A Software Library for Design and Visualization of Adaptive Controllers

by

John Rebula

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Mechanical Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2006

© John Rebula, MMVI. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author

Department of Mechanical Engineering

May 12, 2006

Certified by

John Leonard
Associate Professor
Thesis Supervisor

Accepted by

John H. Lienhard V
Undergraduate Officer, Department of Mechanical Engineering
A Software Library for Design and Visualization of Adaptive Controllers

by

John Rebula

Submitted to the Department of Mechanical Engineering on May 12, 2006, in partial fulfillment of the requirements for the degree of Bachelor of Science in Mechanical Engineering

Abstract

An adaptive control library was written in Java that allowed a user to programatically specify the dynamics of a system and performed adaptive control. Also, visualization software was written to help monitor and modify the controller in real time. The software was tested on several simulated mechanical systems and on a DC motor.

Thesis Supervisor: John Leonard
Title: Associate Professor
Acknowledgments

I would like to thank Prof. Slotine of the MIT Mechanical Engineering Department, who taught me adaptive control theory (twice, in fact). The clarity with which he explained the technique led me to undertake this project. I would also like to thank my thesis advisor, Prof. Leonard.
Contents

1 Introduction .................................................. 11

2 Relevant Control Theory ................................... 13
   2.1 Adaptive Control Library ................................ 14

3 Description of Software .................................... 17
   3.1 Adaptive Control Library ................................ 17
   3.2 Visualization Software .................................. 18

4 Simulation and Experiment ................................. 21
   4.1 DC Motor Example ........................................ 22
      4.1.1 Simulation Results ................................... 22
      4.1.2 Experimental Results ................................ 26
   4.2 Two Link Simulation ...................................... 30

5 Conclusions .................................................. 37

A Example Library Extension: Noise Resistant Controller 39

B Storage File Format ......................................... 41
List of Figures

3-1 Object Inheritance diagram, showing the class hierarchy of the types used in the controller. Large errors denote inheritance while small arrows imply usage. .............................................................. 17

3-2 The main screen of the visualizer Graphical User Interface with components for setting the trajectories of the trial ..................................................... 19

3-3 The adaptation control panel, allowing the user to modify and observe that adaptation in real time ................................................................. 20

4-1 Picture of the experimental assembly, a motor with attached wheel controlled by an ORC board. ................................................................. 21

4-2 Simulated and desired position trajectories for a DC motor ........ 23

4-3 Simulated and desired velocity trajectories for a DC motor ........ 23

4-4 The first adapted parameter and actual model value over time ...... 24

4-5 The second adapted parameter and actual model value over time ... 24

4-6 The generalized error $s$ of the simulated DC motor ................. 25

4-7 Simulated, experimental, and desired position trajectories for a sinusoidal input ................................................................. 26

4-8 Simulated, experimental, and desired velocity trajectories for a sinusoidal input ................................................................. 27

4-9 Simulated and experimental control voltages for a sinusoidal input for a DC motor ................................................................. 27

4-10 Simulated, experimental, and modeled $a_1$ values for a sinusoidal input 28

4-11 Simulated, experimental, and modeled $a_2$ values for a sinusoidal input 28
4-12 Simulated and experimental generalized error $s$ values for a sinusoidal input for a DC motor .................................................. 29
4-13 Simulated and desired position trajectories for a sinusoidal input . . . 32
4-14 Simulated and desired velocity trajectories for a sinusoidal input . . . 32
4-15 Simulated torques for a sinusoidal input to a two link manipulator . . 33
4-16 Simulated values of the generalized error term $s$ for a sinusoidal input to a two link manipulator .................................................. 33
4-17 Estimated values of the adapted parameters for a sinusoidal input to a two link manipulator .................................................. 34
Chapter 1

Introduction

One important technique in the field of modern nonlinear control is adaptive control. It has been mathematically proven that for a properly formulated adaptive controller the response of the controlled system converges to a desired trajectory with exponentially vanishing transient affects[1]. The most basic form of the algorithm uses online parameter adaptation to cancel all of the modeled dynamics of the system and then performing simple proportional control on a generalized error term that takes into account the position and velocity errors of the system. It also requires constant parameters defining the speed of adaptation, the proportional gain, and the relative weighting of position and velocity errors. A controller therefore requires only the dynamics and several appropriately scaled constants to perform tracking control.

The adaptive control library written for this project was designed to take advantage of the fact that the algorithm is generalized and depends only on the dynamics of the system. This allowed the library to hide many of the details of the controller for a user that requires tracking control but can afford relatively long training times. Modifications to the algorithm can be made by modifying the dynamics, allowing the user, for example, to leave natural damping out of the adapted dynamics. The library allows an advanced user to plug in a custom control or adaptation law using specific knowledge of the system under control.

Adaptive control is typically not taught in a general undergraduate introductory course on control theory, perhaps due to the formal mathematics required for the
derivation along with the wide applicability and richness of linear control theory. In addition, there is little material easily available to assist the novice in learning adaptive control. This project therefore also included interactive software allowing the user to visualize the algorithm in real time on a specific hardware device through a PC connected to an ORC Board. This software allowed a user to modify the parameters of the algorithm online and observe in real-time the affects on the adapted model. It therefore illustrated the algorithm graphically, allowing a user new to adaptive control to understand the affects of the controller without knowledge of the detailed mathematics behind its derivation.
Chapter 2

Relevant Control Theory

The type of control problem treated in this study was the general trajectory control of a physical nonlinear plant, with the important limitations that the system be

- Fully actuated
- Fully observable
- Energy conservative
- Dominated by known linearly parameterized nonlinear dynamics

The first two constraints are physical constraints; that is, the system must have sufficient actuation to independently control the various degrees of freedom of the system, and enough sensing to determine the complete state (including all the derivatives) of the system. The third constraint limits the application of the controller to mechanical-like systems with the only power input source as the controlled actuators. The fourth constraint is both a constraint on the system and the designer. First, the system must be linearly parameterizable in terms of known state variables and unknown constant or slowly varying parameters. Formally, the system must be characterized by the relation

\[ \tau = Y(t, q, \dot{q}, \ddot{q}, ...) a, \]   \hspace{1cm} (2.1)
where $\tau$ is a vector of control inputs into the system, $a$ is a column vector of constant or slowly changing parameters, and $q$ is the state vector of the system. The type of system applicable to the basic control law is of the form

$$\tau = H\ddot{q} + C\dot{q} + G.$$  \hspace{1cm} (2.2)

The $H$ matrix can be interpreted as a mass, the $C$ matrix damping, and the $G$ matrix a steady state gravitational force. The constraint on the designer is that the $Y$ matrix must be known. In addition, the trajectory must be twice differentiable, so a certain smoothness is required for the mathematical guarantee of convergence to hold. It shall be shown, though, that discontinuous trajectories can be handled by the library, as all trajectories in digital representations are inherently discontinuous. The performance simply degraded for trajectories with a more discontinuous nature.

### 2.1 Adaptive Control Library

The adaptive controller was designed based on Lyapunov stability theory[1]. In particular, the algorithm dealt with the convergence of a generalized error quantity

$$s = \dot{q} + \lambda\ddot{q}$$  \hspace{1cm} (2.3)

where $\ddot{q}$ denotes the position error of the actual position vector from the desired position vector $q_d$

$$\ddot{q} = q - q_d$$  \hspace{1cm} (2.4)

and $\lambda$ is a positive definite matrix supplied by the user which weights the relative importance of position error versus velocity error. In addition, it is helpful to define a reference velocity

$$\dot{q}_r = \dot{q}_d - \lambda\ddot{q},$$  \hspace{1cm} (2.5)

which is the desired velocity modified by the current position error. The controller is then designed through a Lyapunov analysis of the controlled system. Taking the
Lyapunov function candidate to be

\[ V(t) = \frac{1}{2} \left[ s^T H s + \ddot{a}^T \Gamma^{-1} \ddot{a} \right], \]  

(2.6)

differentiating, substituting in the dynamics equation with the definitions of \( q_r \) and \( s \), and using energy conservation to cancel the \( \dot{H} \) term [1, p. 402], yields

\[ \dot{V}(t) = s^T (\tau - H \ddot{q}_r - C \dot{q}_r - G) + \ddot{a}^T \Gamma^{-1} \ddot{a}. \]  

(2.7)

Taking the control law to be

\[ \tau = Y \ddot{a} - K_d s \]  

(2.8)

with an adaptation law

\[ \dot{a} = -\Gamma Y^T s \]  

(2.9)

where \( K_d \) and \( \Gamma \) are positive definite matrices, gives

\[ \dot{V}(t) = -s^T K_d s \leq 0. \]  

(2.10)

This conclusion implies, by Barbalat’s Lemma [1, p. 122], that the generalized error term \( s \) converges to zero as time goes to infinity. Since \( s \) can be interpreted as a first order filter of the position error of the system, the velocity and position errors also tend to zero.
Chapter 3

Description of Software

3.1 Adaptive Control Library

The module dependency diagram of the software library can be seen in figure (3-1).

Figure 3-1: Object Inheritance diagram, showing the class hierarchy of the types used in the controller. Large errors denote inheritance while small arrows imply usage.

The base class is the *AdaptiveController*, which provides the computational framework for the algorithm. This framework allows the user to specify the adaptation and
control laws and handles the low level integration and parameter storage. It uses the Runge-Kutta 4 method to integrate the adaptation law to determine new parameters. The control and adaptation laws exist formally in the system as interfaces that can be implemented and then passed to the controller. For this study the BasicAdaptiveController is used, which packages the control and adaptation law outlined above for an energy conservative system with inertia, damping, and gravity. This controller is then created by specifying the adaptation gains, the feedback gains, and the dynamics of the system. The dynamics are defined by subclassing NonLinearDynamicDependencies and passing the class into the controller. An InstrumentedAdaptiveController can also be created using any other AdaptiveController, which creates a transcript of the run in a MATLAB® readable format, as described in appendix B.

An extension to the library to handle, for example, unmodeled noise in the system, would only require the definition of a new control law and adaptation law. This could then extend the BasicAdaptiveController with the added functionality. This example is worked out in more detail in appendix A.

3.2 Visualization Software

The visualization software is used to show the affects of the algorithm on an a DC motor. The software interfaces to an ORC board via a serial cable. For the purposes of this study, only the hardware device was controlled using the GUI. Future work would involve more options including multiple hardware systems, the ability to run simulations in addition to hardware, and the ability to change controller dynamics in the GUI. The main screen of the visualization software, shown in figure (3-2), presents a view of a motor's current velocity and commanded velocity, along with interface commands. The commanded velocity can be changed by the buttons B and C, or
commanded manually using the slider D. The feedback gain to the motor can be edited using slider E. The label F shows the updated Pulse Width Modulation value sent to the motor, and G displays the measured velocity of the motor. The graph I displays the velocity history of the motor, and the start/stop button J controls whether or not control commands are sent to the motors. The duplicate panel controls the other motor on the machine. All trials were run using the right motor. The adaption panel button K brings up the interface shown in figure (3-3). Each adapted parameter
Figure 3-3: The adaptation control panel, allowing the user to modify and observe that adaptation in real time

is plotted over time in the graphs D, and the respective adaptation gains for the parameters can be set using the slider A. The scale over which the slider spans can be changed using the top B and bottom C range setters. These values are updated in real time and help the user find appropriate gains for the given hardware and trajectory. Each motor has two adapted parameters, the two graphs on the right correspond to the right motor, similar for the left.
Chapter 4

Simulation and Experiment

The library was exercised on three control problems, two simulations and a hardware system. The simulations illustrate the theoretical correctness of the dynamics chosen and the underlying controller library. The hardware example was chosen to illustrate robustness to model inaccuracies, as the model used for the adapted dynamics was a simple linear representation, while the real system exhibited nonlinear behavior. The experimental apparatus used is pictured in figure 4-1.

Figure 4-1: Picture of the experimental assembly, a motor with attached wheel controlled by an ORC board.
4.1 DC Motor Example

The DC motor provided a simple hardware test case, and the first simulation was the first order model of the motor using the nominal manufacturer’s parameters. The dynamics of this model

\[ \tau = a_1 \ddot{\theta} + a_2 \dot{\theta}, \quad (4.1) \]

where \( \tau \) in this case is the applied voltage \( v \), were used as the adapted dynamics of the controller for both the simulated and hardware examples. The linear model of a motor,

\[ v = \frac{JR}{K_t} \ddot{\theta} + K_e \dot{\theta} \quad (4.2) \]

depends upon its electro-mechanical properties \( K_e \), the back-EMF constant, \( K_t \), the torque constant, \( R \), the resistance of the motor, and \( J \), the load rotational inertia.

4.1.1 Simulation Results

The simulation of the controller’s performance on the model is shown in figures 4-2 and 4-3. The desired velocity is a series of steps with an increment of \( 2 \text{ rad} \) and an increment of \( 2 \text{ rad} \), and the desired position is started at an offset of 2 radians. The constant controller values used were

\[ \lambda = [0.1] \quad K_d = [1] \quad \Gamma = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \quad (4.3) \]

The adapted parameters throughout the trial are shown in figures 4-4 and 4-5, and the generalized error term \( s \) over time is shown in figure 4-6.
Figure 4-2: Simulated and desired position trajectories for a DC motor

Figure 4-3: Simulated and desired velocity trajectories for a DC motor
Figure 4-4: The first adapted parameter and actual model value over time

Figure 4-5: The second adapted parameter and actual model value over time
Figure 4-6: The generalized error $s$ of the simulated DC motor

The controller displayed the desired behavior by converging to the desired position and velocity trajectories. The velocity step series showed the standard problem of velocity control at discrete values, but with the added component of simultaneous position control. The controller parameter $\lambda$ controls the relative importance of the position and velocity error. This case also provided an example of the adapted parameters not necessarily converging to the modeled values, but instead growing to increase the performance of the controller. This phenomenon was due to the properties of the desired trajectory; the steps make $a_1$ a particularly difficult parameter to adapt to, as the near discontinuities stretch the requirement that the desired trajectories must be differentiable.
4.1.2 Experimental Results

The controller was then tested on the hardware, using

\[
\lambda = [0.1] \quad K_d = [1] \quad \Gamma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]  \quad (4.4)

The controller’s trajectory performance on the hardware motor is shown in figure 4-7 and 4-8, along with the results of the simulation on the same trajectory and parameters using the model above. The adapted parameter estimates are shown in figures 4-10 and 4-11, and control voltages and the generalized error term \( s \) in figures 4-9 and 4-12, respectively.

![DC motor test, position over time](image)

Figure 4-7: Simulated, experimental, and desired position trajectories for a sinusoidal input
Figure 4-8: Simulated, experimental, and desired velocity trajectories for a sinusoidal input

Figure 4-9: Simulated and experimental control voltages for a sinusoidal input for a DC motor
Figure 4-10: Simulated, experimental, and modeled $a_1$ values for a sinusoidal input

Figure 4-11: Simulated, experimental, and modeled $a_2$ values for a sinusoidal input
Figure 4-12: Simulated and experimental generalized error $s$ values for a sinusoidal input for a DC motor

One striking difference between the hardware and simulation performance was observed at the beginning of the run, when the velocity remained at zero as the system was overcoming static friction. The other result of this difference between the modeled dynamics and the simulated dynamics was that with all parameters equal the hardware controller provided better position tracking initially than the simulated counterpart. This was an artifact of the particular trajectory chosen, where the friction effects kept the position closer to the desired position as the velocity ramped up. However, it also leads to the observation that some components of the damping are helpful, depending on the desired trajectory. For example, in a pick and place task a manipulator must move from one position to another as quickly as possible with no overshoot. In this case, the damping effects of the manipulator help the controller to stop during the place part of the task, it would therefore be counterproductive to include the damping dynamics in the $Y$ matrix, as this adaptively cancels them. There are also ways to include damping in adaptive control to increase the performance of the system that only involve changes to the $Y$ matrix[1, p. 408].
4.2 Two Link Simulation

The software was tested on a more complicated simulated system, the two link manipulator. The links were modeled as rigid elements, and they were configured in the horizontal axis. Each link of an n-link manipulator in 2 dimensions can be defined by its mass, length, moment of inertia, and the distance of the center of mass from the joint, here denoted $m_i$, $L_i$, $I_i$, $L_{ci}$, respectively, for any joint $i$. The dynamic equations for this configuration yield[1, p. 396]

$$\tau = H\ddot{q} + C\dot{q}$$  \hspace{1cm} (4.5)

where the inertia matrix $H$ is defined by

$$H_{1,1} = a_1 + 2a_3 \cos q_2$$  \hspace{1cm} (4.6)

$$H_{1,2} = H_{2,1} = a_2 + a_3 \cos q_2$$  \hspace{1cm} (4.7)

$$H_{2,2} = a_2$$  \hspace{1cm} (4.8)

where

$$a_1 = I_1 + m_1 L_{c1}^2 + I_2 + m_2 L_{c2}^2 + m_2 L_1^2$$  \hspace{1cm} (4.9)

$$a_2 = I_2 + m_2 L_{c2}^2$$  \hspace{1cm} (4.10)

$$a_3 = m_2 L_1 L_{c2}.$$  \hspace{1cm} (4.11)

The damping matrix $C$ is defined by

$$C_{1,1} = -h\dot{q}_2 \cos q_2$$  \hspace{1cm} (4.12)

$$C_{1,2} = -h(\dot{q}_1 + \dot{q}_2)$$  \hspace{1cm} (4.13)

$$C_{2,1} = h\dot{q}_1$$  \hspace{1cm} (4.14)

$$C_{2,2} = 0$$  \hspace{1cm} (4.15)
The dynamics used in the controller were

\[
Y = \begin{bmatrix}
\ddot{q}_r1 & \ddot{q}_r2 & y_1 \cos(q_2) - y_2 \sin q_2 & y_1 \sin(q_2) + y_2 \cos q_2 \\
0 & \ddot{q}_r2 + \ddot{q}_r1 \cos q_2 + \dot{q}_r1 \dot{q}_r1 \sin q_2 & \ddot{q}_r1 \sin q_2 - \dot{q}_r1 \dot{q}_r1 \cos q_2 \\
\end{bmatrix}
\]  

(4.16)

where

\[
y_1 = 2\ddot{q}_r1 + \ddot{q}_r2 \\
y_2 = \dot{q}_2 \dot{q}_r1 + \dot{q}_1 \dot{q}_r2 + \dot{q}_2 \dot{q}_r2
\]

(4.17)  

(4.18)

The controller was simulated using

\[
\lambda = \begin{bmatrix}
2 & 0 \\
0 & 2
\end{bmatrix}, \quad K_d = \begin{bmatrix}
10 & 0 \\
0 & 10
\end{bmatrix}, \quad \Gamma = \begin{bmatrix}
0.5 & 0 & 0 & 0 \\
0 & 0.5 & 0 & 0 \\
0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0.5
\end{bmatrix}
\]

(4.19)

the results of the simulation are shown in figures 4-13 through 4-17
Figure 4-13: Simulated and desired position trajectories for a sinusoidal input

Figure 4-14: Simulated and desired velocity trajectories for a sinusoidal input
Figure 4-15: Simulated torques for a sinusoidal input to a two link manipulator

Figure 4-16: Simulated values of the generalized error term $s$ for a sinusoidal input to a two link manipulator
Figure 4-17: Estimated values of the adapted parameters for a sinusoidal input to a two link manipulator
The simulation exhibited the desired behavior, with the generalized tracking error generally tending to zero. The coupling of coriolis effects were observed on the first joint as expected, making the tracking problem more difficult for that joint. The controller was built using only a single Java class which defined the matrix $Y$ such that

$$ Y = H\ddot{q}_r + C\dot{q}_r $$

All the knowledge required to design this class was therefore the dynamic dependencies of the inertia and damping matrices, and therefore this same controller class will work for any two link manipulator. The other task in building the controller was finding suitable gains, and the visualization software was designed to build intuition to assist with this task.
Chapter 5

Conclusions

The adaptive control library was found to be a good tool for solving control tasks in a generalizable way. For a few specific systems that can be modeled as a certain class of mechanical system as presented above, the basic controller was found to handle the control task given only an appropriate set of gains. In addition, the library, through modular design, was easily extendible to handle more advanced cases with a minimum amount of work. An advanced user could therefore use the library as a tool to perform research in adaptive control and online model estimation, while a beginner could use the library for the practical purpose of controlling the trajectory of a mechanical system without having to deal with the numerical issues involved in the updating model.

The visualization software was also presented to assist the beginning user to build intuition for the effects of the various parameters of the algorithm. This was considered an important facet of the project because analytical studies of nonlinear systems involve mathematics typically beyond the undergraduate level. Bringing these controllers, which are inherently nonlinear even when controlling linear systems, to practical problems in everyday use requires intuition on the part of the designer. A more complete tool set is required to realize this goal, though. The visualization software currently has no method of altering the underlying controller in real time, nor is there a good way to test and visualize multiple degree of freedom systems. It is thought, however, that such a change would not be difficult to make, but more work
would need to go into the user interface to help the user make better sense of the response.

The goal of this project was to make tools that help the user both implement and learn about adaptive control. By writing the software in Java, it was hoped that it would be widely cross platform and easy to integrate into existing systems, along with promoting reuse of code. It is hoped that the library will make adaptive control a more attractive algorithm for researchers and hobbists who require high performance tracking control, and it is hoped that the visualization software can help build intuition and appreciation for adaptive control.
Appendix A

Example Library Extension: Noise Resistant Controller

One common problem in solving control problems on an actual hardware device is the affect of noise in the sensor values. In the particular case of adaptive control, noise in the system is particularly deleterious as the controller will try to adapt the parameters (which do not model the noise) to cancel the noise. This is a case of over-fitting the controller, where the model is conformed to fit dynamics that it is not intended to account for. This leads to an inaccurate adapted model which can lead to unstable behavior. A solution to this problem is to state a bound on the noise that the system is likely to endure, and simply turn off adaptation and control when the trajectory is within those bounds. In effect, this tells the controller to assume it is performing perfectly when within the bounds, and therefore it does not change the model or attempt to get closer to the nominal desired trajectory. This yields a tradeoff, therefore, between tracking performance and noise rejection. In a properly designed controller, this simple method is effective. Incidentally, it is also straightforward to implement in the library proposed in this study.

One implementation of this behavior involves the definition of a new BasicControlLaw and BasicAdaptationLaw, the SlidingControlLaw and SlidingAdaptationLaw so called due to links to sliding control theory, which involves convergence to a surface, or, in the presence of noise, the region surrounding a surface. Presented here
is a possible implementation of the adaptation behavior, which takes as parameters
the noise threshold values for each degree of freedom of the system. Values of $s$ that
are smaller in magnitude than the noise threshold are clamped to zero before being
passed to the adaptation law. The SlidingControlLaw has a similar implementation.

```java
public class SlidingAdaptationLaw extends BasicAdaptationLaw {

    private double[] noiseThresholds;

    public SlidingAdaptationLaw(Matrix adaptationGains,
                                double[] noiseThresholds) {
        super(adaptationGains);
        this.noiseThresholds = noiseThresholds;
    }

    @Override
    public Matrix adapt(Matrix knownDynamicDependences,
                         Matrix generalizedError) {
        for (int i = 0; i < noiseThresholds.length; i++) {
            // for each dimension of the generalized error, if the error is
            // below the largest expected noise value
            if (Math.abs(generalizedError.get(i, 0)) < noiseThresholds[i]) {
                // set that term of the error to zero
                generalizedError.set(i, 0, 0.0);
            }
        }
        // now send the new generalized error term through to the
        // BasicAdaptationLaw
        return super.adapt(knownDynamicDependences, generalizedError);
    }
}
```
Appendix B

Storage File Format

The InsturmentedAdaptiveController is created with a file name to which it will output the log and an AdaptiveController that it uses as the underlying controller (i.e. all method calls to the InsturmentedAdaptiveController are passed directly through to the controller used in construction). This allows any later subclass to use take advantage of instrumentation. The file is created as soon as the controller starts running, and will continue adding data at each call to the control loop. In principle, therefore, if the run is interrupted at any time, most of the data that has been collected will be in the file. However, to make the data automatically readable by MATLAB, an end bracket must be added through use of the closeFile() method. If this is not done than the bracket must be manually added.

Each line in the log file represents a single control loop iteration and contains all the information the controller has at that time. The file has the general form

```plaintext
filename = [
    [time_one, positions, velocities, desiredPositions, desiredVelocities,
        desiredAccelerations, controlValues, generalizedError, aGuesses]
    ...
    [time_n, positions, velocities, desiredPositions, desiredVelocities,
        desiredAccelerations, controlValues, generalizedError, aGuesses]];
```

41
where filename is the name of the file that was passed into the controller.

Each of the elements of the array in a given line, except time, can be several comma separated values. For example, the two link manipulator logs had 19 comma separated values per line, one for time, two each for the next seven variables, and four for the adapted parameters. There are no extra brackets to delineate the individual vectors. The position vector over time for the first degree of freedom for example can be plotted using the MATLAB command

\[
\text{plot}(\text{filename}(:,1), \text{filename}(:,2))
\]
Bibliography