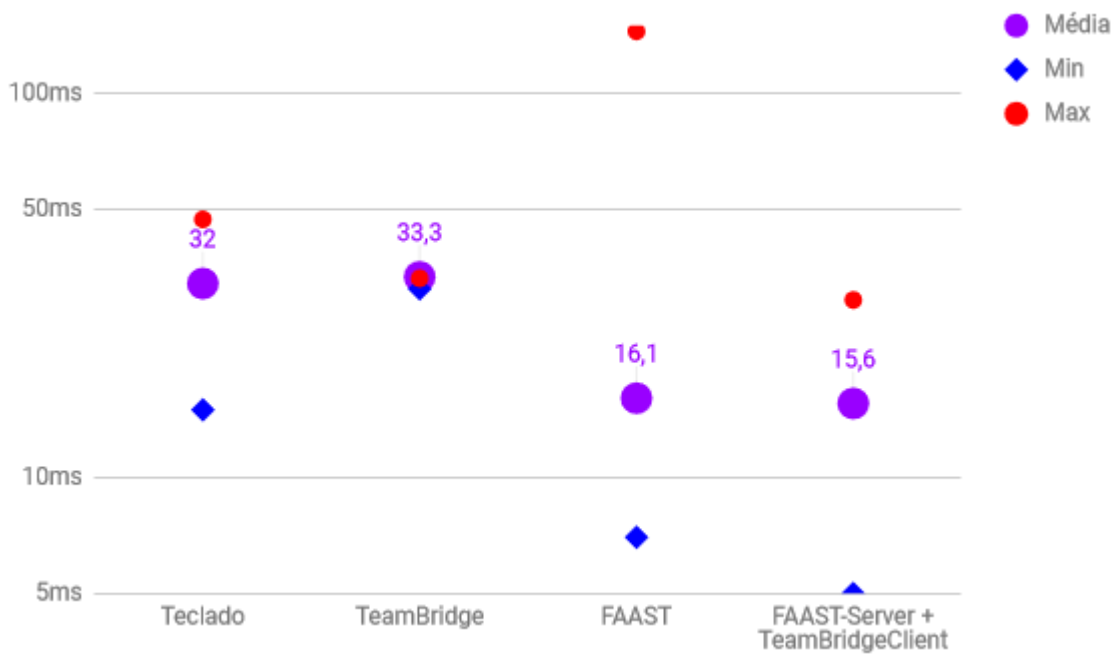
O tempo medido pelo teclado foi abaixo dos 56ms, com 32 $\pm 3,97$ms. Logo, a maioria das pessoas não irá notar diferença de tempo entre o pressionamento de uma tecla e a resposta do software.

Para cumprir o primeiro objetivo do teste, o FAAST foi testado com o Kinect e comparado com o teclado USB e com o TeamBridge. Esse constará como o primeiro cenário, a utilização do Kinect em uma única máquina.

A captura das medidas se dá com o mesmo projeto de teste, mas com a configuração e execução do gesto. Para a comparação com o FAAST, o gesto configurado foi: enquanto elevar a mão acima da cabeça, acionar a tecla “A”. A letra escolhida não faz diferença para esse teste. É importante que não se interrompa o gesto até o final do programa, caso o contrário ele entenderá essa interrupção como tempo de processamento. Para capturar esse gesto foi utilizado o Kinect V1 com SDK 1.7. Com os dados desse teste foi possível gerar o gráfico apresentado na figura 38. O Kinect V2 com SDK 2.0 também foi testado, contudo, o desempenho foi semelhante ao do Kinect V1, não havendo significância

estatística (p=0.9831), sendo assim, para melhor organização serão apresentados apenas como Kinect.

Figura 38 – Teste de carga comparando teclado com FAAST e TeamBridge



Fonte: Elaborado pelo autor.

Na figura 38 é possível observar que o TeamBridge possui um desempenho aceitável por fornecer *inputs* com um intervalo de 33.3 $\pm 5,37$ms, próximo a de um teclado e abaixo dos 56ms. O FAAST por outro lado mostrou um resultado inesperado, conseguindo entregar os *inputs* duas vezes mais rápido que um teclado USB. Devido a esse resultado repetiu-se o teste mantendo o servidor do FAAST, mas com o interpretador do TeamBridge, caracterizando um segundo cenário. De acordo com o resultado ainda foi possível manter o intervalo menor do que o teclado, não havendo significância estatística (p=0,4802) no cenário do FAAST puro contra o servidor do FAAST com o cliente do TeamBridge.

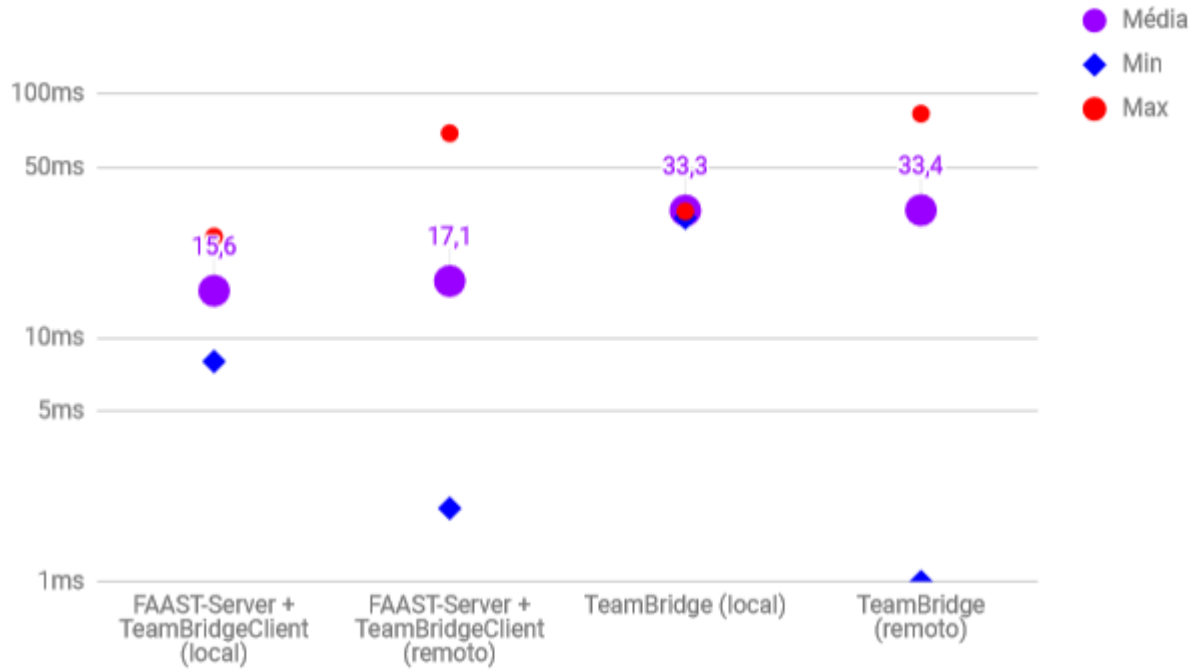
O terceiro cenário leva em consideração o desempenho em rede dos mesmos dispositivos, seus dados estão presentes na figura 39 que compara testes no modo local com remoto. Foi utilizada uma rede cabeada de 100Mbps para o teste.

A figura 39 repete os dados obtidos no teste anterior (local), adicionando o resultado do mesmo teste realizado em rede (remoto).

É possível observar que o desvio padrão do FAAST remoto aumentou de 15,6 $\pm 4,9$ms para 17,1 $\pm 7,21$ms tendo significância estatística (p=0,0041), por outro lado, o TeamBridge passou de 33,3 $\pm 5,37$ms para 33,4 $\pm 8,29$ms, não havendo significância

estatística (p=0,097), com destaque que o TeamBridge conseguiu manter o tempo máximo estável, a diferença foi no tempo mínimo que chegou a ser inferior ao do FAAST.

Figura 39 – Teste de carga comparando modo local com remoto



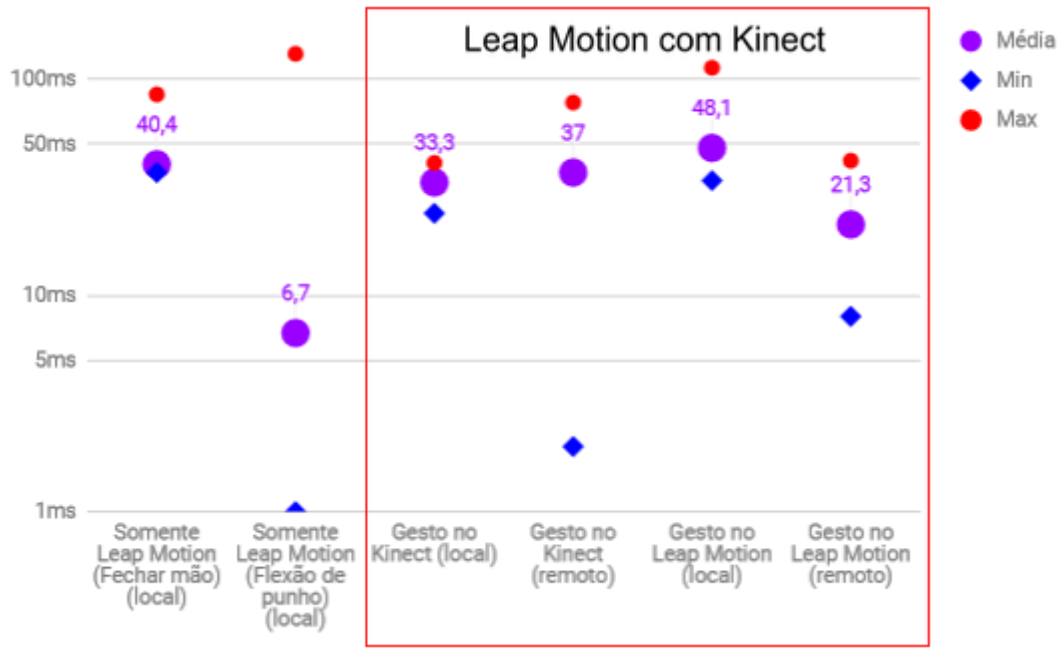
Fonte: Elaborado pelo autor.

Como o VRPN foi pensado para a utilização de vários dispositivos de interação ao mesmo tempo, também foram definidos cenários que atuassem nesse sentido.

Antes de realizar o teste da combinação priorizou-se a repetição dos cenários anteriores do TeamBridge, substituindo o Kinect pelo Leap Motion e a NEDGlove. O Leap Motion mostrou um desempenho inferior ao Kinect. Presumiu-se que isso se devia ao fato de que o gesto de fechar a mão foi reaproveitado da SDK do Leap Motion, logo os cálculos necessários para esse gesto são de sua maioria da própria SDK. A comprovação dessa hipótese requisitou um novo cenário, sendo assim, foi configurado o gesto de flexão de punho. Para este gesto os pontos do punho, mão e cotovelo são passados diretamente para o TeamBridge, onde são realizados os cálculos para verificar se houve flexão ou não. A contabilização deste teste está presente na coluna “Somente Leap Motion (Flexão de punho)(local)”. Em seguida foi feita a mesma configuração dos cenários anteriores com o Kinect, apenas incluindo o Leap Motion no servidor VRPN, esses dados estão presentes na coluna “Gesto no Kinect (local)”. Após esse teste manteve-se o Kinect e o Leap Motion no mesmo servidor, porém foi configurado para que o Leap Motion capturasse: Ao fechar a mão, acionar a tecla “A”. Esses dados constam na coluna “Gesto no Leap Motion (local)”.

Por fim foi feita a comparação com Leap Motion e Kinect em servidores separados, o Kinect foi transferido para outra máquina e foi realizado o teste para o acionamento através do Kinect e do Leap Motion com as mesmas configurações anteriores. A figura 40 apresenta os resultados desse teste.

Figura 40 – Teste de carga unindo Leap Motion com Kinect



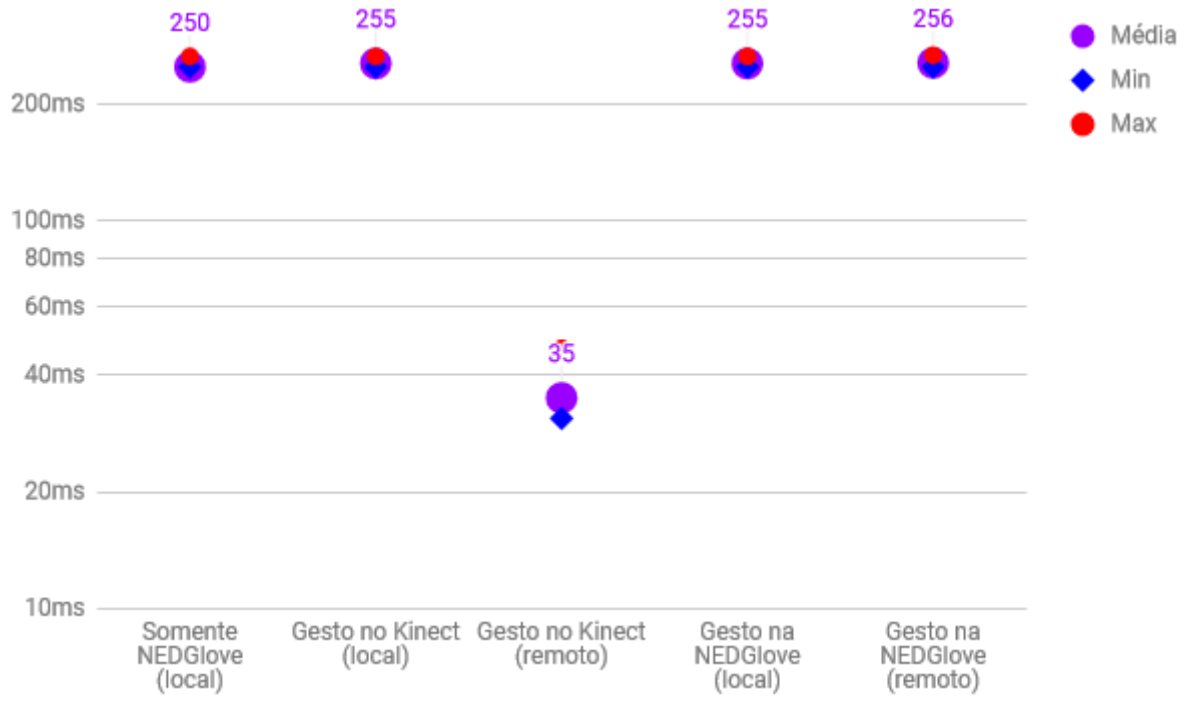
Fonte: Elaborado pelo autor.

É possível observar que o gesto de fechar mão do Leap Motion é mais lento ao entregar os dados para o cliente, contudo ainda mantém-se dentro do tempo ideal. Também comprovou-se a hipótese de que essa diminuição do desempenho se dá por conta dos cálculos realizados pela sua SDK, já que o de flexão de punho mostrou-se muito superior, inclusive quando comparada ao desempenho do FAAST. O gesto de fechar mão identificado pelo Kinect V2 também foi testado, porém o tempo permaneceu igual aos demais gestos do Kinect.

Repetiu-se as mesmas combinações do teste com o Leap Motion, substituindo este pela NEDGlove, a figura 41 apresenta os resultados desses testes.

A NEDGlove apresentou o tempo de $255 \pm 3,45$ms, bem aquém do aceitável, sendo assim esse teste permitiu a observação desse fato e a possibilidade de um trabalho futuro a fim de melhorar o desempenho desse dispositivo, como também trazer uma maior preocupação no desenvolvimento dos próximos dispositivos para esse requisito. É importante observar que a identificação desses problemas fazia parte dos objetivos desse teste.

Figura 41 – Teste de carga unindo NEDGlove com Kinect



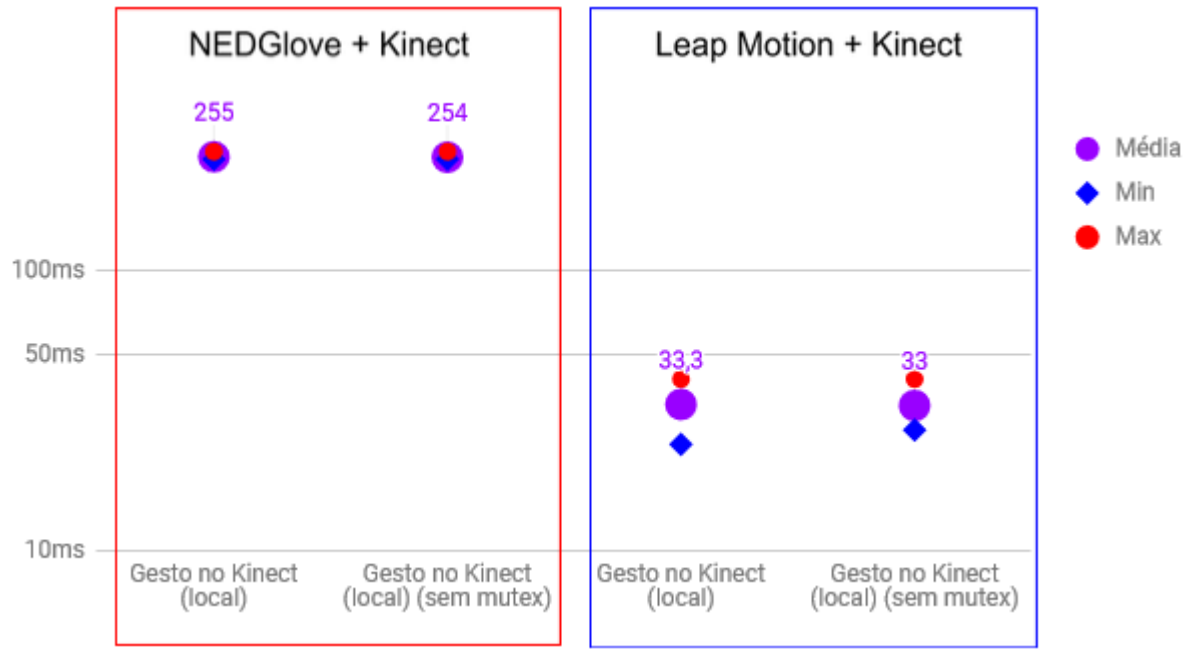
Fonte: Elaborado pelo autor.

Outro problema importante também foi observado durante esses testes. Intermitentemente aconteciam erros causando falha fatal na aplicação, automaticamente fechando a aplicação. Após averiguação foi constatado que o erro ocorria quando dois dispositivos tentavam enviar mensagens ao mesmo tempo. Para solucionar esse problema o código oficial do VRPN foi modificado, implantando a técnica conhecida por *mutex* ou *lock*, técnica utilizada para evitar que dois processos ou *threads* tenham acesso simultaneamente a um setor crítico do código (MARSHALL, 1999). Uma das desvantagens do uso dessa técnica é que uma parte do código deve esperar a outra para poder continuar, devido a isso esperava-se que um dispositivo mais lento influenciasse no desempenho geral do servidor. Diante dessa hipótese, resolveu-se realizar o teste com mais dois cenários, tendo as seguintes combinações: Leap Motion e Kinect, NEDGlove e Kinect, todas no mesmo servidor com o gesto de fechar a mão.

No resultado do teste da combinação do Leap Motion com o Kinect foi observado que somente o dispositivo mais lento perdeu desempenho, no caso o Leap Motion que passou de 40,4 $\pm 3,59$ms para 48,1 $\pm 10,62$ms tendo significância estatística (p=0,0001). Contrariando a hipótese de que o *mutex* iria fazer com que o dispositivo mais rápido fosse atrapalhado pelo mais lento. O Kinect manteve o seu tempo sem significância estatística (p=0,93). Entretanto com o teste de um dispositivo ainda mais lento como o NEDGlove houve uma diminuição drástica no desempenho do Kinect passando de 33,3 $\pm 5,37$ms para

255 ±4, 88ms, ou seja, nesse caso o dispositivo mais lento influenciou consideravelmente no desempenho do dispositivo mais rápido, fazendo com que o melhor tempo fosse determinado somente pelo dispositivo mais lento. Para garantir que esse resultado se devia à implantação do *mutex* o VRPN foi recompilado sem essa técnica e o teste local foi refeito, a figura 42 apresenta o resultado desse teste.

Figura 42 – Teste de carga da união dos dispositivos com e sem *mutex*



Fonte: Elaborado pelo autor.

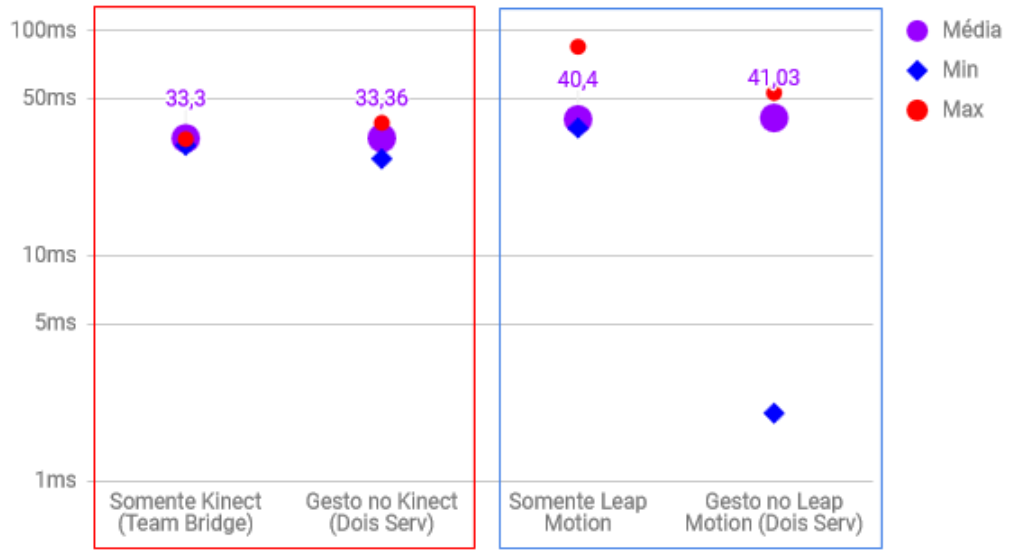
No teste sem o *mutex* foi possível perceber que essa técnica não influenciou no desempenho geral mesmo tendo significância estatística (p=0,0082), uma vez que o desempenho continuou sendo definido pelo dispositivo mais lento. Para confirmação o Kinect e Leap Motion sem *mutex* também foi testado, para esse caso não houve significância estatística (p=0,9638). Logo a hipótese de que o *mutex* iria prejudicar os dispositivos mais rápidos, pode ser descartada para essa versão do VRPN, na qual essa perda ocorre naturalmente.

O problema da perda de desempenho quando utilizamos mais de um dispositivo pode ser contornado utilizando mais de um servidor, tanto em rede quanto localmente. Em rede foi possível observar que o TeamBridge opera com estabilidade, em alguns casos possui um desempenho melhor do que localmente. Sendo assim, pode-se ligar um dispositivo em cada computador.

Porém nem sempre será possível ter vários computadores disponíveis, nesse caso o VRPN também permite a criação de mais de um servidor na mesma máquina, bastando

modificar a porta onde o servidor irá operar. A figura 43 exibe os mesmos testes realizados na união do Kinect com o Leap Motion, porém com dois servidores na mesma máquina, onde foi configurado um servidor na porta original do VRPN para o Kinect e um servidor para o Leap Motion em uma outra porta. Pode-se observar que o Leap Motion quase não sofreu interferência pois sua média operando sozinho era de 40,4 $\pm 3,59$ms passando para 41 $\pm 4,26$ms, inclusive sem significância estatística (p=0,0502). A média do Kinect manteve-se em 33,3 $\pm 2,2$ms também sem significância estatística (p=0,925).

Figura 43 – Teste de carga com dois servidores do TeamBridge na mesma máquina



Fonte: Elaborado pelo autor.

A tabela 7 exibe todas as medições em milissegundos com o desvio padrão e a porcentagem dos resultados que mantiveram-se dentro da média.

Tabela 7 – Tabela com desvio padrão dos testes

Teste	**Desvio padrão (ms)**	**Escore médio**
Teclado	32 ± 3,97	93%
TeamBridge (local)	33,3 ± 5,37	86%
TeamBridge (Remoto)	33,4 ± 8,29	84%
FAAST	16,1 ± 9,4	97%
FAAST-Server + TeamBridgeClient (local)	15,6 ± 4,9	71%
FAAST-Server + TeamBridgeClient (remoto)	17,1 ± 7,21	83%
Kinect V2 (Mão fechada)	34,4 ± 10,1	98%
Kinect V2 (Mão acima da cabeça)	33,3 ± 10,9	70%
Leap Motion		
Leap Motion (Fechar mão) (local)	40,4 ± 3,59	99%
Leap Motion (Flexão de punho) (local)	6,7 ± 13,69	95%
Gesto no Kinect (local)	33,3 ± 2,98	69%
Gesto no Kinect (remoto)	37 ± 10,98	76%
Gesto no Leap Motion (local)	48,1 ± 10,62	80%
Gesto no Leap Motion (remoto)	21,3 ± 4,08	86%
Gesto no Kinect (local) (sem mutex)	33,3 ± 2,38	69%
NEDGlove		
NEDGlove (local)	250 ± 3,45	91%
Gesto no Kinect (local)	255 ± 4,88	68%
Gesto no Kinect (remoto)	35 ± 5,09	77%
Gesto na NEDGlove (local)	255 ± 4,79	73%
Gesto na NEDGlove (remoto)	256 ± 5,27	69%
Gesto no Kinect (local) (sem mutex)	254 ± 4,34	70%
Dois servidores na mesma máquina (Leap Motion e Kinect)		
Gesto no Kinect	33.3 ± 6,52	80%
Gesto Leap Motion	41 ± 4,26	87%

6 Conclusão

Com o TeamBridge foi possível obter uma estrutura de mediação entre os dispositivos de interação e os games de forma não invasiva, levando em consideração que é possível compatibilizar os dispositivos que foram abordados neste projeto até mesmo com games comerciais, não havendo necessidade de modificação do código-fonte. Isso atende a principal motivação que era o retrabalho para readaptar games que já foram produzidos. Com isso a produção de exergames também não necessitará do contato direto com tais dispositivos, um programador interessado em construir um exergame não precisará aprender o funcionamento de uma SDK, ele programará normalmente para funcionar com teclado ou mouse. Apenas o responsável por aplicar o exergame precisará saber como criar um arquivo de configuração para o TeamBridge. A necessidade de modificar o código-fonte do TeamBridge se dará somente quando o desenvolvedor quiser incluir a compatibilidade com um novo dispositivo ou aumentar a funcionalidade do TeamBridge de alguma maneira.

Com este trabalho também foi possível gerar um critério de aceitação e teste de performance para os dispositivos de interação, questão que não era levada em consideração, porém que poderia atrapalhar a experiência do jogador. Com tal teste foi possível identificar que a NEDGlove precisa de melhorias de performance como também identificou que apesar do Leap Motion cumprir o critério de aceitação, a modificação determinados algoritmos desse dispositivo poderá aumentar muito a sua performance. Por fim, o teste comparativo realizado com o FAAST identificou que o TeamBridge possui gargalos quando operado com o Kinect, porém mesmo com tal perda de performance ele ainda está dentro do critério de aceitação.

Também foram identificados dois problemas na união de dois ou mais dispositivos de interação, a necessidade de um controle de fluxo das mensagens e um problema de performance. Porém também foi identificada a possível solução para ambos os problemas.

Por fim o código-fonte está disponível no GitHub e conta com uma documentação na wiki com instruções de instalação, compilação e configuração.

6.0.1 Trabalhos futuros

Este trabalho abre possibilidade para diversos trabalhos futuros como:

- A utilização de aprendizado de máquina para que o TeamBridge reconheça como um novo gesto.
- Criação de um driver e um aplicativo para smartphone a fim de permitir a utilização dos dados gerados no dispositivo, tornando-o um joystick.

- Criar um interpretador de fala para poder-se utilizar palavras como input.
- Disponibilizar os dados dos dispositivos de interação através de um WebService, no lugar de acionar comandos de teclado.
- O TeamBridge poderia ser modificado para acionar equipamentos robóticos, podendo funcionar como intermediador entre sensores e próteses robóticas ou atuar na área de telerobótica.
- Utilização do TeamBridge com dispositivos personalizados para auxiliar pacientes com dificuldades motoras a utilização de computadores.
- Utilização de óculos de realidade virtual para enviar e receber dados para a aplicação VR através do TeamBridge.
- Criação de um algoritmo otimizado para melhorar o desempenho do Leap Motion. Essa melhoria seria não utilizar os métodos da SDK para identificar o gesto de pinça e fechar mão, uma vez que pelos testes foi observado que o algoritmo para identificar flexão do punho teve um desempenho muito superior ao algoritmo da SDK do Leap Motion.
- Implementação de um fluxo contrário de dados para permitir o desenvolvimento de dispositivos hápticos ou dispositivos que gerem algum tipo de feedback para o usuário. Atualmente os dados partem do dispositivo para o TeamBridge em direção à aplicação, nunca o contrário.

Referências

AIRINDO. Lokomat. 2014. Último acesso em 31/01/2018. Disponível em: <http://airindo.com/product/lokomat/>. Citado na página 48.

Artificial Intelligence Laboratory. Falcon haptic device. 2006. Último acesso em 31/01/2018. Disponível em: <http://ai.stanford.edu/~conti/falcon.html>. Citado na página 45.

AUKSTAKALNIS, S.; BLATNER, D. *Silicon Mirage; The Art and Science of Virtual Reality*. [S.l.]: Peachpit Press, 1992. Citado na página 34.

BAKKEN, D. E. *Middleware, Encyclopedia of Distributed Computing*. [S.l.]: Kluwer Academic Press, 2002. Citado na página 33.

BARCALA, L. et al. Análise do equilíbrio em pacientes hemiparéticos após o treino com o programa wii fit. *Fisioterapia em Movimento*, SciELO Brasil, p. 337–343, 2011. Citado 2 vezes nas páginas 25 e 32.

BIANOR, F.; CAVALCANTI, A.; DANTAS, R. R. Evaluate leap motion control for multiple hand posture recognition. *19th Symposium on Virtual and Augmented Reality*, Curitiba, 2017. Citado 2 vezes nas páginas 25 e 38.

BIANOR, F.; CAVALCANTI, A.; DANTAS, R. R. Physiohappy: A low cost device for virtual rehabilitation. *19th Symposium on Virtual and Augmented Reality*, Curitiba, 2017. Citado 2 vezes nas páginas 25 e 37.

BÔAS, A. V. et al. Efeito da terapia virtual na reabilitação motora do membro superior de crianças hemiparéticas. *Rev Neurocienc*, v. 21, n. 4, p. 556–62, 2013. Último acesso em 17/12/2016. Disponível em: <http://repositorio.unicamp.br/handle/REPOSIP/89324>. Citado 2 vezes nas páginas 25 e 32.

BORGHESE, N. A. et al. An integrated low-cost system for at-home rehabilitation. In: *2012 18th International Conference on Virtual Systems and Multimedia*. [S.l.: s.n.], 2012. p. 553–556. Último acesso em 17/12/2016. Citado na página 46.

BRAGA, M. D.; MOTA, G. L. A.; COSTA, R. M. E. M. D. Technologies integration of immersive virtual reality on smartphones with real-time motion capture. In: IEEE. *Virtual and Augmented Reality (SVR), 2016 XVIII Symposium on*. [S.l.], 2016. p. 127–134. Citado na página 34.

BUSCHMANN, F. et al. Pattern-oriented software architecture: A system of patterns. *Jogn Wiley & Sons*, 1996. Citado na página 33.

CAMERON, C. R. et al. Hand tracking and visualization in a virtual reality simulation. In: IEEE. *Systems and Information Engineering Design Symposium (SIEDS), 2011 IEEE*. [S.l.], 2011. p. 127–132. Citado na página 40.

CARVALHO, P. M. C. D. d. *Interação controlada por gestos em ambientes 3D distribuídos*. Dissertação (Mestrado) — Universidade de Aveiro, 2013. Citado 2 vezes nas páginas 39 e 40.

Delft Haptics Lab. Phantom omni. 2014. Último acesso em 31/01/2018. Disponível em: <https://msdn.microsoft.com/en-us/library/windowspreview.kinect.handstate.aspx>. Citado na página 45.

DEMOKRITOS, N. Unobstrusive smart environments for independent living (usefil). 2017. Último acesso em 16/11/2017. Disponível em: <https://usefil.eu/>. Citado na página 55.

ERICH, G. et al. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. Trad. Luiz AM Salgado*. [S.l.]: Bookman, 2000. Citado na página 71.

FERREIRA, G. M. Ferramenta computacional de reconhecimento gestual como apoio a atividades expositivas. 2014. Disponível em: <http://ppgee.vi.ifes.edu.br/wp-content/uploads/2014/10/Acervo-208618-Glauber-Maroto-Ferreira.pdf>. Citado na página 55.

GAMMA, E. *Padrões de Projetos: Soluções Reutilizáveis*. Bookman, 2009. ISBN 9788577800469. Disponível em: <https://books.google.com.br/books?id=U91CYCqTCgkC>. Citado na página 32.

GILES, J. *Inside the race to hack the Kinect*. [S.l.]: Elsevier, 2010. Citado na página 39.

GUERZONI, V. P. D. et al. Análise das intervenções de terapia ocupacional no desempenho das atividades de vida diária em crianças com paralisia cerebral&58; uma revisão sistemática da literatura analysis of occupational therapy interventions in the performance of everyday activities in children with cerebral palsy&58; a systematic review of the literature. *Revista Brasileira de Saúde Materno Infantil*, Directory of Open Access Journals, v. 8, n. 1, p. 17–25, 2008. Citado na página 31.

HAND, C. Other faces of virtual reality. *Multimedia, Hypermedia, and Virtual Reality Models, Systems, and Applications*, Springer, p. 107–116, 1996. Citado na página 34.

HOLLOWAY, R. L. Registration error analysis for augmented reality. *Presence: Teleoper. Virtual Environ.*, MIT Press, Cambridge, MA, USA, v. 6, n. 4, p. 413–432, ago. 1997. ISSN 1054-7460. Disponível em: <http://dx.doi.org/10.1162/pres.1997.6.4.413>. Citado na página 42.

HUANG, M.-C. et al. Smartglove for upper extremities rehabilitative gaming assessment. In: ACM. *Proceedings of the 5th International Conference on Pervasive Technologies Related to Assistive Environments*. [S.l.], 2012. p. 20. Citado 4 vezes nas páginas 28, 52, 53 e 80.

IBM. Conceitos comuns na análise estatística. 2016. Último acesso em 24/02/2018. Disponível em: <https://www.ibm.com/support/knowledgecenter/pt-br/SSEP7J_10.1.1/com.ibm.swg.ba.cognos.ug_cr_rptstd.10.1.1.doc/c_id_rs_stats.html>. Citado na página 89.

JACOBSON, D.; WOODS, D.; BRAIL, G. *APIs: A strategy guide*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 62.

KALAPANIDAS, E. et al. Playmancer: a serious gaming 3d environment. 2008. Último acesso em 17/12/2016. Disponível em: <https://www.researchgate.net/publication/224353421_PlayMancer_a_Serious_Gaming_3D_Environment>. Citado na página 52.

KOLAKOWSKI, N. Microsoft acquires canesta, 3d tech patents. 2010. Último acesso em 07/09/2017. Disponível em: <http://www.eweek.com/news/microsoft-acquires-canesta-3d-tech-patents>. Citado na página 39.

KONSTANTINIDIS, E. I. et al. A lightweight framework for transparent cross platform communication of controller data in ambient assisted living environments. *Information Sciences*, Elsevier, v. 300, p. 124–139, 2015. Citado 2 vezes nas páginas 55 e 56.

KRISHNAMURTHY, G. Moticon's opengo insoles with wireless sensors. 2013. Último acesso em 31/01/2018. Disponível em: <https://www.medgadget.com/2013/09/moticons-opengo-insoles-with-wireless-sensors.html>. Citado na página 45.

KRUEGER, M. W. Responsive environments. In: ACM. *Proceedings of the June 13-16, 1977, national computer conference*. [S.l.], 1977. p. 423–433. Citado na página 34.

KRUEGER, M. W.; GIONFRIDDO, T.; HINRICHSEN, K. Videoplace—an artificial reality. In: ACM. *ACM SIGCHI Bulletin*. [S.l.], 1985. v. 16, n. 4, p. 35–40. Citado na página 34.

LATTA, J. N.; OBERG, D. J. A conceptual virtual reality model. *IEEE Computer Graphics and Applications*, IEEE, v. 14, n. 1, p. 23–29, 1994. Citado na página 34.

LESTON, J. Virtual reality: the it perspective. *ITNOW*, Oxford University Press, v. 38, n. 3, p. 12–13, 1996. Citado na página 34.

LUN, R.; ZHAO, W. A survey of applications and human motion recognition with microsoft kinect. *International Journal of Pattern Recognition and Artificial Intelligence*, World Scientific, v. 29, n. 05, p. 1555008, 2015. Citado 4 vezes nas páginas 38, 39, 40 e 41.

MARSHALL, D. Further threads programming:synchronization. 1999. Último acesso em 23/12/2017. Disponível em: <http://users.cs.cf.ac.uk/Dave.Marshall/C/node31.html>. Citado na página 94.

MARTIN-NIEDECKEN, A. L. et al. Rehabconnex: A middleware for the flexible connection of multimodal game applications with input devices used in movement therapy and physical exercising. In: *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. [s.n.], 2015. p. 496–502. ISSN 2325-4270. Último acesso em 13/11/2016. Disponível em: <http://ieeexplore.ieee.org/document/7317671/>. Citado na página 48.

MATHIOWETZ, V. et al. Adult norms for the box and block test of manual dexterity. *American Journal of Occupational Therapy*, American Occupational Therapy Association, v. 39, n. 6, p. 386–391, 1985. Citado na página 25.

MEIER, J. et al. *Performance Testing Guidance for Web Applications: Patterns & Practices*. Redmond, WA, USA: Microsoft Press, 2007. ISBN 9780735625709. Citado 3 vezes nas páginas 87, 88 e 89.

MELO, T. P. de. Gesture recognition for natural interaction with applications. 2012. Citado na página 56.

MICHOTTE, A. *The perception of causality*. [S.l.]: Routledge, 2017. v. 21. Citado na página 88.

MILGRAM, P. et al. Augmented reality: A class of displays on the reality-virtuality continuum. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Telemanipulator and telepresence technologies*. [S.l.], 1995. v. 2351, p. 282–293. Citado 2 vezes nas páginas 34 e 35.

MORALES, B. et al. Asterics: a new flexible solution for people with motor disabilities in upper limbs and its implication for rehabilitation procedures. *Disability and Rehabilitation: Assistive Technology*, Taylor & Francis, v. 8, n. 6, p. 482–495, 2013. Citado na página 56.

MOSSEL, A. et al. Artifice-augmented reality framework for distributed collaboration. *International Journal of Virtual Reality*, 2013. Citado na página 54.

MSDN, M. Skeletal tracking. 2012. Último acesso em 04/02/2017. Disponível em: <https://msdn.microsoft.com/en-us/library/hh973074.aspx>. Citado na página 86.

MSDN, M. Handstate enumeration. 2014. Último acesso em 27/12/2017. Disponível em: <https://msdn.microsoft.com/en-us/library/windowspreview.kinect.handstate.aspx>. Citado na página 69.

MURTHY, G.; JADON, R. A review of vision based hand gestures recognition. *International Journal of Information Technology and Knowledge Management*, v. 2, n. 2, p. 405–410, 2009. Citado na página 35.

NETTO, A. V.; MACHADO, L. d. S.; OLIVEIRA, M. C. F. d. Realidade virtual-definições, dispositivos e aplicações. *Revista Eletrônica de Iniciação Científica-REIC. Ano II*, v. 2, 2002. Citado na página 34.

OH, Y.; YANG, S. Defining exergames & exergaming. *Proceedings of Meaningful Play*, p. 1–17, 2010. Citado na página 25.

OMELINA, L. et al. Serious games for physical rehabilitation: designing highly configurable and adaptable games. In: *Proceedings of the 9th International Conference on Disability, Virtual Reality & Associated Technologies*. [s.n.], 2012. p. 195–201. Último acesso em 15/12/2016. Disponível em: <http://www.icdvrat.org/2012/papers/ICDVRAT2012_S06N5_Omelina_etal.pdf>. Citado na página 51.

PERRY, J. C. et al. Effective game use in neurorehabilitation: user-centered perspectives. In: *Handbook of research on improving learning and motivation through educational games: Multidisciplinary approaches*. [S.l.]: IGI Global, 2011. p. 683–725. Citado na página 51.

PINTO, F. et al. Automated architectural evaluation of web information systems. In: ACM. *Proceedings of the 19th Brazilian symposium on Multimedia and the web*. [S.l.], 2013. p. 225–232. Citado na página 60.

PIROVANO, M. et al. The design of a comprehensive game engine for rehabilitation. In: IEEE. *Games Innovation Conference (IGIC), 2013 IEEE International*. [S.l.], 2013. p. 209–215. Citado na página 44.

PIROVANO, M. et al. Iger-intelligent game engine for rehabilitation. *IEEE Trans. Comput. Intell. AI Games, PP*, p. 1, 2014. Citado na página 46.

POHL, K.; BÖCKLE, G.; LINDEN, F. J. van D. *Software product line engineering: foundations, principles and techniques*. [S.l.]: Springer Science & Business Media, 2005. Citado na página 26.

PREE, W. Hot-spot-driven framework development. *Framework*, v. 2, p. B1, 1995. Citado na página 33.

PRESMAN, R. S. *Engenharia de Software: Uma abordagem Profissional*. [S.l.]: Porto Alegre: AMGH. 780p, 2011. Citado 4 vezes nas páginas 26, 29, 59 e 64.

Prime Sense. Prime sense™ nite algorithms 1.5. 2011. Último acesso em 06/09/2017. Disponível em: <http://www.openni.ru/wp-content/uploads/2013/02/NITE-Algorithms.pdf>. Citado na página 41.

QUALISYS; GIRAULT, A.; BOGER, Y. Available hardware devices. 2016. Último acesso em 18/10/2017. Disponível em: <https://github.com/vrpn/vrpn/wiki/Available-hardware-devices>. Citado na página 44.

REED, K. L.; SANDERSON, S. N. *Concepts of occupational therapy*. [S.l.]: Lippincott Williams & Wilkins, 1999. Citado na página 31.

REGO, P.; MOREIRA, P. M.; REIS, L. P. Serious games for rehabilitation a survey and a classification towards a taxonomy. 2010. Último acesso em 27/10/2016. Disponível em: <http://www.academia.edu/2831754/A_Survey_on_Serious_Games_for_Rehabilitation>. Citado 3 vezes nas páginas 25, 31 e 32.

Reha Stim. Tyromotion tymo. 2014. Último acesso em 31/01/2018. Disponível em: <http://www.reha-stim.de/cms/index.php?id=133>. Citado na página 45.

ROBERTSON, G. G.; CARD, S. K.; MACKINLAY, J. D. Three views of virtual reality: nonimmersive virtual reality. *Computer*, IEEE, v. 26, n. 2, p. 81, 1993. Citado na página 34.

SANTOS, F. S. *Abstração de eventos de sensores para dispositivos de interação*. Tese (Doutorado) — Universidade de São Paulo, 2008. Citado na página 56.

SANTOS, J. V. S.; CARVALHO, L. C.; BRESSAN, P. A. Physioplay: um exergame para reabilitação física aplicando a interatividade do kinect como biofeedback visual. In: *IX Workshop de Realidade Virtual e Aumentada (WRVA), Paranavaí*. [S.l.: s.n.], 2012. Citado na página 32.

SANTOS, L. H. d. O. Arcabouço para construção de jogos ubíquos com foco em reabilitação. 2016. Citado na página 47.

SATO, A. T.; BATISTA, M. P. P.; ALMEIDA, M. H. M. de. "programas de estimulação da memória e funções cogniti-vas relacionadas": opiniões e comportamentos dos idosos participantes. *Revista de Terapia Ocupacional da Universidade de São Paulo*, v. 25, n. 1, p. 51–59, 2014. Citado na página 31.

SBGames. Kayak supremo. 2013. Último acesso em 04/02/2018. Disponível em: <http://www.sbgames.org/sbgames2013/festival/post.php?tag=game114>. Citado na página 83.

SCHMIDT, D. C.; BUSCHMANN, F. Patterns, frameworks, and middleware: their synergistic relationships. In: IEEE COMPUTER SOCIETY. *Proceedings of the 25th international conference on Software engineering*. [S.l.], 2003. p. 694–704. Citado na página 33.

Sensory-Motor Systems Lab. Charmin. 2013. Último acesso em 31/01/2018. Disponível em: <http://www.sms.hest.ethz.ch/research/current-research-projects/armin-robot/charmin.html>. Citado na página 48.

SHEN, J.; PANTIC, M. A software framework for multimodal humancomputer interaction systems. In: IEEE. *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. [S.l.], 2009. p. 2038–2045. Citado 4 vezes nas páginas 28, 29, 49 e 50.

SHEN, J.; SHI, W.; PANTIC, M. Hciˆ2 workbench: A development tool for multimodal human-computer interaction systems. In: *Face and Gesture 2011*. [s.n.], 2011. p. 766–773. Último acesso em 13/11/2016. Disponível em: <http://ieeexplore.ieee.org/document/5771346/>. Citado 2 vezes nas páginas 49 e 50.

SILVA, L. et al. Phys. io: Wearable hand tracking device. In: IEEE. *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2016 IEEE International Conference on*. [S.l.], 2016. p. 1–6. Citado 3 vezes nas páginas 25, 36 e 37.

SILVA, L. et al. Development of a low cost dataglove based on arduino for virtual reality applications. In: IEEE. *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2013 IEEE International Conference on*. [S.l.], 2013. p. 55–59. Citado 2 vezes nas páginas 25 e 36.

SOARES, E. et al. Projeto memória e envelhecimento: capacitando profissionais e aprimorando aspectos cognitivos em idosos institucionalizados. *Revista Brasileira de Ciências do Envelhecimento Humano*, v. 7, n. 1, 2011. Citado na página 31.

SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Prentice Hall, 2011. ISBN 978-85-7936-108-1. Citado 4 vezes nas páginas 29, 59, 60 e 67.

SU, C.-J. Personal rehabilitation exercise assistant with kinect and dynamic time warping. v. 3, p. 448–454, 01 2013. Citado 2 vezes nas páginas 28 e 53.

SUMA, E. A. et al. Faast: The flexible action and articulated skeleton toolkit. In: IEEE. *Virtual Reality Conference (VR), 2011 IEEE*. [S.l.], 2011. p. 247–248. Citado 2 vezes nas páginas 28 e 43.

Taylor II, R. M. et al. Vrpn: a device-independent, network-transparent vr peripheral system. In: ACM. *Proceedings of the ACM symposium on Virtual reality software and technology*. [S.l.], 2001. p. 55–61. Citado 2 vezes nas páginas 41 e 42.

TEÓFILO, L. F.; NOGUEIRA, P. A.; SILVA, P. B. Gemini: A generic multi-modal natural interface framework for videogames. In: *Advances in Information Systems and Technologies*. [S.l.]: Springer, 2013. p. 873–884. Citado na página 54.

WAINER, J. et al. Métodos de pesquisa quantitativa e qualitativa para a ciência da computação. *Atualização em informática*, v. 1, p. 221–262, 2007. Citado na página 89.

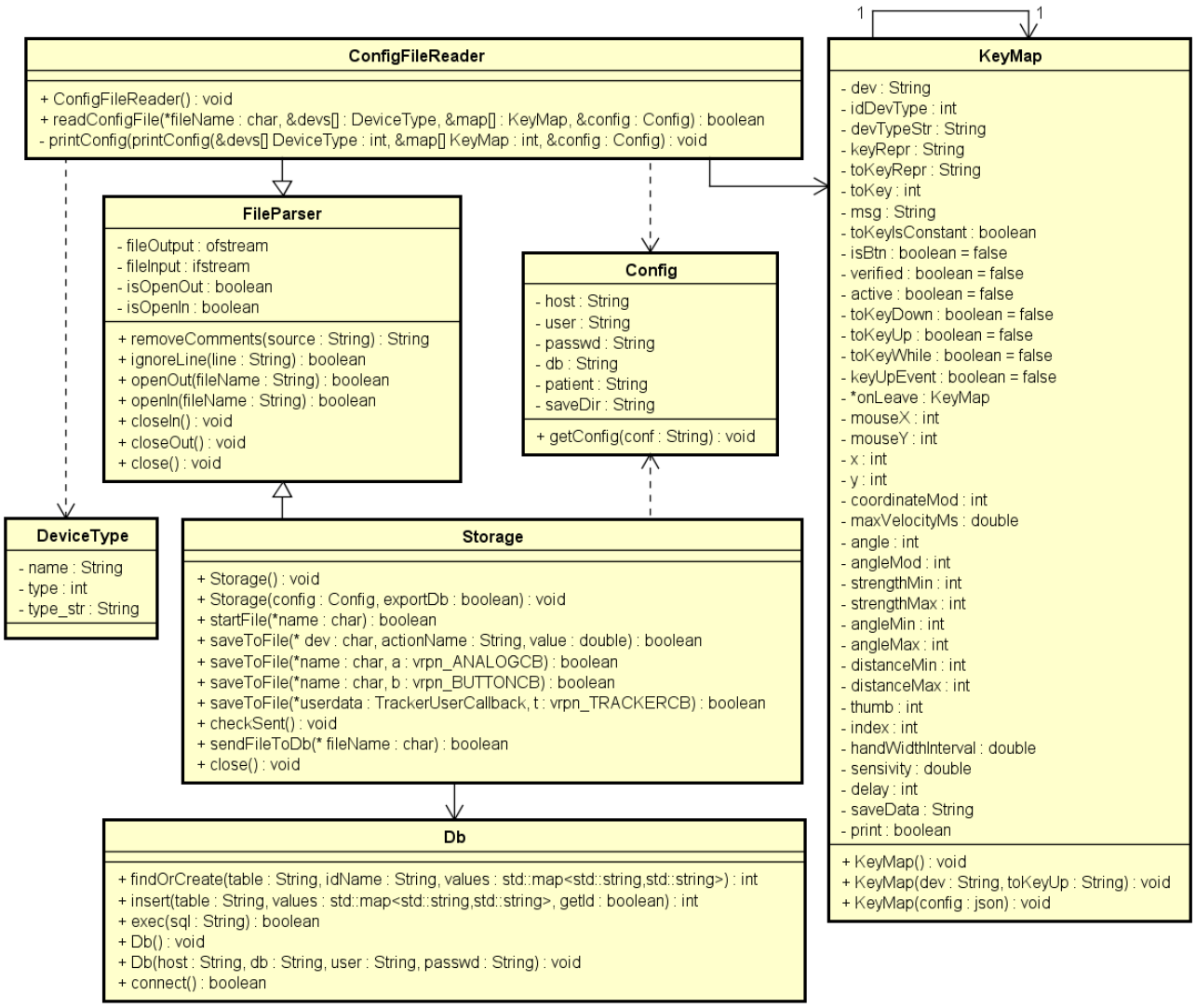
WATERS, K. R. Getting dressed in the early morning: styles of staff/patient interaction on rehabilitation hospital wards for elderly people. *Journal of Advanced Nursing*, Wiley Online Library, v. 19, n. 2, p. 239–248, 1994. Citado na página 31.

WEICHERT, F. et al. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 13, n. 5, p. 6380–6393, 2013. Citado na página 38.

Apêndices

APÊNDICE A – Diagrama UML das classes de configuração e armazenamento

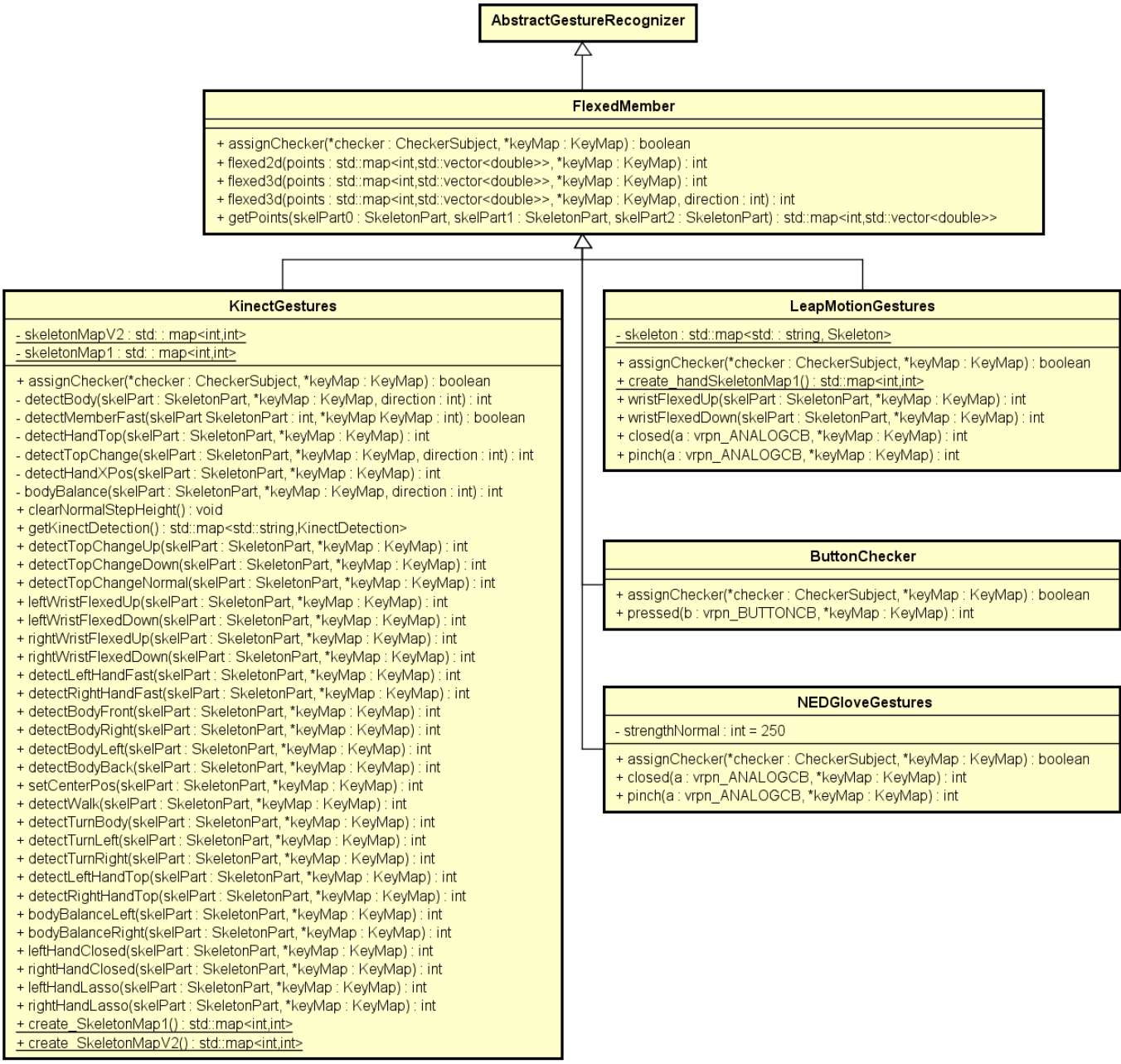
Figura 44 – Classes responsáveis pela leitura da configuração e armazenamento



Fonte: Elaborado pelo autor.

APÊNDICE B – Diagrama UML das classes de gestos

Figura 45 – Classes responsáveis pela detecção dos gestos



Fonte: Elaborado pelo autor.