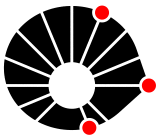

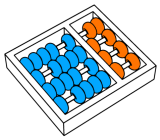

# Sistema Supervisório e de Atuação Remota de Robôs Humanoides

*Bruno Takeshi Hori RA: 145539*
*André Tsuyoshi Sakiyama RA: 150547*
*Orientadora: Profa. Dra. Esther Luna Colombini*

Relatório Técnico - IC-PFG-18-03
Projeto Final de Graduação
2018 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

# Sistema Supervisório e de Atuação Remota de Robôs Humanoides

André Tsuyoshi Sakiyama Bruno Takeshi Hori

## 1 Introdução

Este trabalho visa desenvolver um sistema supervisório e de atuação remota de Robôs Humanoides que permita a um leigo operar um robô bípede real de 25 graus de liberdade (DOF), monitorando, em tempo real, seus diversos sensores.

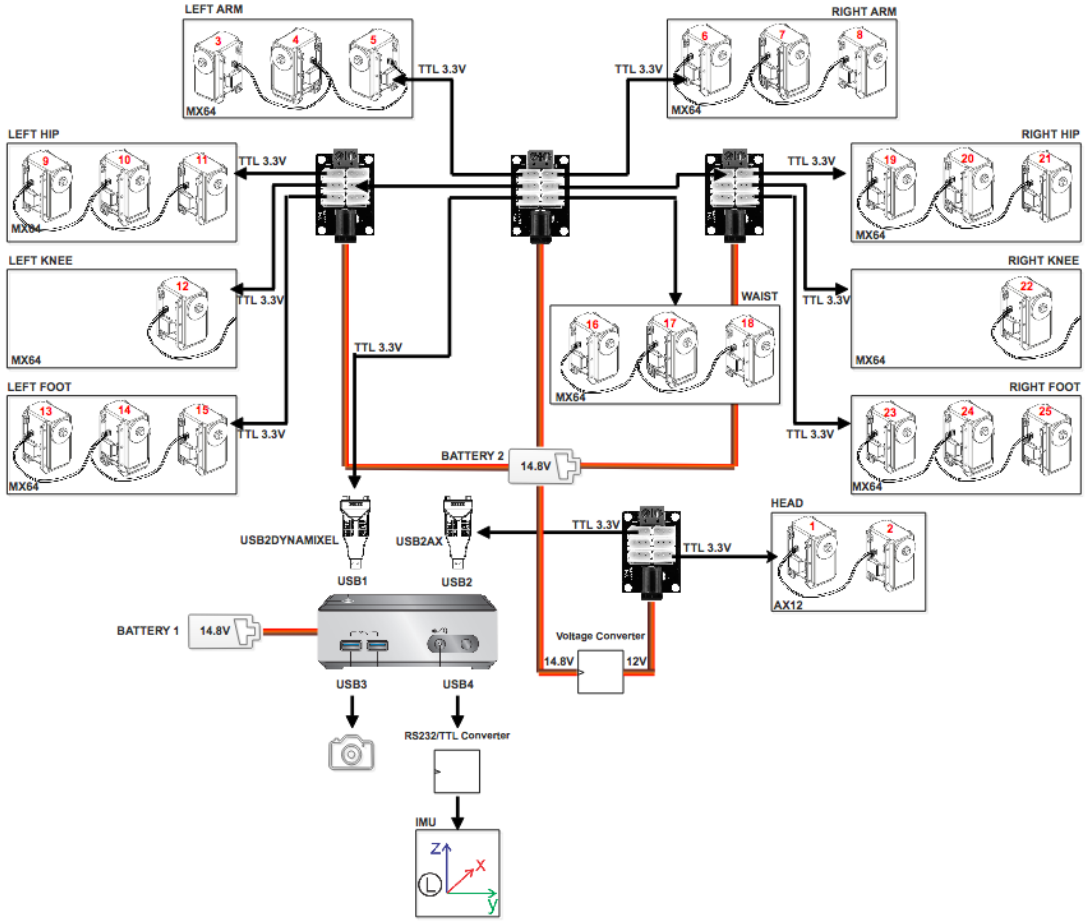


**Figura 1:** Imagem representativa do Robô

É imprescindível que, em todo o momento, o sistema reflita a atual situação do robô no mundo. A comunicação entre o sistema e o robô deverá ser realizada de forma que o módulo remoto interfira minimamente na operação do robô.

## 2 Justificativa

O intuito do projeto foi criar um sistema de supervisão para que pessoas sem conhecimento técnico do robô possam verificar cada propriedade dos motores a partir das informações da interface.

O trabalho foi separado em 2 partes: comunicação com os motores e interface gráfica. A parte de comunicação foi responsabilidade de André Tsuyoshi enquanto a parte de interface ficou sobre responsabilidade de Bruno Takeshi.

## 3 Objetivos

Neste projeto, objetiva-se:
- Definir a lista de comandos que podem ser executados pelo robô;
- Definir a lista de sensores que podem ser executados pelo robô;
- Definir as interfaces de comunicação;
- Projetar a interface gráfica mais adequada ao projeto;
- Definir o protocolo lógico de comunicação dos comandos;
- Implementar o sistema de controle remoto do robô, preferencialmente cross-platform;
- Validar o sistema através da comunicação com o robô real.

## 4 Desenvolvimento do Trabalho

O trabalho foi dividido em três grandes fases: Planejamento, Desenvolvimento e Teste.

A primeira fase se constituiu em discutir as funcionalidades finais e gerar o calendário com cronograma do planejamento e separação de tarefas do grupo. Além disso, nesta mesma fase, foi feito o levantamento dos requisitos funcionais e não funcionais do sistema e o desenvolvimento da interface gráfica, com todas a interações possíveis que o usuário poderia executar.

A fase de Desenvolvimento se resume em preparação do ambiente e execução do planejamento feito na fase passada. Para a construção dos componentes gráficos, foi utilizada a biblioteca WxWidget. A escolha desta biblioteca se deve ao fato que a implementação é feita em C++, portanto o seu desempenho se destaca em relação a bibliotecas baseadas em linguagens de mais alto nível. Junto com o WxWidget, foi utilizada a biblioteca pthread com o intuito de possibilitar a execução simultanea de diferentes funcionalidades. A comunicação com os componentes motor e sensor inercial foi feita via USB serial com o auxílio da Dynamixel SDK.

Por fim, na fase de Teste foi feita a validação do produto final testando todas as funcionalidades discutidas na fase número um. Os teste foram feito utilizando dois motores Dynamixel MX-64 e um sensor inercial UM7-LT

## 4.1 Requisitos não-funcionais

- Foi decidido que o projeto seria feito em C++ para melhorar a performance do sistema.
- A interface será feita utilizando a biblioteca wxWidgets.
- Os comandos mandados para o robô serão feitos com o auxílio da SDK do Dynamixel.
- Usuário pode conferir e modificar as diversas variáveis de um motor que for selecionado.
- Para realizar as funções em multi-thread foi utilizado a biblioteca de pthreads.

## 4.2 Requisitos funcionais

- Usuário poderá conferir e modificar a posição assim como o tempo que levará a transição dos motores que foram selecionados.
- Usuário poderá salvar as informações recuperadas dos motores conectados.
- Usuário poderá carregar a partir de um arquivo os movimentos dos motores conectados do robô.

## 4.3 Interface gráfica

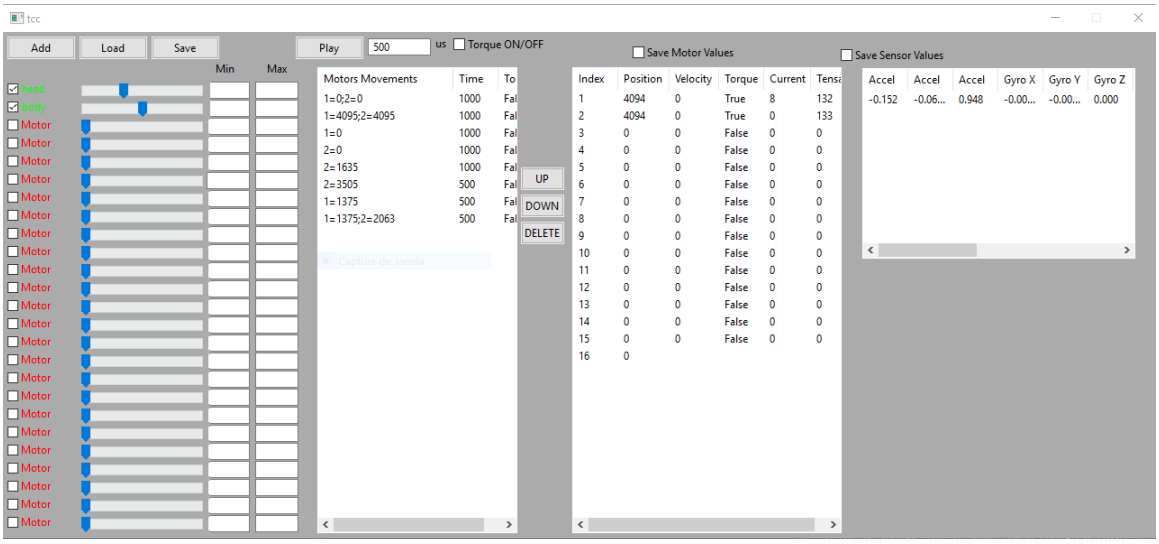


**Figura 2:** Imagem da interface gráfica

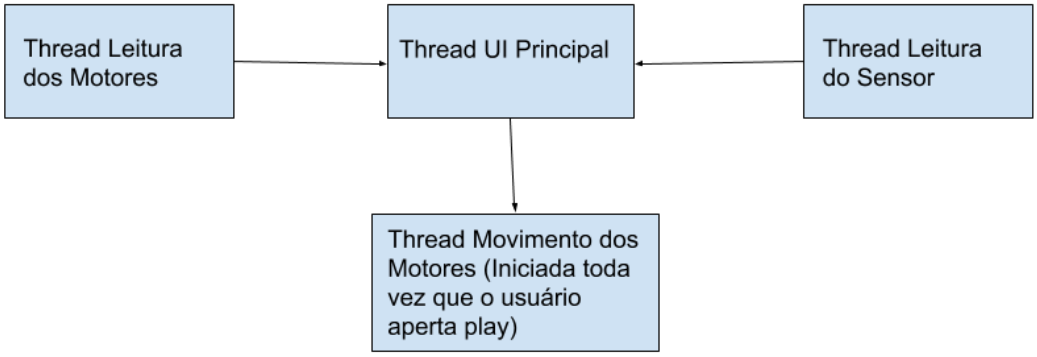


**Figura 3:** Representação das threads

Primeiramente, a interface gráfica foi feita utilizando as bibliotecas do Qt Creator, porém houve muitas dificuldades para integrar outras bibliotecas como por exemplo pthreads.

Consequentemente, tentamos fazer a interface utilizando Java com integração com C++ por bibliotecas nativas, por causa da comunicação com os motores. No entanto, houve novamente problemas com a integração.

Finalmente, foi decidido a utilização da biblioteca wxWidgets, a qual conseguimos integrar praticamente tudo, somente a biblioteca do OpenCV não funcionou de forma esperada.

## 4.4 Modo de Uso

Antes de inicializar a aplicação é necessário atualizar o arquivo de configuração: motor.cfg encontrado na mesma pasta que o arquivo executável.

O arquivo de configuração necessita estar no seguinte formato:
sensorPort=NOME_DA_PORTA_USB_DO_SENSOR
numberOfPorts=N , sendo N = número de portas para os motores
portI=NOME_DA_PORTA_USB_DOS_MOTORES , para cada I portas
motorI=INDICE_CONFIGURADO_DO_MOTOR_I
bodyI=LABEL_DO_MOTOR_I_NA_TABELA , para cada I motores
Exemplo:
sensorPort=COM6
numberOfPorts=1
port1=COM5
motor1=1
body1=head
motor2=2
body2=body
O usuário poderá executar diversos comandos com esta aplicação:

1. Adicionar movimento a cena

2. Ler de um arquivo uma cena
3. Salvar em um arquivo uma cena
4. Executar uma cena
5. Deletar movimento da cena
6. Locomover movimento na cena
7. Salvar informações da tabela de valores dos motores
8. Salvar informações da tabela de valores do sensor

### 4.4.1 Adicionar Movimento

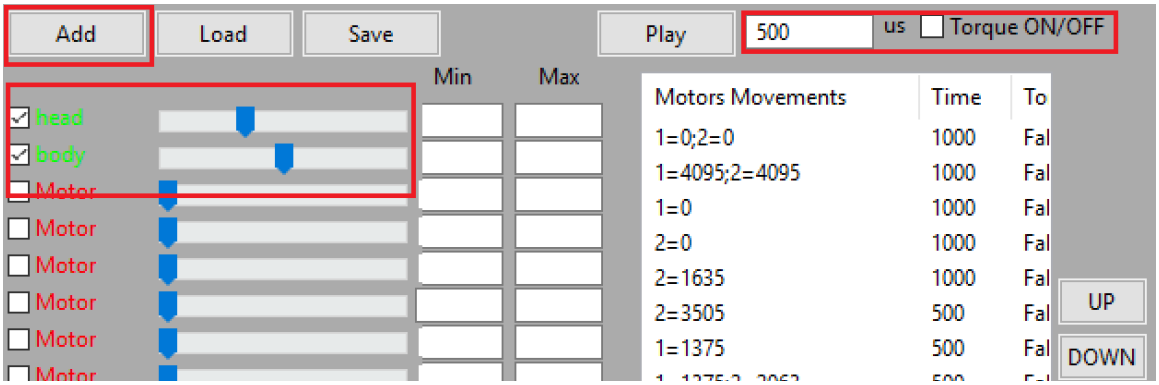


**Figura 4:** Adicionar movimento

O programa verifica quais motores estão ativos (Motor cuja label do CheckBox se encontra com a cor verde) e selecionados e verifica a posição dos sliders e adiciona um movimento a tabela de movimentos da cena, definindo o torque no final do movimento e em quantos segundos durará o movimento.

### 4.4.2 Ler ou salvar cena

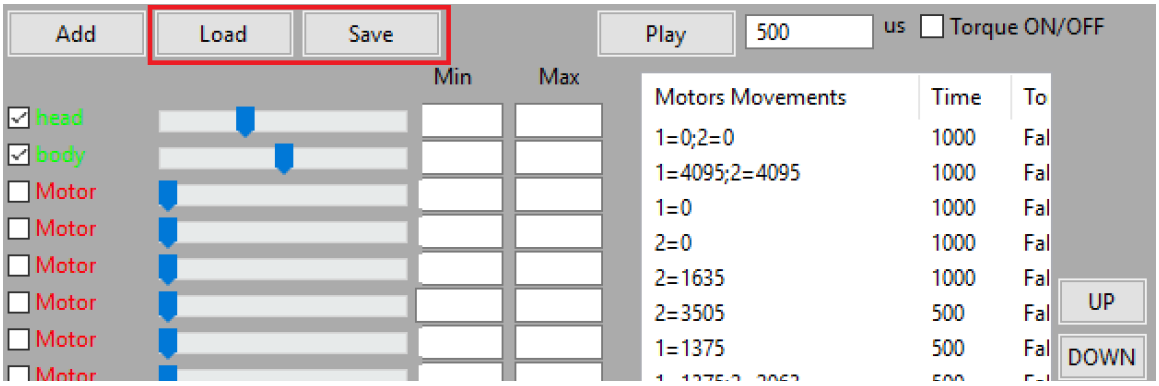

**Figura 5:** Botões de SAVE e LOAD

Ao clicar no botão "Load", uma tela para selecionar um arquivo do tipo .txt se abrirá. Após aberto, a aplicação irá adicionar cada movimento no arquivo à tabela de movimentos. Esse arquivo deverá estar escrito no seguinte formato para poder ser lido pela aplicação:

(index do motor)=(posição que se deseja) seguido de ; para cada motor seguido de : time=(tempo do movimento) seguido de : torque=(true ou false, torque que se deseja) seguido de | caso tenha um próximo movimento.

Exemplo:

1=3;2=10:time=10:torque=true|7=111;8=40:time=35:torque=false

Ao clicar no botão "Save", uma tela para selecionar um arquivo do tipo .txt se abrirá. A aplicação salvará a cena no seguinte formato:

(index do motor)=(posição que se deseja) seguido de ; para cada motor seguido de : time=(tempo do movimento) seguido de : torque=(true ou false, torque que se deseja) seguido de | caso tenha um próximo movimento para cada movimento na tabela de movimentos.

Exemplo:

1=3;2=10:time=10:torque=true|7=111;8=40:time=35:torque=false

### 4.4.3 Executar cena

Ao clicar no botão "Play", a aplicação gerá uma nova thread para execução dos movimentos.

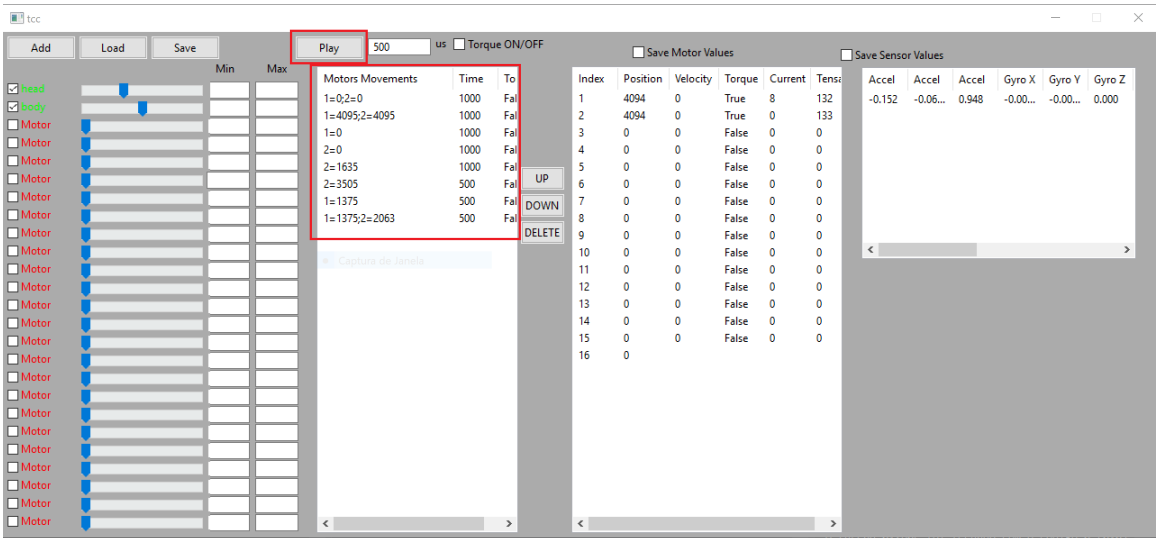


**Figura 6:** Executar cena

### 4.4.4 Locomover movimento na cena

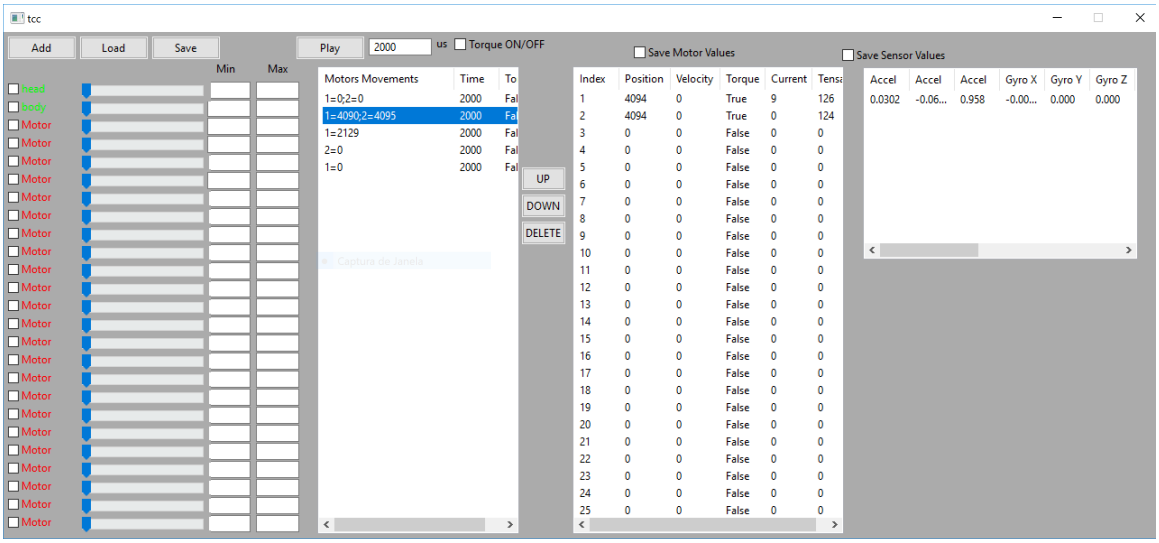


**Figura 7:** Movimento Selecionado

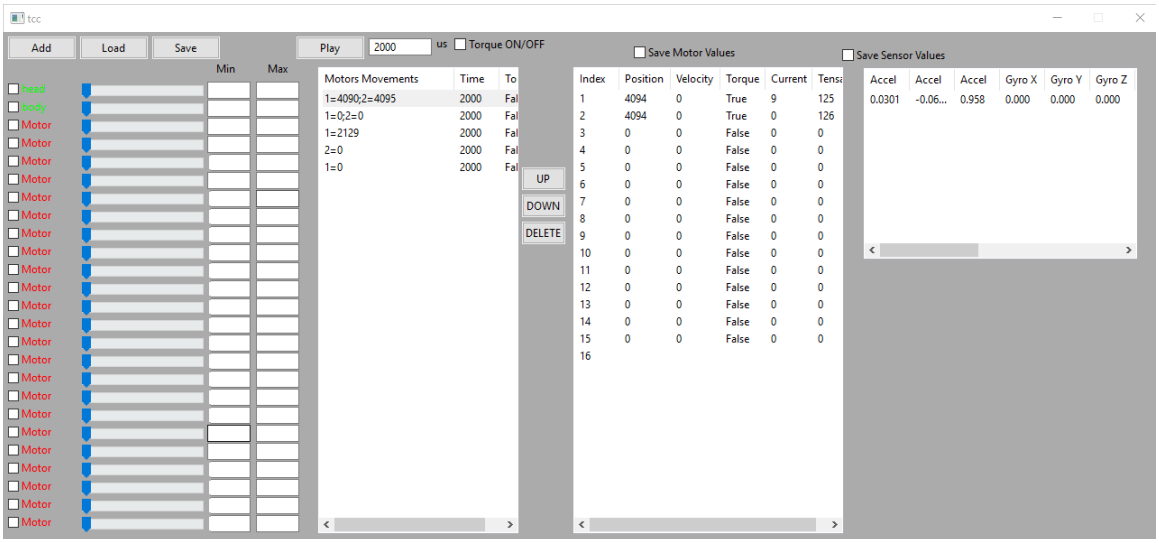


**Figura 8:** Locomovendo o movimento para cima

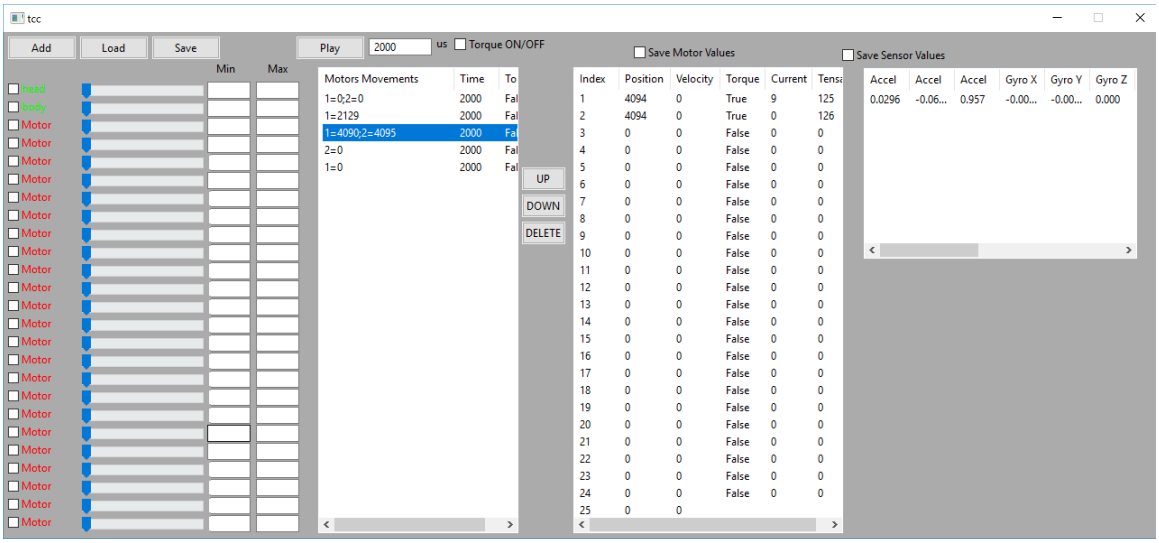


**Figura 9:** Locomovendo o movimento para baixo

O usuário pode selecionar um movimento da tabela e clicar nos botões "Up"ou "Down"para locomover o movimento selecionado na cena.

### 4.4.5 Deletar movimento

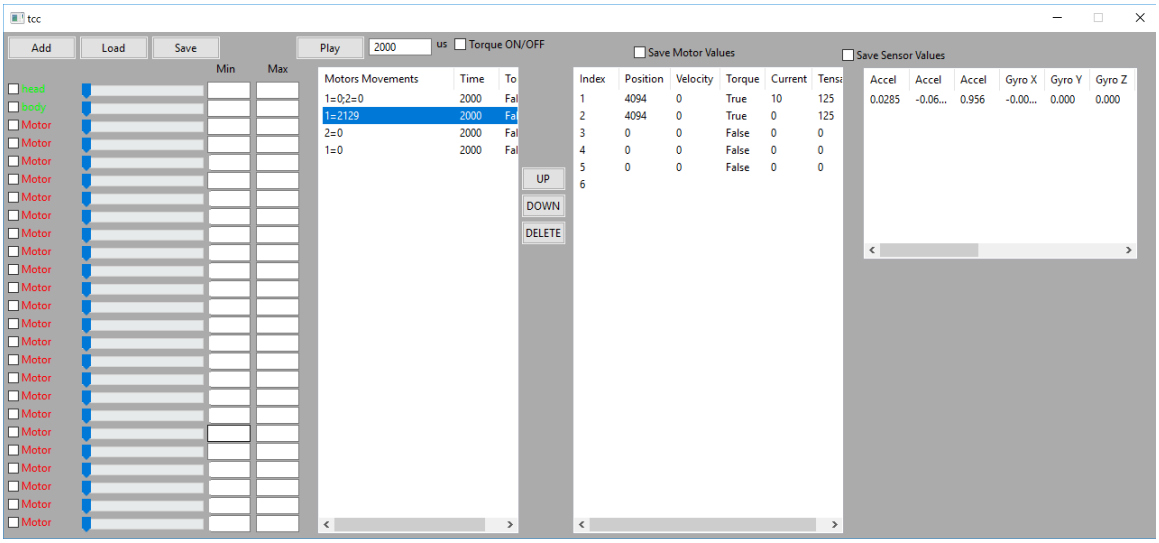


**Figura 10:** Movimento Deletado

O usuário pode selecionar um movimento da tabela e clicar no botão "Delete"para deletar da cena o movimento selecionado.

### 4.4.6 Salvar tabela de valores dos motores

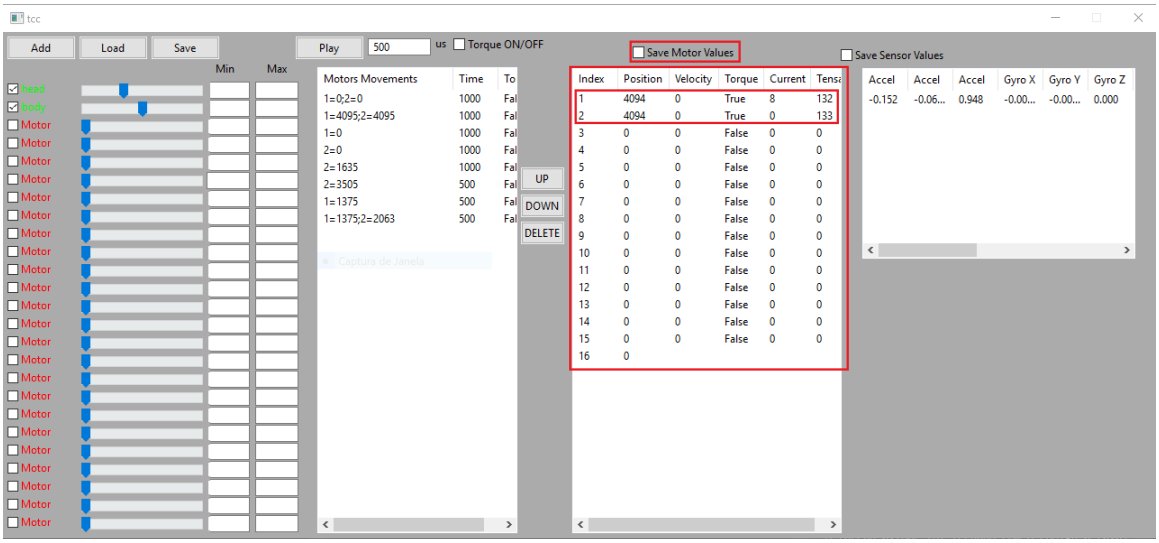


**Figura 11:** CheckBox para salvar tabela de valores dos motores

Ao deixar selecionado o CheckBox da tabela de valores dos motores, a aplicação irá salvar os valores dos motores ativos em um arquivo cujo nome é "valueOutput.txt"enquanto o CheckBox estiver selecionado.

### 4.4.7 Salvar tabela de valores do sensor

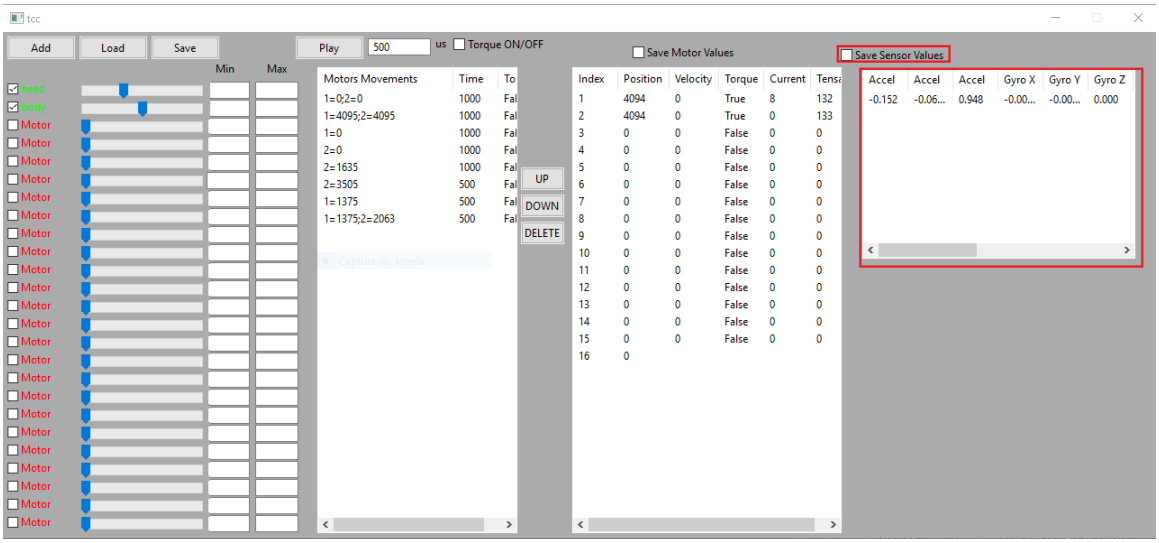


**Figura 12:** CheckBox para salvar tabela de valores do sensor

Ao deixar selecionado o CheckBox da tabela de valores do sensor, a aplicação irá salvar os valores do sensor em um arquivo cujo nome é "sensorOutput.txt"enquanto o CheckBox estiver selecionado.

### 4.5 Motores e sensores

O modelo dos motores utilizados é o Dynamixel MX-64. A sua comunicação foi feita com o auxílio da biblioteca Dynamixel SDK e se baseia em abrir uma porta serial onde estão conectados um ou mais motores com IDs diferentes, definidos em momentos anteriores. Após a abertura da porta é possível escrever e ler dados em espaço do memória específicos de cada motor, de acordo com o protocolo utilizado. O protocolo estabelecido foi o 2.0 e este define qual funcionalidade representa um determindado espaço do memória. A leitura e escrita do dados do motor pode ser feita independente em cada motor, respeitando os requisitos funcionais e possibilitando a leitura em tempo real junto com o motor em movimento.

O modelo do sensor utilizado é o UM7-LT juntamento com o adaptador USB to TTL. O adaptador necessita de um driver para seja estabelecida a comunicação. Da mesma forma que motor, é necessário que seja aberta uma porta serial para leitura dos dados. Porém, neste caso, o código de leitura do sensor foi fornecido e utilizado como API.

## 5 Teste e validação

Após o término da fase de desenvolvimento, foi iniciada a fase de teste. Esta fase basicamente averigua as funcionalidades inicialmente requisitadas e coloca em prática o fluxo de execução de um usuário comum.

Os testes foram feitos com dois motores Dynamixel MX-64 e um sensor inercial UM7-LT.

Inicialmente os motores e o sensor foram conectados, respectivamente, nas portas seriais COM5 e COM6 e logo em seguida foi iniciado o programa. A interface gráfica do projeto foi aberta, permitindo a visualização dos motores conectados e já mostrando os dados em tempo real do motores e do sensor inercial com uma taxa de atualização de 10Hz.

Em seguida foram adicionados alguns comandos de movimento na fila para os motores conectados e o botão Play foi pressionado. Os dois motores começaram a se mover de acordo com os comandos enfileirados e os dados de leituras continuaram a atualizar.

Por fim para testar o armazenamento de arquivos, o botão Save foi pressionado para guardar os comandos enfileirados em um arquivo e as caixas Save Motor Values e Save Sensor Values foram selecionadas por um espaço de tempo para guardar os valores dos motores e do sensor por este tempo. Os dados anteriores foram guardados no arquivos commands.txt, valueOutput.txt e sensorOutput.txt.

Um dos testes teve como saída :

valueOutput.txt:

time61700Motor 1: position=3247: velocity=242: torque=true: current=42
time61700Motor 0: position=4093: velocity=0: torque=true: current=11
time61900Motor 1: position=4095: velocity=0: torque=true: current=0
time62000Motor 1: position=4095: velocity=0: torque=true: current=-2
time62000Motor 0: position=3605: velocity=-210: torque=true: current=-24
time62200Motor 1: position=3666: velocity=-196: torque=true: current=-59
time62200Motor 0: position=2639: velocity=-219: torque=true: current=-38

time62300Motor 1: position=2946: velocity=-224: torque=true: current=-37
time62300Motor 0: position=1937: velocity=-229: torque=true: current=-33

sensorOutPut.txt:

time 1053400 Sensor AccelX: 0.013393 AccelY: -0.085611 AccelZ: 0.956323 GyroX: 0.000099 GyroY: -0.000741 GyroZ: 0.009812 MagX: -0.049988 MagY: -0.085611 MagZ: 0.851992 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.088971 EulerY: -0.014189 EulerZ: -1.609333
time 1053500 Sensor AccelX: 0.013393 AccelY: -0.085611 AccelZ: 0.956323 GyroX: 0.000099 GyroY: -0.000741 GyroZ: 0.009812 MagX: -0.049988 MagY: -0.085611 MagZ: 0.851992 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.088971 EulerY: -0.014189 EulerZ: -1.609333
time 1053600 Sensor AccelX: 0.013393 AccelY: -0.085611 AccelZ: 0.956323 GyroX: 0.000099 GyroY: -0.000741 GyroZ: 0.009812 MagX: -0.049988 MagY: -0.085611 MagZ: 0.851992 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.088971 EulerY: -0.014189 EulerZ: -1.609333
time 1053700 Sensor AccelX: -0.004604 AccelY: -0.050065 AccelZ: 0.974630 GyroX: -0.136961 GyroY: -0.032016 GyroZ: 0.054498 MagX: -0.046442 MagY: -0.050065 MagZ: 0.851997 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.107187 EulerY: -0.018216 EulerZ: -1.599363
time 1053800 Sensor AccelX: -0.004604 AccelY: -0.050065 AccelZ: 0.974630 GyroX: -0.136961 GyroY: -0.032016 GyroZ: 0.054498 MagX: -0.046442 MagY: -0.050065 MagZ: 0.851997 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.107187 EulerY: -0.018216 EulerZ: -1.599363
time 1053900 Sensor AccelX: -0.004604 AccelY: -0.050065 AccelZ: 0.974630 GyroX: -0.136961 GyroY: -0.032016 GyroZ: 0.054498 MagX: -0.046442 MagY: -0.050065 MagZ: 0.851997 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.107187 EulerY: -0.018216 EulerZ: -1.599363
time 1054000 Sensor AccelX: -0.004604 AccelY: -0.050065 AccelZ: 0.974630 GyroX: -0.136961 GyroY: -0.032016 GyroZ: 0.054498 MagX: -0.046442 MagY: -0.050065 MagZ: 0.851997 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.107187 EulerY: -0.018216 EulerZ: -1.599363
time 1054100 Sensor AccelX: 0.031585 AccelY: -0.125313 AccelZ: 0.949779 GyroX: -0.049821 GyroY: -0.023105 GyroZ: 0.037196 MagX: -0.040240 MagY: -0.125313 MagZ: 0.842496 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.146303 EulerY: -0.028570 EulerZ: -1.569067
time 1054200 Sensor AccelX: 0.031585 AccelY: -0.125313 AccelZ: 0.949779 GyroX: -0.049821 GyroY: -0.023105 GyroZ: 0.037196 MagX: -0.040240 MagY: -0.125313 MagZ: 0.842496 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.146303 EulerY: -0.028570 EulerZ: -1.569067
time 1054300 Sensor AccelX: 0.031585 AccelY: -0.125313 AccelZ: 0.949779 GyroX: -0.049821 GyroY: -0.023105 GyroZ: 0.037196 MagX: -0.040240 MagY: -0.125313 MagZ: 0.842496 QuatX: -0.038000 QuatY: 0.027896 QuatZ: -0.793511 EulerX: -0.146303 EulerY:

-0.028570 EulerZ: -1.569067

## 6 Conclusão

Nesse projeto conseguimos planejar e desenvolver um sistema funcional de controle e monitoramento de componentes básicos que constituem um robô humanóide.

A nossa aplicação possui uma interface intuitiva a qual um usuário sem conhecimento técnico conseguiria a partir das informações retiradas inferir possíveis problemas no comportamento dos componentes do robô.

## Referências

[1] Página oficial da biblioteca wxWidgets

https://www.wxwidgets.org/

[2] Manual oficial do ROBOTICS

http://emanual.robotis.com/docs/en/dxl/mx/mx-64/

[3] Dynamixel SDK

http://www.robotis.us/dynamixelsdk/

[4] Sensor Inercial UM7-LT

https://www.pololu.com/product/2740/resources

[5] Motor Dynamixel MX-64

http://support.robotis.com/en/product/actuator/dynamixel/mx_series/mx-64(2.0).htm

[6] Repositório do Projeto

https://github.com/Dekkoh/TCC-MC030