Rehabilitation Robot Software Documentation

Nicholas Berezny

February 27, 2019

1 Real Time Linux

1.1 Background

Real time systems are those requiring that computations be made by fixed deadlines - in other words, systems which must be deterministic. Missing deadlines may result in damage to the system or to its environment. This category can be further subdivided into soft real-time and hard real-time. Hard real-time systems usually mathematical verify that deadlines will not be missed. For example, QNX and VxWork are both hard RTOS's. Soft real-time relaxes this condition, but still contains many of the features of a real-time system. Realtime Linux, for example.

Typical real time procedures include memory-locking, multi-threading, setting priorities and schedulers, and testing latencies. Memory-locking ensures that no pagefaults occur during execution, which can cause significant delays. Multi-threading is a form of parallel computing, allowing for multiple threads of computation to be run simultaneously. a scheduler will determine how many resources to delegate to each thread, based on user-set priority levels. The type of scheduler can also be changed (First-in-first-out FIFO, or Roundrobin RR).

This software runs on a real-time enabled version of Linux which uses the PREEMPT_RT patch to add the functionalities mentioned above. This is a soft RTOS since it is not fully deterministic, which should be adequate for this project as missed time-steps should not cause serious harm or damage.

1.2 Installation

The following is an outline of the installation process for a PREEMPT_RT patched linux kernel with Ubuntu. Other linux flavours may also be used. More detailed instruction can be found at the Linux Foundation Website.

1. Download the linux kernel and the patch. The latest stable release of the patch is 4.14 (as of 11/10/2018)
2. Patch the kernel through the command line
3. Configure the kernel. Make sure to select "Fully Preemptible Kernel"
4. Install the kernel on a machine running Ubuntu ....

1.3 Libraries

2 Controller

2.1 Outline

2.2 Initialization

Initialization can be found in the first few hundred lines on imp_main.c, which calls functions found in imp_init.c.

1. TCP Socket Initialization: E.g. this tutorial.
2. Connecting to the DAQ: Instructions found here.
3. Creating a data log text file
4. Initializing Mutexes:
5. Setting Thread Parameters and Locking Memory : Example here.

2.3 Getting Control Parameters

Parameters like controller gains, desired maximum velcoity, etc can be set either using variables defined in imp_variables.h, or by connecting to the UI and receiving custom parameters. Setting the macro CONNECT_TO_UI = 1 will allow the robot to connect to the UI server, and then setting GET_PARAMS_FROM_UI = 1 will set control parameters based on a message from the UI.

If getting parameters from the UI, the process will wait for a message from the UI containing the parameters. This message will being with a capital 'S'. After the system receives the message, it uses regular expression to extract parameter values. It then waits again for start message from the UI. It will then break the loop and continue executing.

The message encodes parameter values by starting with a letter representing the parameter (e.g. the proportion gain is 'P'), followed by the value for the parameter (potentially containing a decimal point). Each parameter is separated by an underscore. For example, if the user sets the P gain to 5.1, the message will read '_P5.1_'.

2.4 Discretization

Control parameters can be used to construct the admittance control continuous system matrices A and B:

$$\dot{X} = AX + Bf \tag{1}$$

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{K}{m} & -\frac{B}{m} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}$$

The equivalent discrete system can be derived from the continuous matrices:
The equivalent discrete system is:

$$X_{k+1} = A_d X_k + B_d f \tag{2}$$

$$A_d = e^{At_s} \tag{3}$$

$$B_d = A^{-1}(A_d - I)B \tag{4}$$

Where t_s is the sampling time (time in seconds of a single iteration). The matrix exponential can be approximated by calculating the first m terms of the series:

$$A_d = \sum_{n=0}^{m} \frac{A^n t_s}{n!} = I + \frac{At_s}{1} + \frac{A^2 t_s}{2} ... \tag{5}$$

Calculation of the discrete system matrices is handled in the file `RehabRobot/controller/imp_math.c`, which contains the following functions relevant to discretization:

- **matrix_square**: squares a given matrix and stores in another location
- **factorial**: calculations the factorial of a given number
- **matrix_exp**: calculates the exponential of a matrix (e^A) using the series approximation in Eqn. 5.
- **invert_matrix**: inverts a given 2x2 matrix (A^{-1})
- **calc_Bd**: calculates the discrete matrix Bd using Eqn. 4

2.5 Zeroing the Force Sensor & Homing

The motor encoder is not absolute, and so it the controller needs to determine the actuators position before continuing. This is done by homing the device, whereby it is slowly brought forward until triggering the front limit switch. This is considered position 0. At this point, the encoder is zeroed.

The homing process is comprised of a while loop, which is continually checks for contact from the front limit switch. A slow forward motor command is set constantly until the limit is triggered, which stops the motor and breaks the loop.

Next, the force sensor is zeroed. A series of readings are taken. It is important that no force be applied to the sensor during this process. The average of these readings is saved as the variable FT_OFFSET, which is used to calibrate all raw sensor readings throughout the robots operation.

2.6 Thread 1: Controller

2.6.1 Reading & Writing to the DAQ

2.6.2 Filtering

2.6.3 Trajectory Calculation

2.6.4 Admittance Control

2.6.5 Maintaining Frequency

2.7 Thread 2: Server

2.7.1 TCP Socket

2.8 Thread 3: Data Logging

2.8.1 Data Formatting and Printing

2.8.2 Plotting with Python

3 UI Server

3.1 Outline

The purpose of the Node.js server is twofold: to serve the web application to the UI device either locally or over a local network, and to handle communication between the real time controller and the UI. The server consists of a single file, which is located at server/server.js. The following libraries/tools are used:

- **Node.js** as the base JS runtime environment (site)
- **Express** for the server framework (site)
- **Next** for server-side rendering (site)
- **Socket.io** for websocket communication (site)

3.2 HTTP Server

The UI application is served using Node.js, using both Next.js (server-side rendering) and Express.js. First, Next renders the app into a static website, using Webpack among other tools. Express then servers this over a local network through the connected router. The relevant code can be found at the bottom of the server file:

```
nextApp.prepare().then(() => {
   app.get('*', (req, res) => {
    return nextHandler(req, res)
  })
  server.listen(ui_port, (err) => {
    if (err) throw err
    console.log('> Ready on http://localhost:3000')
  })
})
```

3.3 Communication

Communication is routed through the server from the controller to the UI, and vice versa. Communication with the controller is handled by a TCP Socket, whereas communication to the UI is handled by a websocket created with Socket.io.

4 UI Application

4.1 Background

A variety of user interface technologies exist. An emerging trend within UI development is the use of web tools, which take advantage of already established programs like Chrome or other browsers. Frameworks like Angular, React, and Ionic can be used to build reusable UI components using familiar web utilities like HTML, Javascript, and css.

This software using React for its UI (site). The UI is essentially a website which can be served and loaded in a browser like any other local site. The upside of this method is that users do not need to install anything on their device, only requiring that they know the IP address. In the future, it may be beneficial to build a native application that can be installed on a Windows machine or on a tablet. Fortunately, it is fairly easy to convert a React website to a native app, using either Electron (site) for the windows app or React Native(site) for the android/iPad app.

In addition to React, the UI uses the following tools or frameworks:

- **Node.js** as the base JS runtime environment (site)
- **Redux** for state management (site)
- **Next** for server-side rendering (site)
- **Material-UI**, a UI component library based on Google's Material Design philosophy (site)
- **Three.js** as the 3D graphics library (site)

4.2 Installation

First, Node.js must be installed on your system, along with the package manager npm. All other tools are then installed using npm, which can be done by either running the following command in the server folder, or by running ... in the git repository.

```
sudo npm install react react-dom redux react-redux next
@material-ui/core @material-ui/icons react-websocket three express
```

4.3 React

4.4 Three.js & Visuals

Three.js is a javascript library for the creation of 3D graphics for the web. It is used in the UI to create visuals and games related to the therapeutic motions to increase user engagement.

4.4.1 Scene Setup in React

Setting up the scene within React components differs from the standard Three.js tutorials. Firstly, the scene is created in the componentDidMount() function. Any object or which does not remain static in the scene must be stored in the component's state. All scenes must contain some fundamental objects:

- Scene
- Camera
- Renderer

First, create the scene:

```
var scene = new THREE.Scene()
```

Next, create the renderer and the camera. The following code is an example – more options are available and can be found online in the Three.js documentation.

```
var camera = new THREE.PerspectiveCamera(90, window.innerWidth/window.innerHeight, 0.1, 800 );
scene.add( camera );

const renderer = new THREE.WebGLRenderer({ antialias: true })
renderer.setSize(width, height)
```

Some objects have positions and orientations which can be changed, for example the position of the camera can be set as follows:

```
camera.position.set( -95,-50,30);
camera.rotation.set(1.5,0.0,0.0);
```

Next, you create the 3D objects in your scene. Three.js provides certain geometries, like spheres, discs, planes, and blocks. For more complex objects, you will likely need to import a 3D model made from an external program (see, for example, Blender). These objects also have materials, which can be single colours or more complex textures. For example, the following code creates a blue cube with a side length of 100 at the origin.

```
var material = new THREE.MeshBasicMaterial( { color: 0x0036FF} );
var geometry = new THREE.BoxGeometry( 100, 100, 100 );
var cube = new THREE.Mesh( geometry, material );
cube.position.set(0, 0, 0);
scene.add(cube);
```

The final step is to add all non-static objects to the state of the component. This includes the scene, renderer, the camera, and any objects that will move or interact during the game.

```
this.scene = scene
this.camera = camera
this.renderer = renderer
this.cube = cube
```

4.4.2 Games

5 Controller Folder Organization

6 Server Folder Organization

7 Suggestions for Future Improvements

- Security
- Handling multiple connections to the http server
- Improve games