UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

Desenvolvimento de uma comunicação entre o Unity3D e o braço robótico do tipo SCARA utilizando protocolo TCP/IP.

Guilherme Rodrigues Chiqueti

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

São Carlos

2017

GUILHERME RODRIGUES CHIQUETI

Desenvolvimento de uma comunicação entre o Unity3D e o braço robótico do tipo SCARA utilizando protocolo TCP/IP.

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

Curso de Engenharia Elétrica com ênfase em Sistemas de Energia e Automação

ORIENTADOR: Glauco Augusto de Paula Caurin

São Carlos

2017

Chiqueti, Guilherme Rodrigues
C532d Desenvolvimento de uma comunicação entre o Unity3D e o braço robótico do tipo SCARA utilizando protocolo TCP/IP. / Guilherme Rodrigues Chiqueti; orientador Glauco Augusto de Paula Caurin. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Jogos sérios. 2. Reabilitação Robótica. 3. Unity3D. 4. TCP/IP. 5. Socket. I. Título.

FOLHA DE APROVAÇÃO

Nome: Guilherme Rodrigues Chiqueti

Título: "Desenvolvimento de protocolo de comunicação TCP/IP entre o Unity3D e o braço robótico do tipo SCARA"

Trabalho de Conclusão de Curso defendido e aprovado em 28 / 06 / 2017,

com NOTA 7,5 (sete, cinco), pela Comissão Julgadora:

Prof. Associado Glauco Augusto de Paula Caurin - Orientador - SEM/EESC/USP

Prof. Dr. Maximiliam Luppe - SEL/EESC/USP

Mestre Viviane Cristina Roma Appel - Doutoranda - SEM/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado José Carlos de Melo Vieira Júnior

Agradecimentos

A Deus por ter me dado saúde e força para superar as dificuldades.

A minha família, principalmente aos meus pais, Edson e Bárbara, por me darem forças nos altos e baixos desse caminho. Sou grato pelo apoio e ajuda nos momentos fáceis e de desespero, pelas conversas que me fortaleceram a terminar esse trabalho. Sou grato pelos momentos de felicidade e de preocupação. Não há outra fonte de inspiração maior que eles pra mim.

Aos mestrandos e doutorandos dos laboratórios de Dinâmica e Mecatrônica da EESC, pelas conversas e motivações comigo durante o período deste trabalho. O aprendizado decorrido dessas conversas foi de grande importância para a conclusão deste trabalho. Agradecimento especial ao Thales e Henrique.

Ao meu orientador, Glauco Caurin, por me permitir trabalhar no Laboratório de Mecatrônica e fornecer todos os equipamentos necessários. Agradeço também pela motivação e ajuda com os problemas que tive.

Aqueles que seguiram comigo durante os anos da graduação: Noel Araújo, Bruno, Kae, Gabriel, Augusto, e a todos aqueles que de alguma forma estiveram e estão próximos a mim, fazendo esta vida valer cada vez mais a pena.

Sumário

Agradecimentos....3

Índice de figuras....7

Lista de quadros....9

Lista de Abreviações e Siglas....11

Resumo....13

Abstract....15

Capítulo 1- Introdução....17

1.1 Objetivos....18

1.2 Estrutura do trabalho....18

Capítulo 2-Revisão Bibliográfica....19

2.1- Redes de Computadores....19

2.2- Camadas de comunicação e o protocolo TCP/IP....20

2.3- Servidores e Clientes....21

2.4- *Socket*....21

2.5- Reabilitação robótica e os jogos sérios....22

Capítulo 3– Materiais e Métodos....25

3.1 – O robô do tipo SCARA....25

3.2- Controle de impedância....28

3.3- O Unity3D....30

3.4- Funcionalidade do Software....31

Capítulo 4- Experimento....35

4.1- O jogo "Capture o Azul"....35

Capítulo 5- Conclusões finais.....39

Capítulo 6- Referências Bibliográficas.....41

Apêndice A – Manual do SCARA.....43

Índice de figuras

Figura 1- Paciente com AVE durante terapia no hospital de reabilitação Burke.....18

Figura 2– Exemplo de endereço IPv4.....19

Figura 3- Uma rede TCP/IP.....21

Figura 4- O SCARA.....25

Figura 5- Ilustação dos ângulos de movimento do manipulador.....26

Figura 6 - Representação do espaço de trabalho do SCARA.....26

Figura 7- Transdutor de força em seis eixos acoplado ao TCP do manipulador.....27

Figura 8- Visão geral dos componentes do sistema SCARA.....28

Figura 9- Ilustração de um manipulador robótico ccom duas juntas em série e alinhadas.....29

Figura 10- Ambiente de trabalho do Unity3D.....30

Figura 11- Fluxograma do servidor.....31

Figura 12- o dado encapsulado em um pacote TCP/IP é um vetor de bytes com os dados de posição e velocidade do TCP do SCARA e comprimento total de 48 bytes.....32

Figura 13- Exemplo de uso da classe do cliente.....33

Figura 14- Foto retirada do jogo Capture o Azul.....35

Figura 15- Fluxograma do cliente usado no Capture o Azul. A parte START do fluxograma é executada no início do jogo e a UPDATE é executada uma vez a cada frame.....36

Figura 16 - Interface do usurário do programa de controle de impedância.....46

Figura 17- Imagem da instância CAN1 após o término do processo.....48

Figura 18– Imagem da instância CAN2 após o término do processo.....48

Figura 19- Localização do endereço local IPv4 da máquina na rede conectada.....50

Lista de quadros

Quadro 1- Descrição do programa controlador do robô SCARA.....46

Quadro 2- Funções contidas na API para a utilização no Unity3D.....49

Lista de Abreviações e Siglas

MMORPG	Massively Multiplayer Online Role Playing Game
TCP	Tool Center Point
TCP/IP	Transmission Control Protocol / Internet Protocol
I/O	Input/Output
API	Application Programming Interface
CAN	Controler Area Network
fps	frames per second

Resumo

Este projeto apresenta o desenvolvimento de um software contendo um protocolo de comunicação para utilização no braço robótico SCARA. O robô serve para propósitos gerais, mas nesse projeto é utilizado como ferramenta de terapia de movimentos no tratamento de pacientes em reabilitação de membros superiores. A solução oferece ao paciente uma alternativa mais atraente para o tratamento. O projeto baseou-se na elaboração de dois programas distintos que se comunicam usando protocolo TCP/IP. Um programa atua como servidor e fica responsável por enviar informações do robô SCARA para a rede, enquanto o outro programa, atuando como cliente, recebe essas informações para que o desenvolvedor de jogos, utilizando o Unity3D, possa trabalhá-las. A implementação do protocolo TCP/IP permite que o jogo e o programa do SCARA sejam executados em computadores distintos, o que permite ao terapeuta analisar o desempenho do paciente durante a terapia. Por fim realizou-se um experimento com o objetivo de validar os softwares desenvolvidos. Foi criado um jogo no ambiente Unity3D que utilizou o programa cliente. Durante 120 segundos o jogo permaneceu ativado e o manipulador SCARA, com o programa servidor, foi movimentado de forma a cumprir com os desafios do jogo. Foi observado que a taxa média de envio de mensagens do servidor para o cliente chegou a 115 fps, maior que a taxa de atualização do jogo que atingia um máximo de 80 fps.

Socket, TCP/IP, Reabilitação Robótica, Jogos sérios, Unity3D, Software

Abstract

This project presents the development of a software containing a communication protocol for use in the robotic arm SCARA. The robot serves for general purposes, but in this project it is used as a tool of movement therapy in the treatment of patients in rehabilitation of upper limbs. The solution offers the patient a more attractive alternative to the treatment. This project was based on the elaboration of two distinct programs that communicate with each other using TCP/IP protocol. One program acts as a server and is responsible for sending information from SCARA to network, while the other program, acting as a client, receives the information so that the game developer using Unity3D can work with them. The implementation of the TCP/IP protocol allows client and server to run on separate computers, allowing the therapist to analyze the patient's performance during therapy. A game was created in the Unity3D environment that used the client program. For 120 seconds the game remained activated and the SCARA handler, with the server program, was moved to meet the challenges of the game. It was observed that the average rate of sending messages from the server to the client reached 115 fps, higher than the game update rate that reached a maximum of 80 fps (frames per second).

Socket, TCP/IP, Robotic rehabilitation, Serious games, Unity3D, Software

Capítulo 1- Introdução

Jogos sérios são definidos como jogos que abordam aspectos que transcendem o objetivo de entretenimento. Esses jogos são desenvolvidos para diversas aplicações como treinamento militar, comunicação estratégica, educação e a área da saúde. Em adição ao entretenimento proporcionado pelos jogos comuns, os jogos sérios promovem e avaliam o desenvolvimento dos jogadores (ZYDA, 2005).

A fim de verificar a eficácia dos jogos sérios, a Universidade do Colorado realizou um estudo comparando treineiros que aprenderam conteúdos diversos usando o método convencional com aqueles que usaram jogos para o aprendizado. Verificou-se um aumento de 9% na retenção do aprendizado, 14% na habilidade comportamental e 11% no aprendizado factual dos alunos que utilizaram os jogos (UNIVERSITY OF COLORADO DENVER, 2010), mostrando a eficiência que os jogos podem trazer.

Dentre outros campos, a reabilitação robótica é uma forma de terapia que, assistida por jogos, pode trazer benefícios. Enquanto os jogos aumentam a motivação do usuário proporcionando desafios, os manipuladores robóticos controlados auxiliam o movimento de alguma articulação do paciente de forma confiável e segura (ANDRADE et al., 2013). Além disso as sessões de treinamento deixam de ser influenciadas pelo cansaço ou estado emocional do terapeuta, garantindo uniformidade no tratamento (CAURIN et al., 2011).

Como exemplo de reabilitação robótica assistida a jogos, podemos citar o braço robótico MIT-MANUS. O manipulador, mostrado na figura 1, foi desenvolvido para ser utilizado na reabilitação de membros superiores de pacientes que sofreram AVE (acidente vascular encefálico). Este manipulador move-se suavemente graças ao controle de impedância implementado, cumprindo com as ações motoras do paciente (KREBS et al., 2003), enquanto os jogos sérios motivam o paciente com desafios e entretenimento, e avaliam seu desempenho ao decorrer do jogo.

Figura 1- Paciente com AVE durante terapia no hospital de reabilitação Burke.

Fonte: KREBS (2003).

1.1 Objetivos

Visando futuros trabalhos relacionados à reabilitação robótica com jogos no Laboratório de Mecatrônica da USP de São Carlos, o objetivo foi desenvolver uma ferramenta de comunicação utilizando o protocolo TCP/IP entre o braço robótico do tipo SCARA pertencente ao Laboratório de Mecatrônica, que opera em linguagem C/C++ e a ferramenta de desenvolvimento de jogos digitais Unity3D, cujo algoritmo foi escrito na linguagem C#. O projeto deste trabalho adicionado ao controle de impedância previamente implementado no manipulador SCARA permite a utilização do mesmo para reabilitação robótica com jogos.

1.2 Estrutura do trabalho

O capítulo 2 revisa os conceitos que foram utilizados no desenvolvimento deste trabalho. Inclui o conceito de IP, servidor e cliente, protocolo de comunicação TCP/IP e *socket.*

O capítulo 3 introduz e detalha os materiais e métodos usados neste trabalho. Ele detalha o funcionamento do braço robótico SCARA pertencente ao laboratório de Mecatrônica. Além disso, descreve os hardwares utilizados e o software de controle de impedância, que foi desenvolvido em outro projeto (FERNANDES, 2013). Por fim, é apresentado o ambiente de desenvolvimento de jogos Unity3D utilizado neste trabalho.

O capítulo 4 mostra o experimento realizado com o software gerado no braço robótico. Inclui o experimento de comunicação com um jogo criado com o Unity3D.

Capítulo 2-Revisão Bibliográfica

A comunicação entre os dois computadores foi realizada com *sockets* que, uma vez acoplados em portas de comunicação, realizam o tráfego de informação entre diferentes aplicações. Um estudo dos conceitos descritos neste capítulo foram indispensáveis para o uso de *sockets* e consequente realização deste projeto de graduação.

2.1- Redes de Computadores

Uma rede de computadores consiste de máquinas interconectadas por canais de comunicação. Essas máquinas podem ser *hosts* e roteadores. *Hosts* são máquinas cuja finalidade é executar as aplicações do usuário (TANENBAUM, 2003), como um computador pessoal por exemplo.

Roteadores, por outro lado, são dispositivos com a função de passar dados de um canal de comunicação para outro. Sua importância vem do fato de que não é conveniente interligar todos os computadores entre si dois a dois. Em vez disso, vários computadores são conectados a um roteador, que por sua vez conectados a outros roteadores, e assim por diante, formando uma rede de comunicação (DONAHOO; CALVERT, 2009).

Cada dispositivo conectado a uma rede possui um endereço único. Esse endereço é como uma identificação do computador, e é usado para reconhecimento do dispositivo por outros visando troca de informações. No protocolo IP versão 4 ou IPv4, cada endereço é representado por um número de 4 bytes separados por ponto, sendo cada byte um valor entre 0 e 255. Um exemplo de endereço IPv4 se encontra na figura 2, imagem esta tirada do *prompt* de comando no Windows 7 de um computador pessoal após o uso da função *ipconfig*.

Figura 2– Exemplo de endereço IPv4.

O endereço no protocolo IPv4 pode ser dividido em duas partes. Em uma parte está armazenado o endereço local da rede e em outra o número do *host*. O que define quais dos números do protocolo IPv4 pertencem à rede e quais números pertencem aos *hosts* é a máscara de Sub-rede, que contém números de mesma magnitude que o IPv4. Esse número é como uma senha, e seus bits setados em 1 indicam quais números do endereço representam a rede. No exemplo da figura 2 os três primeiros números da máscara de sub-rede estão em 255 e o último em 0, indicando que os três primeiros bytes estão com seus bits setados em

1 e o restante em 0. Isso indica que os três primeiros bytes do endereço representam o endereço da rede local e o último byte representa o número do *host* conectado a ele.

Porém, com a quantidade de dispositivos crescendo, o número de endereços disponíveis no IPv4 vai saturar e não sobrará endereços para os novos dispositivos. Por isso, recentemente foi oficializado o IPv6 que possui um tamanho de endereçamento de 128 bits.

Para realizar a comunicação de dispositivos, é necessário saber qual a porta de comunicação onde será realizada o envio e coleta de dados. Por definição, porta é um software de aplicação específica ou processo específico servindo de ponto final de comunicações em um sistema operacional hospedeiro de um computador. O propósito das portas é identificar aplicações e processos de um computador e assim possibilitá-los a compartilhar uma única conexão física com uma rede de comutação de pacotes, como a internet. (DONAHOO; CALVERT, 2009)

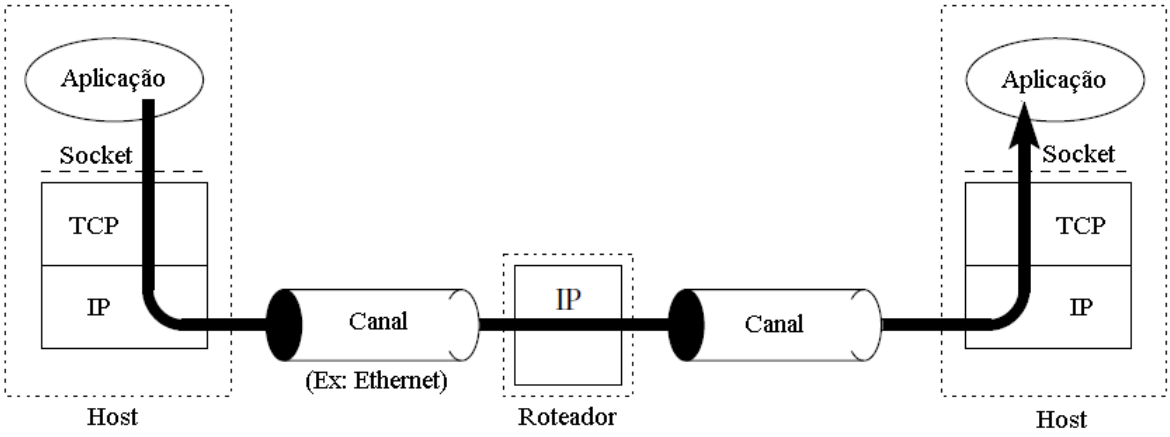
2.2- Camadas de comunicação e o protocolo TCP/IP

Na comunicação é essencial que haja uma padronização no tipo de informação a ser trocada entre as aplicações, uma vez que essas aplicações não são necessariamente executadas na mesma linguagem. Esse conjunto de regras e normas são denominados de protocolos de comunicação, e definem regras no envio de pacotes (DONAHOO, M. J.; CALVERT, L. K, 2009). A figura 3 ilustra o caminho percorrido por um pacote de dados entre duas aplicações TCP/IP.

O modelo ou arquitetura TCP/IP de encapsulamento busca fornecer abstração aos protocolos e serviços para diferentes camadas de uma pilha:

- Camada de Rede – camada física pois trata-se das tecnologias usadas para as conexões. Exemplo: Ethernet, Wireless, Modem, etc.
- Camada de Internet – Responsável pelo empacotamento dos dados e o seu roteamento. Exemplo: IP
- Camada de Transporte – Responsável pelo controle da comunicação host a host. Exemplo: TCP e UDP.
- Camada de Aplicação – Contém todos os protocolos para um serviço específico de comunicação de dados em um nível de processo a processo (por exemplo: como um web browser deve se comunicar com um servidor da web). Exemplo: HTTP, DHCP, Telnet.

Figura 3- Uma rede TCP/IP



Fonte: DONAHOO (2009).

2.3- Servidores e Clientes

Servidores e Clientes são teorias usadas para realizar a conexão entre dois *hosts* ou dois simples programas via rede. É chamado de servidor a aplicação que vai aguardar um pedido de conexão. Um servidor é responsável por se preparar e aguardar até que algum outro programa tente se comunicar com ele. Uma vez que ele só tenha que aguardar um pedido de conexão, não é necessário que ele saiba o endereço do computador que vai pedir a conexão. Um servidor pode ter vários clientes conectados a ele ao mesmo tempo. Um exemplo são os jogos de MMORPG online, onde vários jogadores (clientes) acessam o jogo (presente no servidor) ao mesmo tempo.

Cliente, por outro lado, é o programa que vai solicitar uma conexão e portanto é fundamental que ele saiba o endereço do servidor para poder pedir a conexão. Quando o cliente pedir a conexão e o servidor aceitá-la, o envio de dados pode acontecer em ambos os sentidos, isto é, do servidor para o cliente ou do cliente para o servidor.

2.4- *Socket*

Um *socket* é uma abstração através do qual uma aplicação pode enviar e receber dados, da mesma maneira que um arquivo aberto permite que uma aplicação escreva e leia dados da memória (DONAHOO; KENNETH, 2009).

Sockets são uma maneira eficiente de se enviar dados entre hospedeiros usando as portas do computador. A vantagem do uso dos *sockets* neste projeto é a capacidade de transmissão de dados à distância,

uma vez que eles se comunicam por rede. Dentre as funções usadas na programação via *socket*, a lista abaixo mostra as essenciais para uma comunicação TCP/IP:

- Bind(): função que acopla o *socket* previamente criado em uma porta desejada. Na comunicação TCP/IP , essa função é usada no servidor..
- Listen(): Coloca o *socket* em modo de escuta, preparando-o para receber pedidos de comunicação de outros *sockets.*
- Accept(): Aguarda por uma comunicação. Se o *socket* for bloqueado, essa função trava o programa até que algum programa peça por uma comunicação. Caso contrário retorna um erro se não tiver comunicação pendente. Usado no programa servidor logo após Listen();
- Connect(): Função que acopla o *socket* em uma porta aleatória livre do computador e faz o requerimento de uma conexão. Retorna SOCKET_ERROR caso a conexão falhe.
- Close(): Função que encerra o *socket* e limpa a memória alocada por ele.

2.5- Reabilitação robótica e os jogos sérios

A reabilitação é uma área da saúde que busca reintegrar uma pessoa, com uma ou mais funcionalidades prejudicadas, à sua atividade de vida diária. Acidentes e doenças, por exemplo, podem levar à perda parcial de movimentos, que dificultam uma pessoa de continuar sua rotina. Através de técnicas específicas, a reabilitação busca auxiliar na recuperação dos pacientes.

Normalmente, os procedimentos de reabilitação tradicionais são realizados com exercícios repetitivos, que tendem a diminuir a motivação do paciente. Além disso, estes procedimentos não possuem um monitoramento informatizado, o que não proporciona dados armazenáveis para análise (APPEL,2014; BURDEA, 2002). Por fim, "as sessões de reabilitação requerem intenso contato do terapeuta com o paciente durante todo o tempo de retreinamento" (APPEL, 2014).

A reabilitação robótica introduz os robôs às sessões de reabilitação. Segundo Robertson, Jarrassé e Roby-Brami (2010), um robô é uma máquina capaz de se adaptar ao ambiente extendendo ou substituindo habilidades humanas em algumas atividades. Os robôs são compostos de uma estrutura mecânica com diversos mecanismos e um certo número de graus de liberdade, e possuem sensores capturando informações de seu estado atual e do ambiente ao seu redor e que permitem a execução de uma tarefa corretamente. Na reabilitação robótica, os robôs são chamados de cooperativos, pois existe contato físico entre o robô e o paciente (APPEL, 2014).

Uma vantagem da reabilitação robótica é a capacidade de medir e armazenar dados do paciente sob a forma de grandezas físicas como posição, velocidade e força. Dessa forma, é possível avaliar o desempenho do paciente através de comparações das medidas ao longo do tempo de forma visual e automática (APPEL, 2014). Além disso, outra vantagem é que devido à característica colaborativa dos robôs na reabilitação, é

possível alterar os parâmetros do robô para modificar seu grau de assistência ao paciente, e esses parâmetros podem ser alterados diretamente pelo terapeuta. Por fim, a robótica na reabilitação permite a inserção de jogos sérios à terapia.

Zyda (2005) define jogos sérios como "uma disputa mental, jogado com um computador de acordo com as regras, que utiliza entretenimento como incorporação em treinamentos, educação, saúde, políticas públicas e objetivos de comunicação estratégica". Em outras palavras, jogos sérios são jogos cuja finalidade não é o entretenimento, mas utiliza da diversão proporcionada pelos jogos para outros objetivos.

No caso da reabilitação robótica, os jogos sérios auxiliam no processo de reabilitação trazendo benefícios ao paciente e ao terapeuta. Os jogos podem aumentar a motivação do paciente criando desafios que precisam ser concluídos para se avançar na terapia, além de mascarar o processo repetitivo e as vezes tedioso da reabilitação. Por fim, os jogos sérios geram dados para o terapeuta analisar o desempenho do paciente na reabilitação (ZYDA, 2005).

Dentre as pesquisas de jogos na reabilitação, podemos citar o jogo Roll the Ball. ANDRADE et al (2013) introduz e explica o funcionamento deste jogo, criado no ambiente Unity3D, e dos dispositivos utilizados para reabilitação robótica de punho. Desenvolvido no Laboratório de Mecatrônica da USP de São Carlos, o jogo e os dispositivos se comunicam por protocolo TCP/IP (ANDRADE et al, 2013), mesmo protocolo de comunicação utilizado por este trabalho.

Capítulo 3 – Materiais e Métodos

Este capítulo introduz e explica os materiais utilizados neste trabalho e, posteriormente, o método de comunicação é explicado.

3.1 – O robô do tipo SCARA

Para desenvolvimento da parte experimental foi utilizado o braço robótico SCARA pertencente ao laboratório de Mecatrônica da Escola de Engenharia de São Carlos. “A sigla SCARA é proveniente da descrição do conceito construtivo e graus de liberdade presentes neste manipulador: Selectively Compliant Assembly Robot Arm” (FERNANDES, 2013).

O manipulador, mostrado na figura 4, é composto de quatro juntas, sendo três de rotação e uma de translação. Três juntas possuem eixos que rotacionam de forma perpendicular ao chão, ou seja, na horizontal, e o quarto eixo realiza a translação do TCP na vertical com 200 mm de curso. Cada eixo é acionado por um motor distinto e portanto cada junta pode ser operada separadamente.

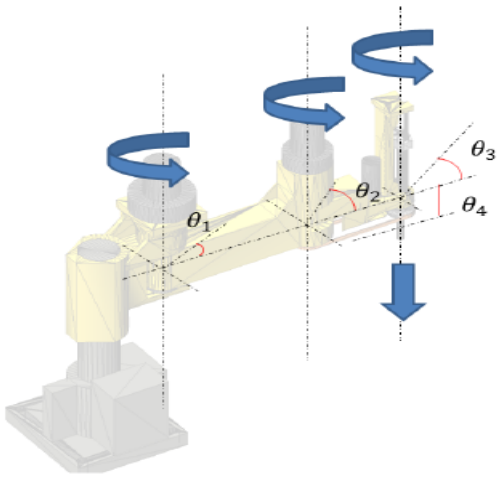
Figura 4 - O SCARA.

Fonte: FERNANDES (2013)

A figura 5 apresenta os graus de liberdade do SCARA. A primeira junta, responsável pela rotação do primeiro braço do sistema no plano x–y tem seu ângulo definido entre $0º < \Theta_1 < 200º$ e é acionado através de uma relação de transmissão de 1:157. O comprimento do primeiro braço é 400 mm. O segundo braço do manipulador possui limite de movimento entre $0º < \Theta_2 < 135º$ e apresenta uma redução de 1:80. O comprimento do segundo braço é de 250 mm. A terceira junta é responsável pela rotação do eixo do TCP,

que é limitado entre $-180º < \Theta_3 < 180º$ e pode ser acionada sem a influência dos movimentos do primeiro e segundo manipuladores. Por fim, o último eixo, responsável pelo movimento de translação do TCP na direção vertical, é acionado através de um fuso de esferas de passo 5 mm e possui um deslocamento total de 200 mm (FERNANDES, 2013). O espaço de trabalho do SCARA é ilustrado na figura 6.

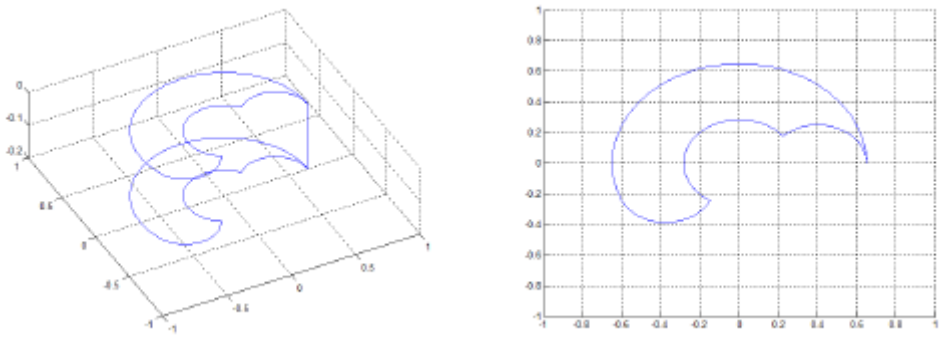
Figura 5- Ilustação dos ângulos de movimento do manipulador.



Fonte: FERNANDES (2013)

Outra característica deste manipulador é o sensor de força que está acoplado em seu TCP. O transdutor é mostrado na figura 7. Ele é composto de seis extensômetros espaçados entre si no mesmo plano e alocados da forma que é possível encontrar o vetor de forças e torques descritos no sistema de referência do centro do transdutor (FERNANDES, 2013).

Figura 6 - Representação do espaço de trabalho do SCARA.



Fonte: FERNANDES (2013)

O sensor têm os sinais de seus extensômetros amplificados por um condicionador de sinal, que disponibiliza seis sinais de amplitude entre -10 V e 10 V com cada um relativo a um extensômetro. Esses

sinais são levados a um bloco de I/O da PHOENIX CONTACT, cujo em seu módulo estão ligados três cartões de entrada analógica com conversores de 16 bits (FERNANDES, 2013).

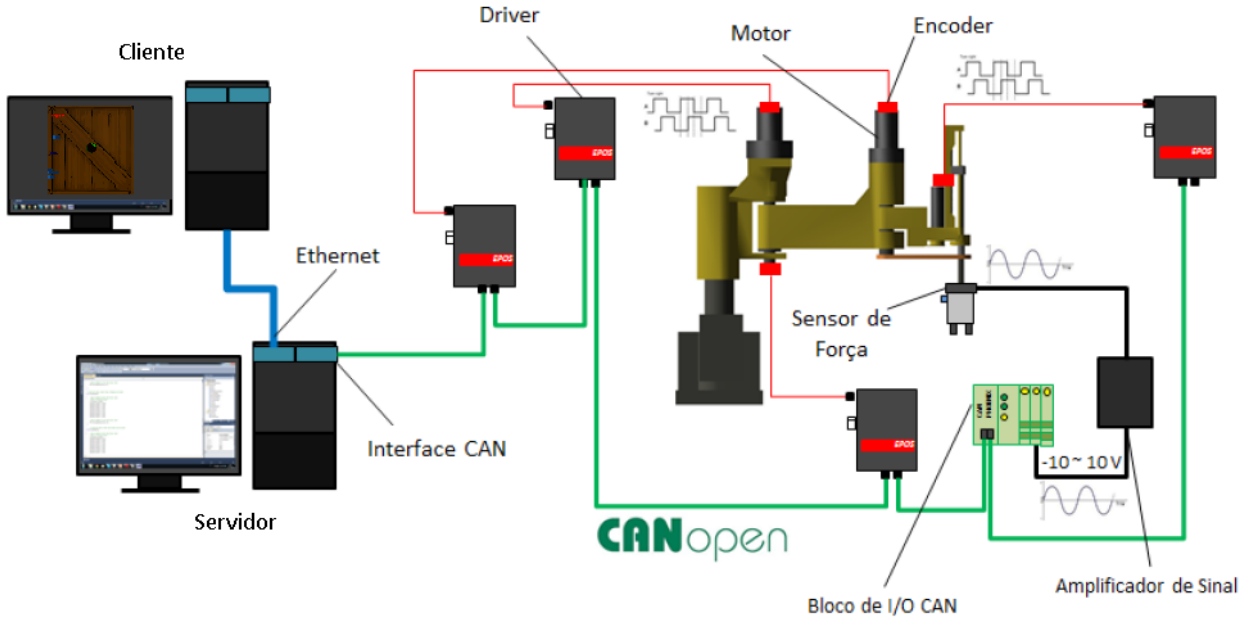
Cada eixo do SCARA possui um *encoder* que mede o ângulo atual do respectivo eixo, e seus sinais são levados para o computador via USB, RS232 ou CAN/CANOpen. Além disso, o manipulador utilizado pertencente ao Laboratório de Mecatrônica não possui sensores de fim de curso, apenas um botão de emergência que corta a corrente de todos os motores.

Figura 7- Transdutor de força em seis eixos acoplado ao TCP do manipulador.

Fonte: FERNANDES (2013)

Cada servomotor do SCARA é controlado por um servodriver da marca MAXON do tipo EPOS2 70/10. Os servodrivers estão conectados entre si via CAN e um deles está conectado ao computador via CAN também. O sensor recebe a força de forma analógica, passa para o amplificador de sinal e depois para o conversor de 16 bits, que entrega o sinal digitalmente para os servodrivers (FERNANDES, 2013). A figura 8 apresenta uma arquitetura geral do robô com ênfase nas tecnologias utilizadas para comunicação entre os dispositivos apresentados.

Figura 8- Visão geral dos componentes do sistema SCARA.



Fonte: FERNANDES (2013). Adaptado

3.2- Controle de impedância

Para o sistema de controle do manipulador foi utilizado um controle de impedância escrito por Guilherme Fernandes, que "consiste em controlar a relação dinâmica existente entre força e posição em vez de controlar somente a força ou somente a posição" (FERNANDES, 2013; HOGAN, 1985). Este programa é responsável por realizar a movimentação do braço robótico de acordo com a força aplicada no sensor e armazenar as informações dinâmicas em variáveis facilmente acessíveis. Entre essas informações, encontra-se a posição e velocidade do TCP (Tool Center Point) no sistema de coordenadas 3D e a força e torque, capturados pelo extensômetro em tempo real.

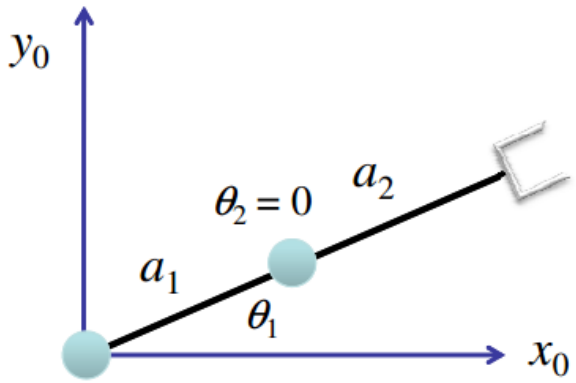
Todas as informações do manipulador são armazenadas em variáveis globais e atualizadas constantemente durante a execução do controle de impedância. Porém, a introdução de novos comandos logo após o controle não foi possível, uma vez que novos comandos implica num gasto de tempo de execução maior que 1 ms por junta, alterando o período do controle e comprometendo toda sua estrutura. A solução encontrada foi acessar essas variáveis em um processo paralelo, copiando os dados dinâmicos para uma variável local para que pudessem ser transmitidos via rede.

Com relação ao controle de impedância, a velocidade V do TCP pode ser relacionada com o vetor de velocidades das juntas através da equação 3.1, sendo J(θ) a matriz Jacobiana formada pelas derivadas parciais do ângulo das juntas em relação à coordenada cartesiana.

$$V = J(\Theta) * \dot{\Theta} \tag{3.1}$$

Para um manipulador com duas juntas como mostrado na figura 9, o Jacobiano é determinado pela equação 3.2 e o determinante do Jacobiano é representado na equação 3.3:

Figura 9- Ilustração de um manipulador robótico ccom duas juntas em série e alinhadas.



Fonte: BECKER (2008).

$$J(\Theta) = \begin{bmatrix} -a_1 * sen(\Theta_1) - a_2 * sen(\Theta_1 + \Theta_2) & -a_2 * sen(\Theta_1 + \Theta_2) \\ a_1 * \cos(\Theta_1) + a_2 * \cos(\Theta_1 + \Theta_2) & a_2 * \cos(\Theta_1 + \Theta_2) \end{bmatrix} \tag{3.2}$$

$$det(J) = a_1 * a_2 * sen(\Theta_2) \tag{3.3}$$

Para a situação de alinhamento das juntas, o determinante da matriz Jacobiana se torna nulo, caracterizando um ponto de singularidade. Em torno deste ponto, pequenas velocidades no espaço de trabalho podem resultar em alta velocidade no espaço das juntas (BECKER, 2008), tornando o controle de impedância impreciso.

Por fim, devido a um acidente, a primeira junta se encontra mais rígida do que antes, o que desregula o controle de impedância da primeira junta e consequentemente o resto.

3.3- O Unity3D

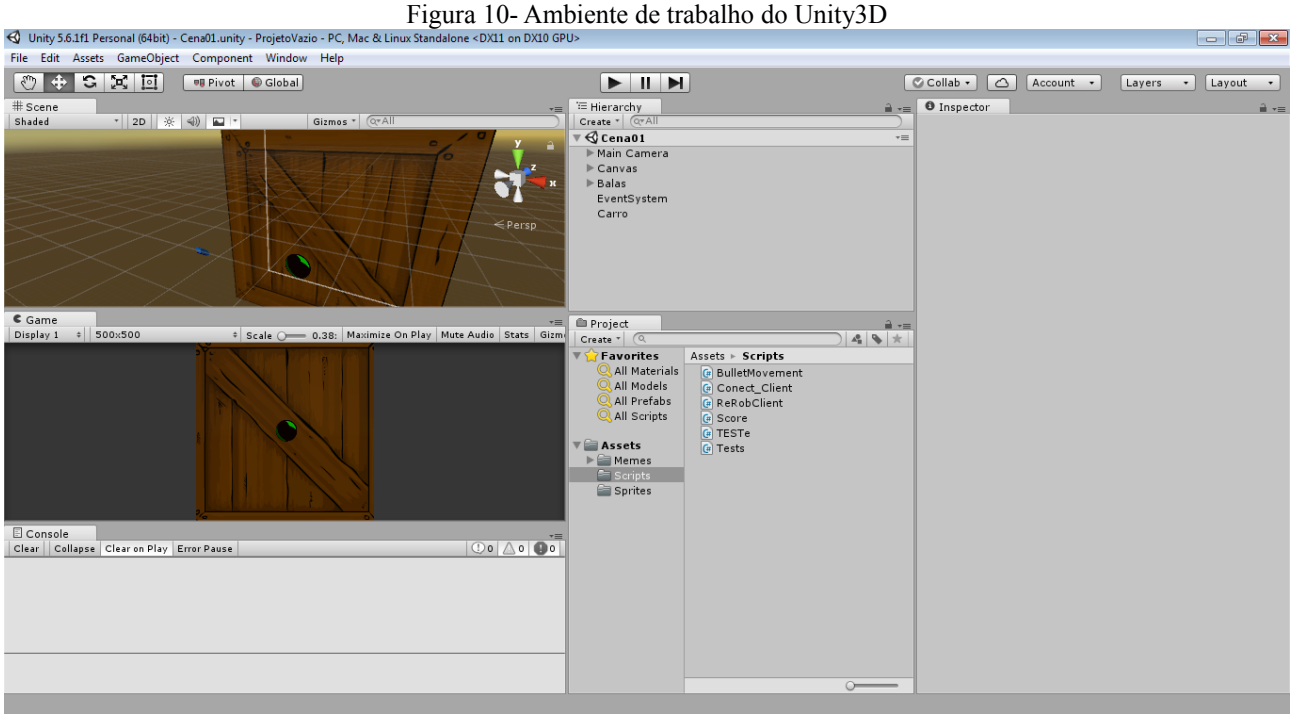
Unity3D é um ambiente de desenvolvimento de jogos, tanto 2D quanto 3D. Ele conta com uma grande comunidade online, que compartilham informações sobre a ferramenta, e também com uma enorme quantidade de recursos em uma galeria de exemplos e modelos, onde artistas, modeladores, programadores e desenvolvedores de jogos disponibilizam seus trabalhos para serem usados nesse ambiente. A versão 5, última versão lançada tem como grande vantagem a possibilidade de exportar código executável para diferentes sistemas operacionais como Android, iOS, Windows, Linux, entre outros. Essa característica em especial levou a sua escolha. O ambiente de trabalho do Unity3D é ilustrado na figura 10.

O Unity3D conta com um ambiente de programação, chamado *Scripting*, construído sobre o *framework* MonoDevelop que é *open-source*. Graças a esse ambiente, a Unity3D possui suporte às linguagens JavaScript, C# e Boo (que tem uma sintaxe inspirada pela linguagem Python). Logo, esse ambiente permite ao programador utilizar mais de uma linguagem, sendo prático e versátil (PIRES, 2014).

Todos os scripts criados e usados no Unity3D possuem como padrão duas funções de programação: a função Start que é chamada no início do processamento do Script, geralmente usada para inicialização de variáveis, e a função Update, que é chamada após o Start e executado uma vez a cada frame.

Esse trabalho utilizou a linguagem C# para a programação do cliente, pois é a linguagem padrão escolhida para projetos em Unity3D no Laboratório de Mecatrônica.

Figura 10- Ambiente de trabalho do Unity3D

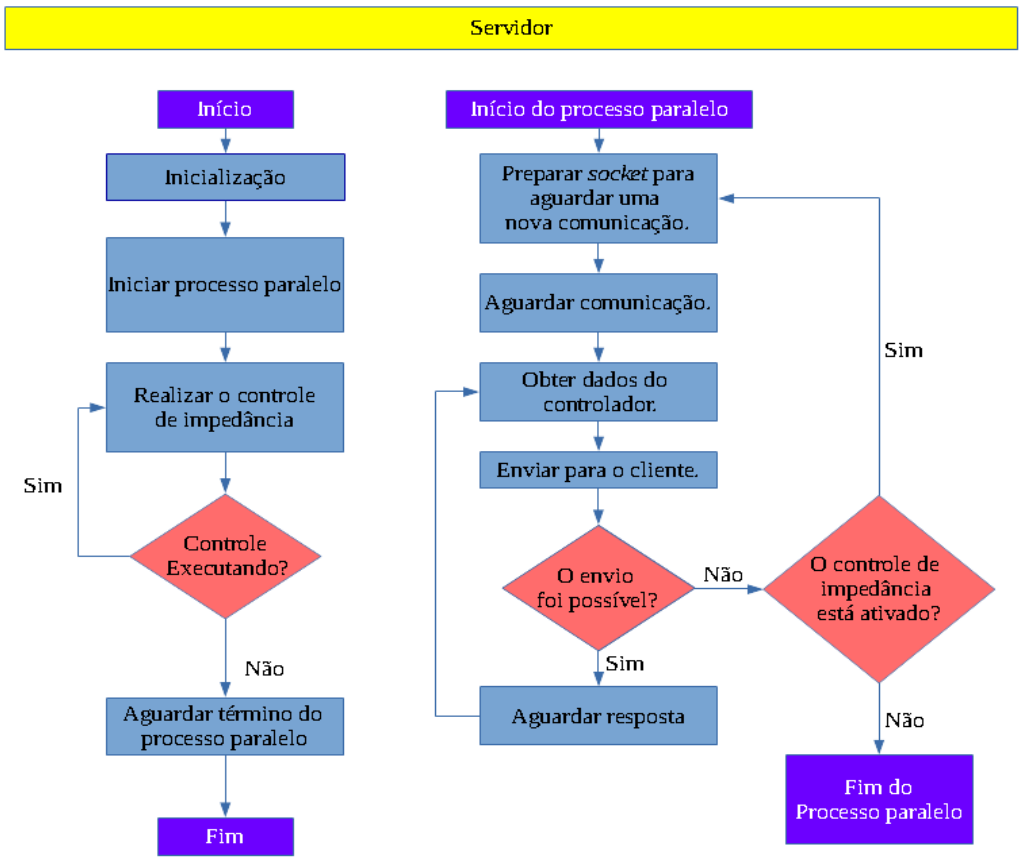


3.4- Funcionalidade do Software

O software de comunicação criado neste trabalho é dividido em dois programas separados e trabalhando ao mesmo tempo, que podem ou não serem executados em diferentes computadores, desde que estejam na mesma rede.

O programa servidor deste trabalho foi desenvolvido em C/C++ e incluído no programa de controle de impedância previamente escrito por Guilherme Fernandes (FERNANDES, 2013) como um processo paralelo visando interferir minimamente no tempo de execução do controle. O servidor criado tem como característica a capacidade de se preparar para uma nova comunicação caso a conexão com o cliente se perca. A figura 11 mostra o fluxograma do servidor.

Figura 11- Fluxograma do servidor.

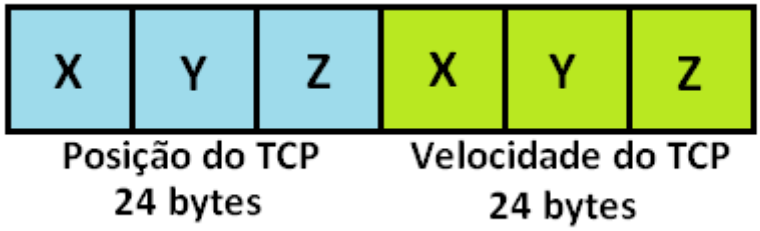


A escolha da inserção do programa servidor no controlador do SCARA, e não no jogo, visa simplificar o desenvolvimento de jogos, uma vez que a inicialização do controle de impedância é mais demorada. Com a inserção do servidor no controle de impedância, não é necessário desligá-lo caso o desenvolvedor encontre alguma falha em seu jogo., basta apenas desligar o jogo.

Ao iniciar o controle de impedância, um processo paralelo em separado é criado para gerenciar o servidor. Primeiro, é criado um *socket* do tipo stream, usado para comunicações TCP/IP, e conectado o *socket* na porta lógica 8888 do computador, que foi escolhida arbitrariamente. Caso seja necessário, o número da porta pode ser alterado. Por fim o *socket* é configurado para modo de escuta e permanece em espera, aguardando algum outro *socket* se comunicar com ele.

Caso algum programa peça conexão a esse *socket*, ela é aceita. O servidor captura os dados dinâmicos do controle de impedância, isto é, a posição e velocidade do TCP do manipulador contidos em variáveis do tipo *double*, e faz uma cópia destes dados em um vetor de bytes utilizando a função memcpy (do inglês *memory copy*). O dado, ilustrado na figura 12, é enviado via *socket* para a camada TCP e posteriormente para a IP onde o dado é empacotado e enviado para a rede até chegar ao cliente. A quantidade de bytes enviados ao cliente é retornada ao servidor para confirmação. Caso o envio do pacote de dados falhar, o servidor declara que a conexão com o cliente foi perdida. O *socket* é então reconfigurado para aguardar até que uma nova conexão seja realizada. Essa capacidade de reconfiguração é feita automaticamente.

Figura 12- o dado encapsulado em um pacote TCP/IP é um vetor de bytes com os dados de posição e velocidade do TCP do SCARA e comprimento total de 48 bytes.



O programa cliente deste trabalho foi escrito na linguagem C# e é inserido no algoritmo do jogo para interagir com o manipulador. Para o software cliente, foi criada uma API em forma de classe para que o desenvolvedor possa escolher como deseja realizar a comunicação, sempre respeitando o funcionamento do servidor. Dentre as funções criadas, inclui a função de conectar ao servidor, funções de enviar e receber pacotes via protocolo TCP/IP, uma função que converte as informações recebidas como vetor de bytes em números racionais e funções de verificação. Um exemplo de uso das funções encontra-se na figura 13.

Ao instanciar a classe, um novo *socket* do tipo stream é criado. O *socket* não é conectado a nenhuma porta e só pode ser usado caso esteja conectado ao servidor.

A função de conectar ao servidor, nomeada como *SCARA_Connect*, recebe como parâmetro um texto, que deve conter o IP da máquina do servidor na rede, sob a forma de IPv4, e o número da porta lógica que o servidor está aguardando a conexão. Uma vez chamada essa função, o *socket* é acoplado a uma porta lógica aleatória e então faz o requerimento de conexão para o servidor de acordo com o endereço IP e porta

dados. Independente do sucesso ou fracasso da tentativa de conexão, a função retorna nada. A verificação da conexão deve ser feita utilizado a função *is_connected*.

Figura 13- Exemplo de uso da classe do cliente.

```
Thread oThread  = null;                 //Criação de uma variável do tipo thread
ReRobClient cli = new ReRobClient();    //Criação de uma instância da classe de comunicação


//****************************************
//Função chamada separadamente via THREAD.
//Faz o recebimento e envio de dados para o servidor
//****************************************
public void gama()
{
    //Loop
    do
    {
        cli.SCARA_Receive(ref buff, true); //Aguarda um pacote de dados
        if (cli.SCARA_MessageReceived())   //Verificação de erro
        {
            //Extrai a posição do TCP do dado e coloca numa variável
            cli.ExtractV3 (ref SCARA_position, buff, 0);

            //Permite o movimento do player e o uso do dado coletado
            movePlayer = true;

            //Envia uma resposta de confirmação de recebimento para servidor
            cli.SCARA_Send ("RIGHT", 5);
        }
    } while (cli.SCARA_MessageSent () && cli.SCARA_MessageReceived () && !stopgame); //Verificação de erro
    Debug.Log ("Thread encerrada");
}
//****************************************
//****************************************


//INICIALIZA VARIAVEIS E INICIALIZA A THREAD
void Start()
{
    cli.SCARA_Connect (Constants.IP, Constants.port);   //Tenta uma conexão com o Servidor
    if (cli.SCARA_isConnected ())                       //Caso a conexão for bem
    {
        // Cria uma instância da thread de controle de fluxo de dados.
        //partindo do método gama
        oThread = new Thread(new ThreadStart(this.gama));
        //Inicia a thread
        oThread.Start();
        //Aguarda o início da thread
        while (!oThread.IsAlive);
    }
    else{                                               //Caso a conexão falhou
        Debug.Log ("Não foi possível conectar ao servidor");//Mostra mensagem de erro ao desenvolvedor
        UnityEditor.EditorApplication.isPlaying = false;    //Encerra a aplicação
    }
}
```

A função de receber um pacote de dados, nomeada como *SCARA_Receive*, possui em um de seus parâmetros a possibilidade de bloquear ou desbloquear o *socket* usado na comunicação. Um *socket* em modo desbloqueado impede que a função aguarde o servidor enviar o dado. Caso a função seja chamada em modo de *socket* desbloqueado e o servidor não está enviando dado algum, a função retorna.

A função *SCARA_Send* envia uma mensagem ao servidor codificada em código ASCII. Ela é usada para manter o servidor atualizado com as informações do jogo, e também manter um controle na transmissão, uma vez que o servidor só envia o próximo dado caso receba uma resposta do cliente. Para verificar se as funções de receber e enviar foram bem-sucedidas, é necessário utilizar as funções *was_sent* e *was_received* que verificam, respectivamente, se a última função *SCARA_Send* enviou a mensagem e se a última função *SCARA_Receive* recebeu algum dado.

Por fim a função de extração de dados, nomeada de *ExtractV3*, captura os números contidos no pacote enviado pelo servidor e os inserem em vetores de 3 dimensões comumente utilizados por desenvolvedores no Unity3D.

Todo o trabalho foi documentado para permitir a reprodução do mesmo ou para a criação de novas aplicações. Ele inclui composição do maquinário, explicação e detalhes do modo de operação, solução de prováveis erros ou problemas, explicação dos programas usados, e cuidados a serem tomados visando à integridade do equipamento e de seus usuários. O Manual encontra-se no apêndice A.

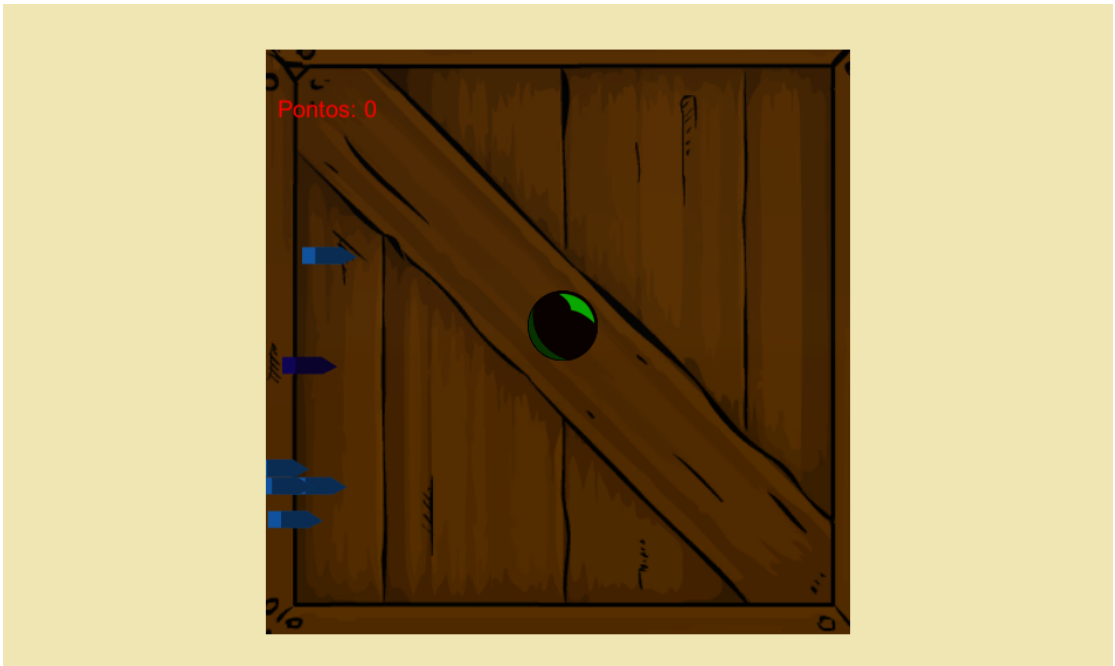
Capítulo 4- Experimento

Os objetivos do experimento envolveram validar o estado da comunicação e a capacidade de reconfiguração do servidor caso a comunicação com o cliente fosse perdida, como no caso do jogo ser desligado. Nesse teste o braço robótico se move livremente e não impõe restrição ao movimento do jogador. Além disso, o teste foi realizado com a máxima velocidade de transmissão de dados possível entre cliente e servidor.

4.1- O jogo "Capture o Azul"

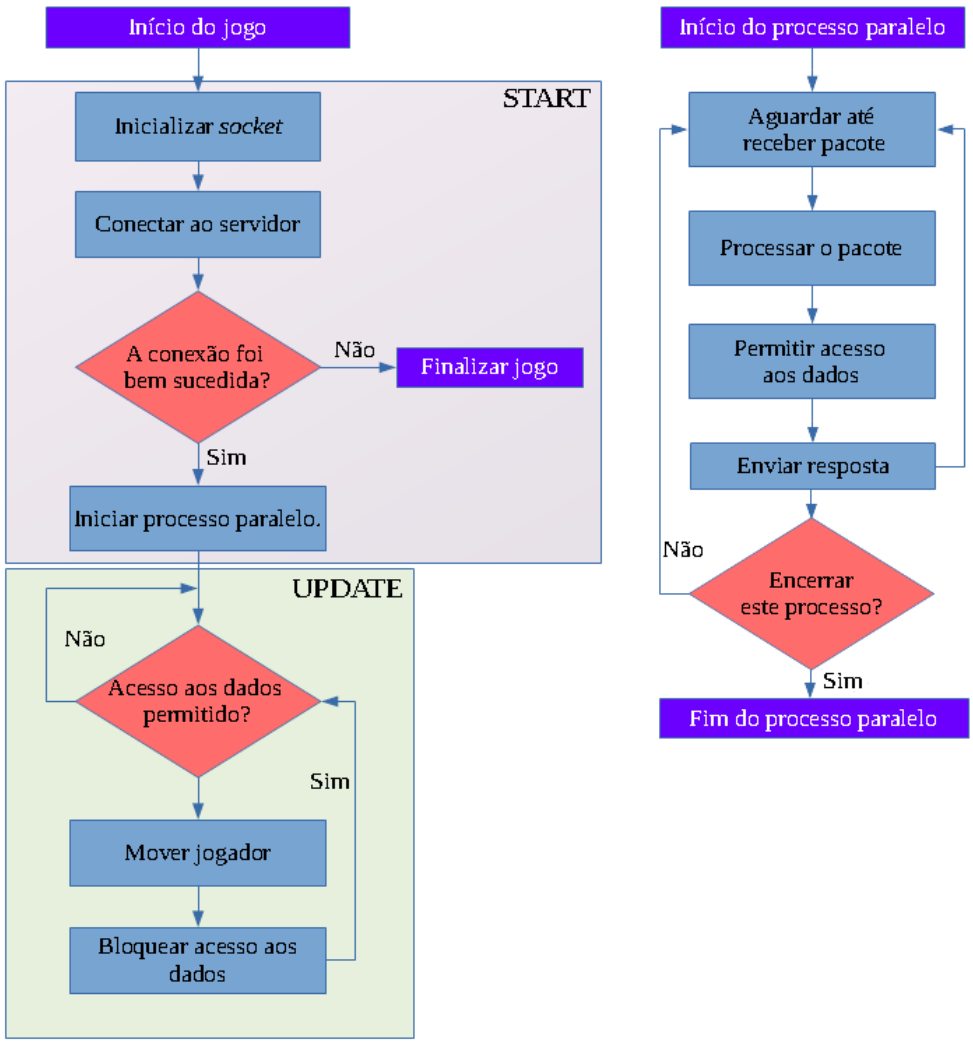
O jogo Capture o Azul visualizado na figura 14 foi criado no Unity3D versão 5, e pertence ao estilo 2D com perspectiva de cima. Nele o jogador, utilizando o SCARA, comanda um elemento (bolinha verde na figura) que deve interceptar os objetos azuis para coletá-los, estes objetos se movem da esquerda para a direita com diferentes velocidades. A cada coleta, um ponto é ganho. O jogador pode se mover em todo o plano do jogo e não há nada que impeça seu movimento. Grandezas como atrito, gravidade, força ou momento linear não foram incluídas. Para esse jogo usou-se apenas os dados de posição X e Y, descartando a posição Z e os valores de velocidade contidos no dado recebido do servidor.

Figura 14- Foto retirada do jogo Capture o Azul.



A figura 15 mostra o fluxograma do cliente criado para esse jogo usando a API.

Figura 15- Fluxograma do cliente usado no Capture o Azul. A parte START do fluxograma é executada no início do jogo e a UPDATE é executada uma vez a cada frame.



O experimento foi realizado com uma rede com velocidade de transmissão de 100 Megabits por segundo. Nele, foi inicializado o servidor e em seguida o jogo com o programa cliente. O jogo permaneceu ligado por 120 segundos, contados com um cronômetro. Para o cálculo do número de mensagens recebidas foi adicionado uma variável do tipo inteiro no programa cliente agindo como contador, de forma que a cada mensagem recebida do servidor este contador é incrementado em 1. Ao fim dos 120 segundos, o cliente foi desligado e observou-se através do contador que o número total de mensagens recebidas pelo cliente foi aproximadamente 13,7 mil. Dividindo este número pela duração do experimento, temos que o número médio aproximado de mensagens recebidas por segundo foi de 115 fps.

Para cálculo da taxa de atualização do jogo ou fps (frames per second), utilizou-se a função *stats* presente no Unity3D. Durante a execução do jogo, observou-se que seu fps atingia picos de 80 e mínimos de 60. Esses valores mostram que mesmo que a comunicação entre servidor e cliente ocorra a uma taxa de 115 segundos, no máximo 80 mensagens serão utilizados para movimentar o objeto presente no jogo (bolinha verde). Logo, a comunicação mostrou-se satisfatória uma vez que a quantidade de dados enviadas é superior à utilizada, permitindo que a movimentação do jogador seja atualizada em todos os frames. Por fim, vale destacar que todos os pacotes de dados enviados pelo servidor chegaram ao cliente, isto é, não houve perda de pacote na rede.

Capítulo 5- Conclusões finais

Este trabalho utilizou o protocolo de comunicação TCP/IP para criar uma ferramenta de comunicação entre o software de controle de impedância que movimenta o braço robótico do tipo SCARA pertercente ao Laboratório de Mecatrônica e o ambiente de desenvolvimento de jogos Unity3D. A ferramenta é um sistema composto de dois softwares distintos.

O primeiro software, o servidor, escrito em C/C++, está inserido no software de controle de impedância do robô SCARA previamente criado por Guilherme Fernandes (FERNANDES, 2013), ex-aluno de mestrado do Laboratório de Mecatrônica da USP. O software servidor é responsável por aguardar uma comunicação de um cliente e caso uma comunicação anterior é terminada, reconfigurar os *sockets* para aguardar uma nova comunicação.

O segundo software é composto de uma API escrita em C# contendo funções de cliente TCP/IP para gerir uma comunicação com o servidor. A criação de uma API de comunicação foi de importante para dar liberdade ao desenvolvedor de games ao criar um jogo para utilizar a comunicação.

O teste de comunicação realizado com a ferramenta criada neste trabalho mostrou resultados satisfatórios. Durante um período de teste de 120 segundos, nenhum pacote de dado trocado entre servidor e um cliente se perdeu na rede, cliente este que foi desenvolvido com a API criada. Além disso, a taxa de envio de dados do servidor para o cliente chegou a 115 mensagens por segundo, e portanto satisfatória para a utilização desses dados no movimento de um objeto em um jogo do Unity3D, uma vez que a taxa de atualização do Unity3D é menor, cerca de 80 mensagens por segundo. Portanto, esses dados mostram que a escolha do protocolo TCP/IP para este projeto cumpriu com os objetivos propostos.

A ferramenta criada com este trabalho permite a expansão do uso do robô SCARA para a área de reabilitação robótica. Como um exemplo de um futuro trabalho, o TCP do manipulador contém um instrumento que segura um membro superior do corpo do paciente. Então, o robô realiza uma força que auxilia este membro a atingir uma posição específica no espaço para completar um desafio do jogo.

O entretimento proporcionado pelos jogos digitais e a robótica podem tornar a reabilitação mais prazerosa e motivadora para o paciente, e ainda cumprir com o principal objetivo do tratamento, o retorno do paciente às suas atividades de vida diária.

Capítulo 6- Referências Bibliográficas

ANDRADE, K. et al, **Rehabilitation robotics and serious games: An initial architecture for simultaneous players,** 2013 ISSNIP Biosignals and Biorobotics Conference: Biosignals and Robotics for Better and Safer Living (BRC), Rio de Janerio, 2013, pp. 1-6.

APPEL, V.C.R. **Classificando emoções em processos de reabilitação robótica.** 2014. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2014.

BECKER, M. **Jacobiano: Velocidades e Forças Estáticas,** Anotações de Aula da Disciplina "Dinâmica e Controle de Sistemas Robóticos I", Escola de Engenharia de São Carlos - Universidade de São Paulo, 2008.

BURDEA, G. Keynote address: virtual rehabilitation-benefits and challenges. In: INTERNATIONAL WORKSHOP ON VIRtUAL REALITY REHABILITATION (Mental Health, Neurological, Physical, Vocational), 2002. **Proceedings…** [S.1.:s.n], 2002. p.1-11.

CAURIN, G.A.P et al. **Adaptive strategy for multi-user robotic rehabilitation games.** In: Annual International Conference of the IEEE engineering in medicine and biology society, 2011, Boston. Proceedings… Piscataway: IEEE, 2011. p.1395-1398.

DONAHOO, M. J.; CALVERT, L. K. **TCP/IP Sockets in C: Practical Guide for Programmers.** Amsterdam: Morgan Kaufmann, 2009.

FERNANDES, G. **Exploração de ambientes não estruturados através de manipulador robótico implementando controlador de impedância com parâmetros variáveis.** Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos. 2013.

HOGAN, N. **Impedance control: An approach to manipulation: I,ii and iii**. Journal of dynamic system, measurement, and control, v. 107, n.2, p. 17, 1985.

KREBS, H. I. et al. **Rehabilitation robotics: performance-based progressive robot-assisted therapy.** Autonomous Robots,v. 15, n. 1, p. 7-20, 2003.

PIRES, F. A. **Avaliação de métodos de telereabilitação robótica utilizando comunicação TCP/IP e Unity.** Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos. 2014.

ROBERTSON, J.; JARRASSÉ, N.; ROBY-BRAMI, A. Rehabilitation robots: a compliment to virtual reality. **Schedae,** v.6, n.1, p. 77-94, 2010.

TANENBAUM, A. S. **– Redes de Computadores** – 4ª Ed., Editora Campus (Elsevier), 2003.

STUDY shows employees learn best from videogames. Universidade Denver, Colorado, 2010. Disponível em: <http://www.cudenvertoday.org/videogamesmakebetteremployees/>. Acesso em: 12 maio. 2017.

ZYDA, M. (2005) From visual simulation to virtual reality to games. Computer 38(9): 25-32. IEEE.

Apêndice A – Manual do SCARA

Manual do SCARA
do Laboratório de Mecatrônica da EESC/USP.

Autor Guilherme Rodrigues Chiqueti, com referências das anotações do Prof. Dr. Leonardo Marques Pedro e Guilherme Fernandes. Aqui você encontrará alguns cuidados e dicas para utilizar o SCARA do laboratório de Mecatrônica da ESSC e utilizar o software criado para comunicar o manipulador com o Unity3D.

Sumário

A1- Ligar e desligar o SCARA.....45

A2- Requisitos.....45

A3- O botão de emergência.....45

A4-Utilizando o controle de impedância.....46

A5- Usando o EPOS Studio para eliminar erros nas EPOS.....47

A6- Resolvendo problemas de captura de dados parciais pelo programa de controle de impedância.....47

A7- Utilizando o SCARA para jogos do Unity3D.....48

A8- Descobrindo o IP do servidor.....50

A9- Recomendações.....51

A1- Ligar e desligar o SCARA

O manipulador possui 3 tomadas de energia: Um para o acionamento dos motores, um para o acionamento da EPOS e outro para o bloco de I/O que converte os dados do sensor de força. As duas tomadas grandes devem ser ligadas no 220V, enquanto a do bloco do sensor em 127 V. Em seguida arme os dois disjuntores encontrados na parte central esquerda do painel para ligar. Para verificar se a ligação está correta, desarme o botão de emergência. O sucesso é validado se ouvir um estalo.

Para desligar o manipulador pressione o botão de emergência e desligue todos os programas que utilizam o robô. Então, desligue os dois disjuntores no painel e retire a tomada do sensor de força.

A2- Requisitos

O programa de controle de impedância foi executado em um computador do Laboratório de Mecatrônica, que possui os hardwares e softwares necessários instalados anteriormente.

Para uso do programa de controle de impedância

- Windows 7
- Visual Studio 2010
- Placa CAN instalada
- Bibliotecas Boost e Rapidjson instaladas

Para solução de prováveis erros

- BusMonitor
- Epos Studio
- Conexão USB

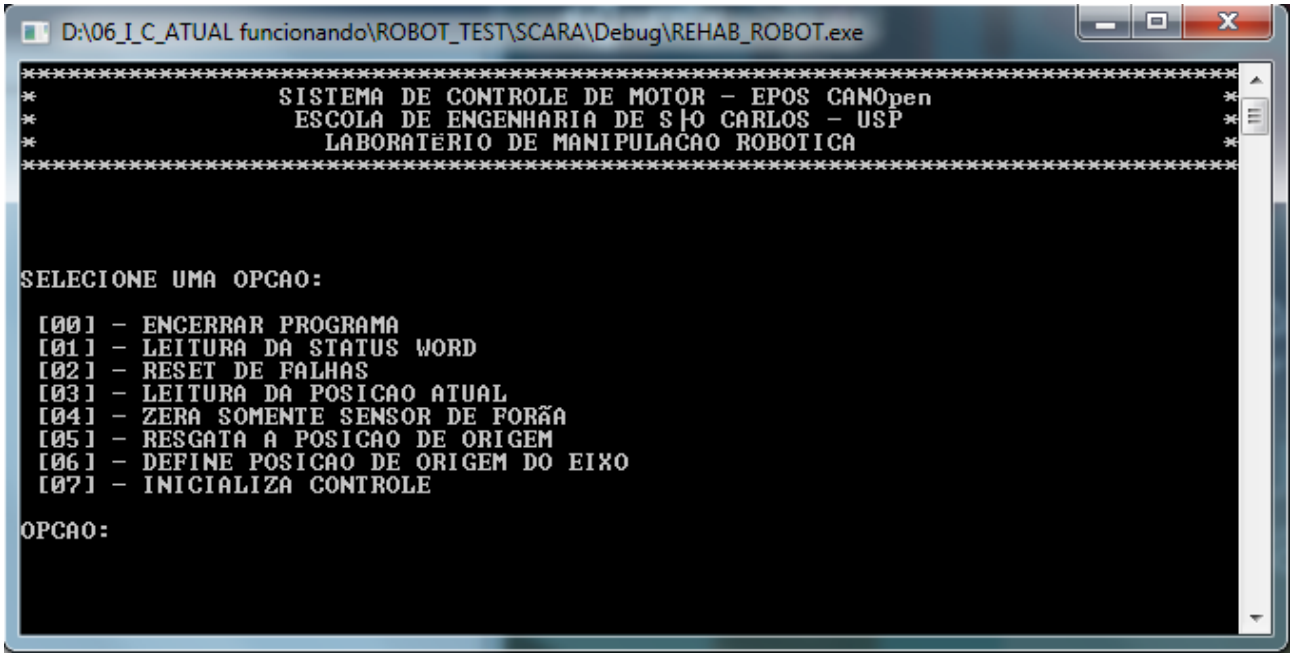
A3- O botão de emergência

O SCARA não possui sistema de fim de curso. Por isso, ao movimentar as juntas, deve-se tomar cuidado para que ele não bata nas laterais, o que pode danificá-lo. Por essa razão, ao comandar o braço robótico, mantenha sempre o botão de emergência em mãos e mantenha-se atento ao seu movimento.

A4-Utilizando o controle de impedância

Plugue os cabos CAN do painel do SCARA e do sensor na placa CAN do computador. Abra e execute o programa no Microsoft Visual Studio 2010. Após a inicialização das variáveis e a verificação da comunicação com a EPOS, é aberto o menu de opções ilustrado na figura.

Figura 16 - Interface do usurário do programa de controle de impedância.



Quadro 1- Descrição do programa controlador do robô SCARA

Commando	Função
[00] – ENCERRAR PROGRAMA	Termina o programa e libera as variáveis.
[01] – LEITURA DO STATUS WORD	Mostra o estado atual do hardware das EPOS e dos motores.
[02] – RESET DE FALHAS	Elimina as falhas deixadas por usos anteriores.
[03] – LEITURA DA POSIÇÂO ATUAL	Mostra informações de posição dos encoders e o torque e força que o sensor de força está capturando.
[04] – ZERA SENSOR DE FORÇA	Seta as forças atuais aplicadas ao sensor de força como sendo a origem.
[05] – RESGATA A POSIÇÃO DE ORIGEM	Utiliza como referência a última posição salva.
[06] – DEFINE POSIÇÃO DE ORIGEM DO EIXO	Redefine a origem da coordenada do TCP e salva essa posição.
[07] – REALIZA CONTROLE	Inicializa controle de impedância e o servidor.

Use o comando 4 para zerar o sensor de força. Essa função deve ser usada antes do início do controle de impedância para garantir seu correto funcionamento e evitar movimento irregular do SCARA. Uma falha comum envolve a não captura dos dados dos encoders 2, 3 e 4. Usando o comando "*LEITURA DA POSIÇÂO ATUAL*" do programa, mexa o SCARA manualmente e verifique se os valores de q2, q3 e q4 alteram. Se eles não se alterarem, desligue o programa e siga as instruções do sub-ítem *"A6- Resolvendo problemas de captura de dados parciais pelo programa de controle de impedância"* para resolver o problema.

Em seguida ligue o controle de impedância (comando 7). Mantenha sempre o botão de emergência em mãos caso o robô comece a agir desordenadamente. Nesse ponto o controle está funcionando e o servidor está prepara para receber a comunicação do jogo no Unity3D.

Evite rodar o programa de controle de impedância por mais de 5 minutos.

A5- Usando o EPOS Studio para eliminar erros nas EPOS

É possível verificar o estado das EPOS observando seus respectivos LEDs no painel.

- LED vermelho contínuo → EPOS com erro.
- LED verde piscando → EPOS sem erros, porém desabilitada.
- LED verde contínuo → EPOS habilitada e funcionando.

Para eliminiar algum erro, plugue a conexão USB do painel do robô SCARA no computador, abra o programa EPOS Studio e crie um novo projeto de modelo EPOS2 se já não estiver criado antes. Então clique em conectar todos para que o EPOS Studio conecte as EPOS. Se aparecer algum erro em qualquer das EPOS clique com o botão direito na EPOS com o erro e clique em *reset all errors*. Desconecte as EPOS e feche o EPOS Studio.

A6- Resolvendo problemas de captura de dados parciais pelo programa de controle de impedância.

Siga os passos abaixo:

- Abra duas instâncias do BusMonitor, sendo uma instância utilizando a CAN1 e outra a CAN2.
- Configure ambas as instâncias para uma taxa de transmissão de 500kbits.
- Inicie a CAN2.
- Usando a CAN2, transmita o dado: 01 00 00 00 00 00 00 00 para a CAN 1 com o Id: x00 e observe a resposta na instância da CAN2, que deve ter apenas 1 linha.
- Ainda usando a CAN2, transmita o dado: 00 00 00 00 00 00 00 00 para a CAN 1 com o Id: x80 e observe a resposta na instância da CAN2. O resultado final está ilustrado nas figuras 17 e 18. Então termine a instância da CAN2 e feche ambas as instâncias para terminar o processo.

Figura 17- Imagem da instância CAN1 após o término do processo.

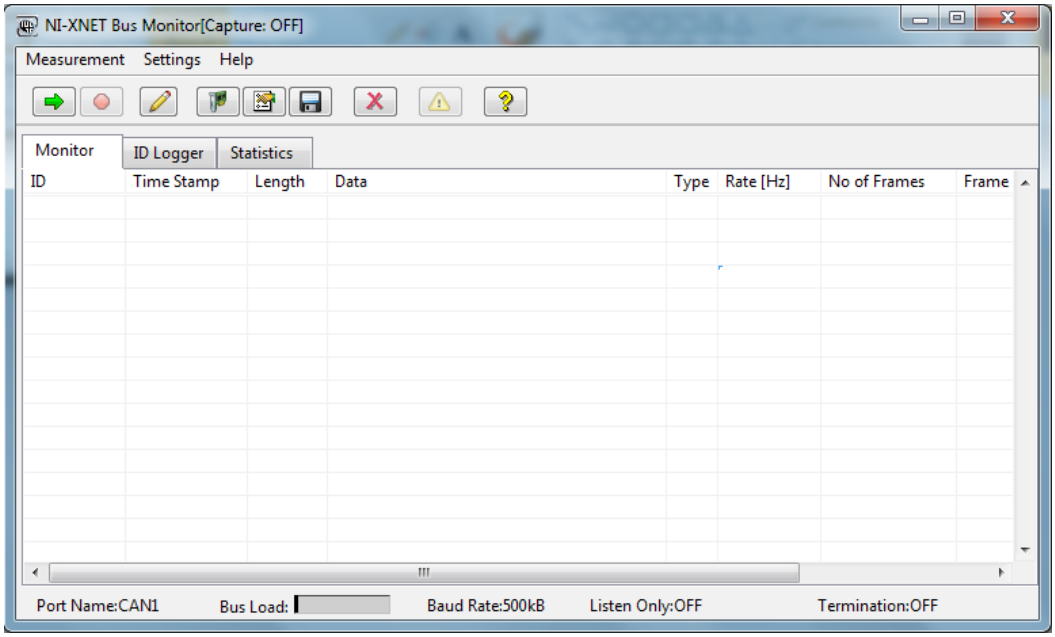
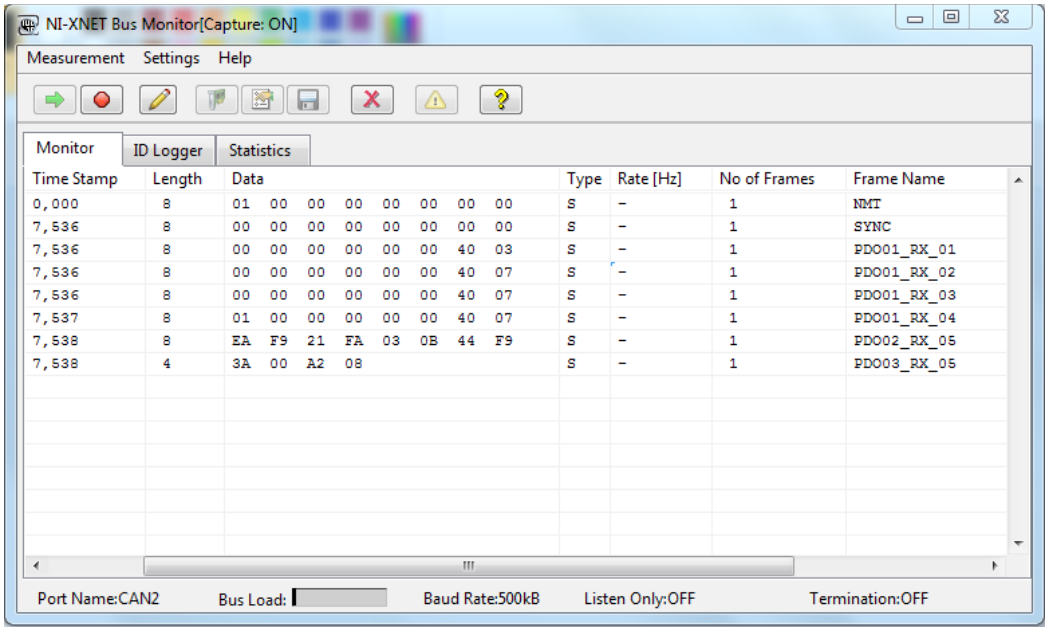


Figura 18– Imagem da instância CAN2 após o término do processo.



A7- Utilizando o SCARA para jogos do Unity3D.

Esse tópico é um passo a passo explicativo para o uso correto do programa cliente nos jogos. Inclui um jogo modelo com códigos em C# comentados. Para criar suas próprias aplicações, estude os *scripts*

ReRobClient, que é a classe responsável pela comunicação, e Conectar_Client, que utiliza a comunicação para movimentar a bolinha verde. A API criada para o cliente possui as funções descritas no quadro 2.

Quadro 2- Funções contidas na API para a utilização no Unity3D

Função	Descrição	Entrada(s)	Saída	Exemplo de uso
SCARA_Connect	Tenta uma conexão com um servidor.	IP do servidor Porta de conexão	–	SCARA_Connect("127.0.0.1", 8888);
SCARA_Send	Envia uma mensagem para o servidor.	Mensagem a ser enviada	–	SCARA_Send("RIGHT");
SCARA_Receive	Recebe uma mensagem do servidor.	Referência do vetor que vai armazenar a resposta e a opção de desbloquear o *socket**	–	SCARA_Receive(ref byte, true);
ExtractV3	Extrai posição ou velocidade do dado recebido retornando um vetor de 3 dimensões contendo os eixos x, y e z.	Referência do dado contendo os valores e o tipo de dado, sendo 0 para posição e 1 para velocidade	Vector 3	Vector3 vetor = ExtractV3(ref data, 0);
is_connected	Retorna true se o cliente está conectado com o servidor e false caso contrário.	-	bool	bool error = is_connected();
was_sent	Retorna true se a última chamada da função Send enviou a mensagem corretamente.	-	bool	bool error = was_sent();
was_received	Retorna true caso nenhuma mensagem foi recebida na função Receive.	-	bool	bool error = was_received();

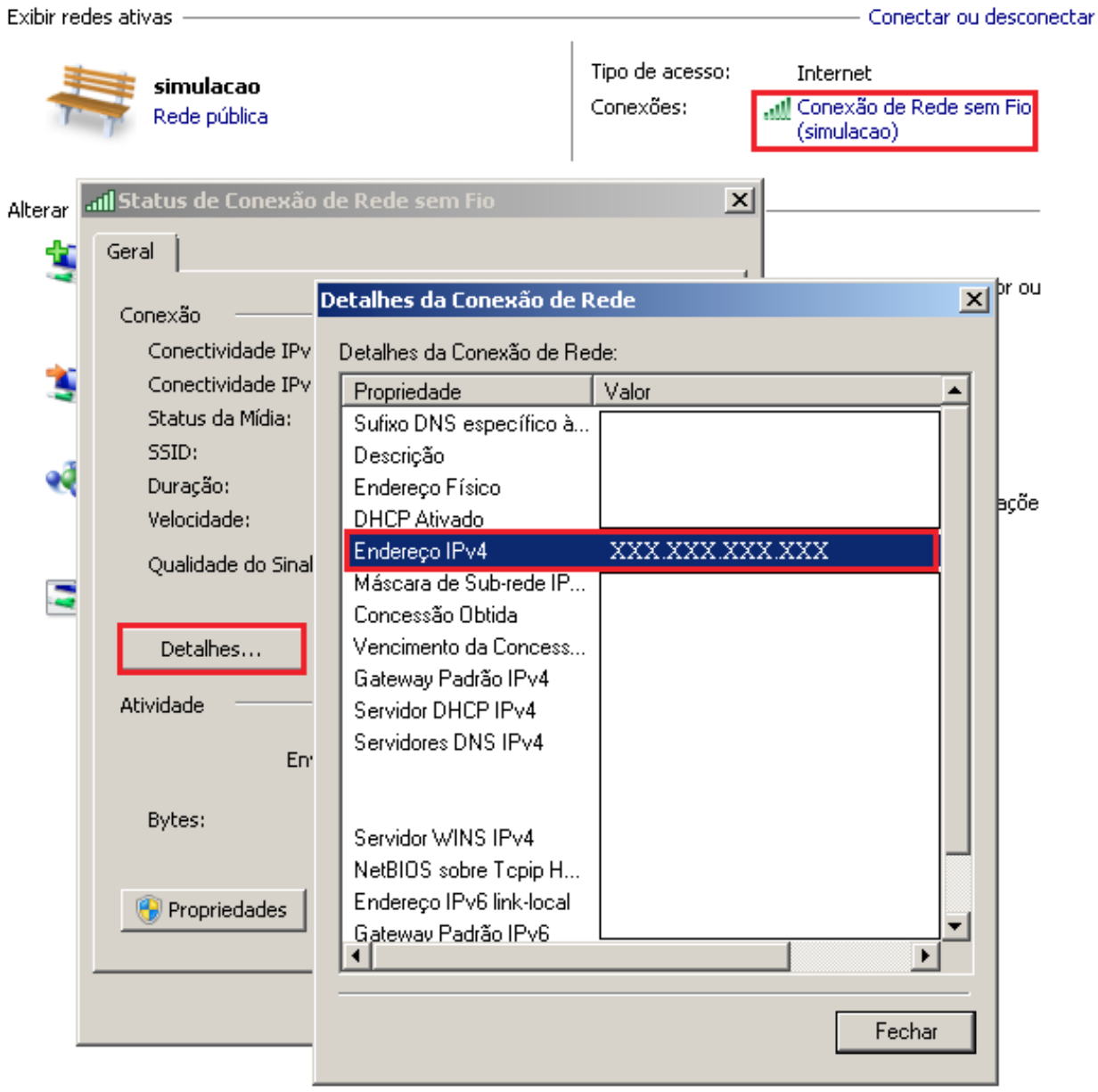
*Se o *socket* for bloqueado a função aguarda até que a mensagem seja recebida, parando o código. Se for false e o servidor não estiver enviando mensagem alguma, então *Receive* retorna e a função *was_received* retorna false.

A8- Descobrindo o IP do servidor

O cliente deve saber o IP do computador servidor para se comunicar. Se o programa de controle de impedância e o jogo estão executando no mesmo computador, o IP a ser inserido pode ser o "127.0.0.1", que representa o IP do próprio computador. Do contrário, é necessário encontrar o IP do *host* servidor.

Para encontrar o IP, use a máquina do servidor. Se for Windows 7, vá em Painel de Controle, Rede e Internet e Central de Rede e Compartilhamento. No campo Conexões, clique em sua conexão. Em seguida clique em Detalhes e anote o IP que está na linha "*Endereço Ipv4*". Observe a figura 19 caso tenha dúvidas.

Figura 19- Localização do endereço local IPv4 da máquina na rede conectada.



Para ver a comunicação funcionando, abra o projeto modelo no Unity3D. Abra o script ReRob Client. Na variável IP insira o IP encontrado. Então execute o jogo. Se a comunicação ocorreu com sucesso o jogo iniciará normalmente. Utilize o robô para movimentar a bolinha verde e tente capturar os objetos azuis.

A9- Recomendações

- O sistema de controle foi programado em tempo real, com funções que duram 1ms. Por isso, ao adicionar códigos no programa utilize processamento paralelo. A adição de códigos na função de controle de impedância pode acarretar em um movimento desordenado do braço robótico durante a execução, podendo danificar o robô e ferir as pessoas à sua volta.
- Mantenha o botão de emergência sempre em mãos durante o funcionamento do controle de impedância. Caso note algum movimento incomum do robô, pressione o botão de emergência e desligue o controle de impedância. Com o botão ainda pressionado rode o programa e execute a função *Reset* de Falhas. Então libere o botão de emergência e prossiga normalmente.
- De preferência, desligue primeiro o jogo e somente depois o controle de impedância. O jogo pode ser encerrado sem se preocupar com a função do servidor.
- Evite utilizar o controle de impedância por mais de 5 minutos.
- O controle de impedância tem sua precisão reduzida à medida que os eixos 1 e 2 se alinham. Procure usar o robô na posição menos alinhada possível.
- Uma batida da junta 1 à sua limitação mecânica em Novembro de 2016 gerou um endurecimeto desta junta. Portanto, o controle de impedância não está funcionando corretamente. Caso utilize o manipulador, foque seu trabalho na junta 2, 3 e na junta de translação. Uma tentativa de conserto é bem vinda.
- Este manual não substitui um estudo aprofundado no assunto e nem lições aprendidas em conversas com estudantes e professores.