Rehabilitation Robot Software Documentation

Nicholas Berezny

December 13, 2018

1 Real Time Linux

1.1 Background

Real time systems are systems requiring that computations be made by strict time deadlines - in other words, they 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 is a hard real time OS. 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, multithreading, 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 that can speed up your process. Threads must have a priority set based on how critical they are to the functioning of the system. Thread execution is handled by the kernel?, which delegates processing time according to the set scheduler (first-in first-out, round robin, etc.)

This software runs on a real-time enabled version of Linux which uses the PREEMPT_RT patch.

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_'.

After getting the parameters, the controller needs to calculate the discrete system matrices for the admittance controller, which is based on the set stiffness (K), damping (B), and mass (m). The continuous time matrices are:

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

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

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

The equivalent discrete system is:

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

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

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

The matrix exponential can be approximated by calculating the first n terms of the series:

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