Observer Based Approach to the Passive Control of Bilateral Teleoperated Manipulators

Benjamin Peirce
ME Honors Thesis

Advisor
Perry Y. Li

August 15, 2000
Abstract

Passive control of bilateral teleoperated manipulators using a state observer to estimate environmental forces is demonstrated. The system is designed to be energetically passive so that it remains stable in most environments and is safe for humans to interact with. Energetic passivity requires knowledge of the forces input to the system by the environment. Instead of directly measuring these environmental forces with force sensors, the controller estimates the forces using a state observer that assumes them to be constant. This simplified approach is shown through simulation and physical experimentation to be a possible alternative to force sensors for basic manipulation tasks.
Contents

1 Introduction ....................................................... 7

2 Theory .......................................................... 9
   2.1 System Decoupling ........................................ 9
   2.2 Passive Dynamics ........................................ 11
   2.3 Force Estimation .......................................... 13

3 System Identification ........................................ 15
   3.1 Manipulator Model ....................................... 15
   3.2 Data Acquisition ......................................... 17
   3.3 Data Analysis ............................................ 17

4 Controller Design and Verification ....................... 21
   4.1 Simulink Implementation ............................... 21
   4.2 Observer Design ......................................... 21
   4.3 Frequency Response ..................................... 23

5 Conclusion ........................................................ 27

A Program Code ....................................................... 31
   A.1 acceltest.m ............................................... 31
   A.2 accelcalc.m ............................................... 32
   A.3 armid.m .................................................. 33
   A.4 arm2id.m .................................................. 33
   A.5 observerdesign.m ....................................... 34
   A.6 controller.m ............................................. 34
   A.7 plantmodel.m ............................................ 37
   A.8 plant_hardware.m ....................................... 39
List of Figures

2.1 Shape and locked system geometry. ........................................... 10
2.2 Hyperbolic behavior of $g(f)$. ................................................... 12
3.1 Physical teleoperator system. ...................................................... 16
3.2 Large arm identification results. ............................................... 18
3.3 Small arm identification results. ............................................... 19
4.1 Simulink simulation window. .................................................... 22
4.2 Simulation frequency response with different speed observers. ......... 23
4.3 Simulation and hardware frequency response comparison. ............... 24
Chapter 1

Introduction

Teleoperated manipulators are bilateral when inputs at either slave or manipulator produce outputs at on the opposite link. Each manipulator has both input and output and the goal is for the slave to accurately follow the motion of the master manipulator. It is in this way that the human interacts the environment.

Examples of teleoperator applications include microsurgery and earth moving / construction, each of which requires kinematic and force scaling. The idea with these systems is to establish a “telepresence” for the human. That is, the feeling of interacting directly with the environment without the teleoperators in between. This is accomplished by transmitting the forces between the human and environment through the manipulators.

One concern is that the system operates safely when a human is interacting with it and does not become unstable. To maintain stability, energetic passivity is applied to limit its energy output to no more than the energy input. In this way the system behaves like a mechanical coupling with the possibility of storing energy, but not creating it. The large amounts of energy output necessary for a system to become unstable are therefore not available.

Energy is transferred into and out of the system when the manipulators become un-coordinated and the controller attempts to correct them. This relative motion between the manipulators that measures the coordination is referred to as the shape system. The overall motion of both manipulators simultaneously then becomes the locked system. The controller is then only concerned with minimizing the error in the shape system. [1]

As mentioned above, energetic passivity requires the system output force to be no greater than the system input force. To maintain this relationship, it is necessary to know the environment and human input forces (simply referred to as environmental forces). Normally this is achieved by using force sensors to directly measure these forces. The proposed method will instead attempt to estimate these input forces using a state observer.

In order to apply a state observer, it is necessary to have a model of environmental forces that are to be observed. Since these forces are unknown, it is assumed here that they are constant. Based on this assumption, an observer is designed and implemented in the controller to provide the necessary environmental force information. The versatility of the above assumption is proven in both simulation and on the physical system.

The thesis is outlined as follows. This first chapter serves as an introduction to the problem. The second chapter introduces the theory behind the system passivity and force ob-
server. The third chapter describes the system identification. The fourth chapter describes the implementation of the controller on a physical system. The fifth chapter provides the conclusion and remarks.
Chapter 2

Theory

2.1 System Decoupling

The main goal of controlling bilateral teleoperators is to minimize coordination error between the two arms. The dynamics of the arms can be described as

\[
M_1 \ddot{q}_1 = T_1 + F_1 \\
M_2 \ddot{q}_2 = T_2 + F_2
\]

where \( q_1 \) and \( q_2 \) are the positions and \( M_1 \) and \( M_2 \) are the inertias of arms 1 and 2, respectively. The control forces for each arm are denoted by \( T_1 \) and \( T_2 \) and the environmental forces acting on those arms are denoted by \( F_1 \) and \( F_2 \).

Since the coordination of the manipulators can be expressed in one rather than two variables, it is desirable to decouple the system into a shape and a locked system [1]. The shape system considers only the relative displacement between the two links while the locked system considers only the absolute position of the shape system. The complexity of the control problem may be reduced by focusing only on the shape system and ignoring the locked system.

The linear transformation from two arm position variables, \( q_1 \) and \( q_2 \), to the coordination error of the shape system, \( E \), and the position of the locked system, \( L \), is calculated as

\[
\begin{bmatrix}
E \\
L
\end{bmatrix} = S \begin{bmatrix}
q_1 \\
q_2
\end{bmatrix}
\]

where the transformation with equal inertias on both arms and no scaling is defined by

\[
S = \begin{bmatrix}
\frac{1}{2} & -\frac{1}{2} \\
\frac{1}{2} & \frac{1}{2}
\end{bmatrix}
\]

This geometry is illustrated in Figure 2.1.

The transformation can be modified to account for unequal inertias, kinematic scaling, and power scaling, as follows

\[
S = \frac{\zeta}{\rho + \zeta} \begin{bmatrix}
\frac{\alpha}{\xi} & -1 \\
\frac{\alpha}{\xi} & 1
\end{bmatrix}
\]

(2.4)
where the kinematic scaling is defined by
\[ \alpha q_1 = q_2 \]
the force scaling is defined by
\[ \rho F_1 = F_2 \]
and \( \zeta \), the ratio of the arm inertias, is defined by
\[ \zeta M_1 = M_2 \]

[1]. The implemented controller will first transform \( q_1 \) and \( q_2 \) into \( E \) and \( L \) and then minimize \( E \). The control force required to minimize \( E \) in the shape system is then converted to control forces for each arm and output to the plant.

The system dynamics of (2.1) and (2.2) may be rewritten as
\[
\begin{bmatrix}
M_1 & 0 \\
0 & M_2
\end{bmatrix}
\begin{bmatrix}
\ddot{\bar{q}}_1 \\
\ddot{\bar{q}}_2
\end{bmatrix}
= \begin{bmatrix}
T_1 \\
T_2
\end{bmatrix}
+ \begin{bmatrix}
F_1 \\
F_2
\end{bmatrix}
\]

and may be expressed in terms of \( E \) and \( L \) by applying the transformation (2.3) as
\[
\begin{bmatrix}
M_1 & 0 \\
0 & M_2
\end{bmatrix}S^{-1}
\begin{bmatrix}
\ddot{E} \\
\ddot{L}
\end{bmatrix}
= \begin{bmatrix}
T_1 \\
T_2
\end{bmatrix}
+ \begin{bmatrix}
F_1 \\
F_2
\end{bmatrix}
\]

To keep the inertia matrix symmetric, (2.1) is multiplied by the transpose of \( S^{-1} \) to produce
\[
S^{-T}
\begin{bmatrix}
M_1 & 0 \\
0 & M_2
\end{bmatrix}S^{-1}
\begin{bmatrix}
\ddot{E} \\
\ddot{L}
\end{bmatrix}
= S^{-T}
\begin{bmatrix}
T_1 \\
T_2
\end{bmatrix}
+ S^{-T}
\begin{bmatrix}
F_1 \\
F_2
\end{bmatrix}
\]

The transformations can then be multiplied through to give the dynamics in terms of the shape and locked systems as
\[
\begin{align*}
M_E \ddot{E} &= T_E + F_E \quad (2.5) \\
M_L \ddot{L} &= T_L + F_E \quad (2.6)
\end{align*}
\]
where

\[
\begin{bmatrix}
T_E \\
T_L
\end{bmatrix} = S^{-T} \begin{bmatrix} 
T_1 \\
T_2
\end{bmatrix}, \quad \begin{bmatrix}
F_E \\
F_L
\end{bmatrix} = S^{-T} \begin{bmatrix} 
F_1 \\
F_2
\end{bmatrix} \tag{2.7}
\]

It can also be shown that the system of (2.5) and (2.6) has the same kinetic energy as that of (2.1) and (2.2) [1].

### 2.2 Passive Dynamics

The basic controller implements proportional and derivative control on the shape system to minimize $E$. The control output of (2.5) is therefore expressed as

\[
T_E = -KE - b\dot{E} - F_E \tag{2.8}
\]

where the first and second terms on the R.H.S. reflect the dynamics of the error minimization and the third term cancels the environmental force input. To be passive, however, it must maintain a total of zero energy in the system.

The four methods of energy transfer and storage in the system are the spring action of the proportional gain inside the controller, $KE$, the controller output to the plant, $T_E$, the environment force input to the system, $F_E$, and an internal variable implemented to balance the first three. This internal variable will act as a “flywheel” to store energy when a surplus is input to the system and release energy when it is drawn from the system. The “flywheel velocity” will be denoted by $f$ and the force it produces on the system will be denoted by $T_f$ defined as

\[
T_f = M_f \dot{f}
\]

where $M_f$ is the “flywheel inertia.”

Since $F_E$ is an unknown variable it must be either measured or predicted. For experimental simplicity, this controller will attempt to predict $F_E$ under the assumption that it is constant. The details of this force prediction are discussed in Section 2.3. The multiplication of the energy transferring forces by their corresponding velocities gives the overall power output of the system. The condition for passivity is therefore

\[
EK\dot{E} + \dot{E}T_E + fT_f = 0 \tag{2.9}
\]

It is also observed that $E$, $\dot{E}$, and $f$ are the necessary state variables to maintain (2.9). The state variables are separated from the L.H.S. of (2.9) as follows

\[
EK\dot{E} + \dot{E}T_E + fT_f = \begin{bmatrix} E & \dot{E} & f \end{bmatrix} \begin{bmatrix} K \dot{E} \\
T_E \\
T_f \end{bmatrix} \tag{2.10}
\]

Using (2.8), the column vector on the R.H.S. of (2.10) can be decomposed into the product of a skew symmetric matrix and the state variables;

\[
\begin{bmatrix}
K\dot{E} \\
T_E \\
T_f
\end{bmatrix} = \begin{bmatrix} 
0 & K & 0 \\
-K & 0 & -\frac{b\dot{E} + \dot{E}_E}{f} \\
0 & \frac{b\dot{E} + \dot{E}_E}{f} & 0
\end{bmatrix} \begin{bmatrix} E \\
\dot{E} \\
f
\end{bmatrix} \tag{2.11}
\]
where the dynamics of $T_f$ are defined to make the matrix skew symmetric.

It is important that the matrix in (2.11) be skew symmetric since it allows (2.9) to be satisfied and the system to behave passively regardless of the values of the matrix elements, as follows

$$\begin{bmatrix} E & \dot{E} & f \end{bmatrix} \begin{bmatrix} 0 & K & 0 \\ -K & 0 & \frac{b\dot{E} + \hat{F}_E}{f} \\ 0 & \frac{b\dot{E} + \hat{F}_E}{f} & 0 \end{bmatrix} \begin{bmatrix} E \\ \dot{E} \\ f \end{bmatrix} = 0$$

The purpose of the controller is to continuously integrate the state-space system

$$\begin{bmatrix} K\dot{E} \\ M\dot{E} \\ M_f\dot{f} \end{bmatrix} = \begin{bmatrix} 0 & K & 0 \\ -K & 0 & \frac{b\dot{E} + \hat{F}_E}{f} \\ 0 & \frac{b\dot{E} + \hat{F}_E}{f} & 0 \end{bmatrix} \begin{bmatrix} E \\ \dot{E} \\ f \end{bmatrix}$$

and output the control force according to (2.8).

One consideration in the above control scheme is the presence of $f$ in the denominator of two matrix elements. These elements increase hyperbolically with decreasing $f$ and would make it impossible to calculate (2.12) on a computer if $f$ were too small or equal to zero.

To solve this problem, $\frac{1}{f}$ is replaced by $g(f)$ defined as

$$g(f) = \begin{cases} \frac{1}{f}, & |f| > f_0 \\ \frac{1}{f_0} \cdot \text{sign}(f), & 0 < |f| \leq f_0 \\ \frac{1}{f_0}, & |f| = 0 \end{cases}$$

[2] with behavior illustrated in Figure 2.2. The applied version of (2.12) then becomes

$$\begin{bmatrix} K\dot{E} \\ M\dot{E} \\ M_f\dot{f} \end{bmatrix} = \begin{bmatrix} 0 & K & 0 \\ -K & 0 & -(b\dot{E} + \hat{F}_E)g \\ 0 & (b\dot{E} + \hat{F}_E)g & 0 \end{bmatrix} \begin{bmatrix} E \\ \dot{E} \\ f \end{bmatrix}$$

Figure 2.2: Hyperbolic behavior of $g(f)$. 

12
2.3 Force Estimation

To avoid using expensive and experimentally difficult force sensors, a state observer was used to predict $F_E$. This required the creation of a system model using $\hat{E}$ and $F_E$ as state variables. The dynamics of $\hat{E}$ were taken from (2.5). Since the behavior of $F_E$ is unknown without force sensors, it was assumed that $\hat{F}_E = 0$. This represents a constant environmental force input and produced the system model

$$ \frac{d}{dt} \begin{bmatrix} \hat{F}_E \\ \hat{E} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \frac{1}{M_E} & 0 \end{bmatrix} \begin{bmatrix} \hat{F}_E \\ \hat{E} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M_E} \end{bmatrix} T_E $$

$$ \dot{\hat{E}} = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{F}_E \\ \hat{E} \end{bmatrix} $$(2.15) (2.16)

which could be tested for observability with

$$ A = \begin{bmatrix} 0 & 0 \\ \frac{1}{M_E} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{M_E} \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \end{bmatrix} $$

according to the standard observability matrix [3] as follows

$$ \begin{vmatrix} C^* & A^* C^* \end{vmatrix} = -\frac{1}{M_E} \neq 0 $$

A state observer can then be constructed according to

$$ \hat{x} = A \hat{x} + Bu + K_c (y - C \hat{x}) $$

which, when used to on the system of (2.16) and (2.16), becomes

$$ \frac{d}{dt} \begin{bmatrix} \hat{F}_E \\ \hat{E} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \frac{1}{M_E} & 0 \end{bmatrix} \begin{bmatrix} \hat{F}_E \\ \hat{E} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M_E} \end{bmatrix} T_E + K_c \left( \hat{E} - \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{F}_E \\ \hat{E} \end{bmatrix} \right) $$

(2.17)

where $K_c$ is determined based on the desired performance characteristics of the observer. A MATLAB program was created for the determination of $K_c$ and is described in Section 4.2. Once the observer was designed, it could be implemented in the controller to predict $F_E$ for use in (2.14).
Chapter 3

System Identification

Since an accurate model of the system was necessary for force prediction, the system needed to be properly identified. The physical system is shown in Figure 3.1. The system identification was carried out as described below.

3.1 Manipulator Model

Identification of the behavior of the two-link system required finding each arm’s inertia according to

$$\tau = M \alpha$$

where $\tau$ is torque, $\alpha$ is angular acceleration and $M$ is the mass moment of inertia. The specific equations for this system were

$$c_1 = M_1 \alpha_1$$
$$c_2 = M_2 \alpha_2$$

where $c_1$ and $c_2$ are the command signals sent to the large and small arms, respectively, and represent the unit-less torque input. $M_1$ and $M_2$ are the inertias and $\alpha_1$ and $\alpha_2$ are the angular accelerations for the large and small arms, respectively.

The independent variable was the command input, $c$, and the dependent variable was the angular acceleration, $\alpha$, which was calculated by fitting a second degree polynomial to the arm’s angular displacement versus time curve. The polynomial was

$$\theta = at^2 + bt + c$$

which could be time differentiated twice to find the angular acceleration in terms of the first coefficient of the fitted polynomial,

$$\ddot{\theta} = \alpha = 2a$$

Once the angular accelerations were found for each command input, a line could be fit to the collection of $\alpha$’s versus $c$’s to find $M$ according to (3.1) and (3.2).
Figure 3.1: Physical teleoperator system.
3.2 Data Acquisition

The range of commands input to each arm was determined by the hardware limitations. The commands needed to be large enough to cause the arm to overcome static friction and small enough to allow the computer time to capture the parabolic acceleration curve before it became linear (i.e., friction force balanced input force and acceleration ceased). The latter limitation was the result of a minimum time of one second needed to acquire the one hundred data points used in the displacement curves. Since the displacement curve is only parabolic near the origin, and this parabolic region decreases with increasing acceleration, useful data could only be gained from commands small enough to produce parabolic displacements during the entire first second. To ensure fitting of a parabolic curve, only the first fifty of the captured points were used in the polynomial fit.

Ten acceleration curves were recorded for each command input to each arm to allow computation of a precision uncertainty. The average acceleration was used for calculation of $M$ and a $t$-distribution was assumed to calculate the precision uncertainty as

\[ \Delta \alpha = \frac{\sigma t_{0.25,9}}{\sqrt{10}} \]

where $\sigma$ is the standard deviation of the ten acceleration curves and $t_{0.25,9}$ is the $t$-statistic for 95% confidence and 9 degrees of freedom.

The uncertainty introduced by inaccurate polynomial fitting to the displacement curve is not represented in the acceleration uncertainty. Since fifty points were used in the least squares fit it was assumed that errors from this the fit could be ignored.

The calculation of $M_1$ and $M_2$ required the solution of (3.1) and (3.2) for each command input and respective acceleration. For each arm, a first-degree polynomial was fit to the collections of $\alpha$ versus $c$ and its slope was interpreted as $M$.

The collection of points recorded was only a small portion of the operating range making it more important to obtain a linear fit with a slope parallel to the data points than one that intersected them, thus becoming less accurate for greater (unmeasured) control inputs. It was for this reason that the line was fit according to

\[ y = mx + b \]

with the $y$-intercept, $b$, dropped from the final model to maintain linearity.

3.3 Data Analysis

Figures 3.2 and 3.3 show the results of the system identification calculations. Both plots show an offset in the data from the linear relationship expected. As discussed previously, a first-degree polynomial was fit to this data and its slope was interpreted as $M$. It can be seen in both plots that the data matches rather closely the slope of $1/M$ but has a negative $y$-intercept. This might represent a reluctance in the arm to accelerate as predicted due to a fixed amount of static friction.

Since the addition of an intercept to the system model would invalidate the linearity of (3.1) and (3.2), this offset is ignored. Another solution might have forced the first-degree
Figure 3.2: Large arm identification results.
Figure 3.3: Small arm identification results.
polynomial fit to pass through zero, causing the line to pass through the center of the data points, where the slope would have presumably been far too shallow. While this may have made sense for the small range of command inputs used for system identification, it would not have been accurate for larger command inputs.
Chapter 4

Controller Design and Verification

4.1 Simulink Implementation

The controller for the system was implemented on a computer running MATLAB with Simulink. The hardware interaction was via Honeywell’s Real Time Toolkit for MATLAB. All of the simulation and experimentation was carried out using S-Function M-files used in Simulink.

The three S-Functions were controller.m, plant_model.m and plant_hardware.m. The controller and plants were connected in standard block diagram form as shown in Figure 4.1. The S-Function controller.m implemented (2.14) and (2.17) and integrated $f$ and $\tilde{F}_E$ (see Appendix A.6). The S-Function plant_model.m modeled the system dynamics by implementing (2.1) and (2.2) where $T_1$ and $T_2$ were input by the controller and $F_1$ and $F_2$ were added to $T_1$ and $T_2$ to simulate the environment as necessary (see Appendix A.7).

The S-Function plant_hardware.m replaced plant_model.m when the controller was to be implemented on the physical system (see Appendix A.8). Instead of integrating (2.1) and (2.2), it passed the control signals directly to the servo motors and returned the encoder readings to the controller.

4.2 Observer Design

Once an accurate system model was obtained, it was used to design the state observer. The design of the observer consisted of identifying the $2 \times 1$ vector $K_e$ that gave the observer the desired response characteristics.

This was accomplished by using the M-file observerdesign.m (see Appendix A.5). An observer was designed for the identified system with $\zeta = 0.6$ and $\omega_n = 5 \text{rad/s}$ as follows

\[ \gg \text{observerdesign}(0.0072,0.0016,5,0.6) \]

\[
J = \\
\begin{bmatrix}
-3.0000 + 4.0000i & 0 \\
0 & -3.0000 - 4.0000i
\end{bmatrix}
\]
Figure 4.1: Simulink simulation window.
Figure 4.2: Simulation frequency response with different speed observers.

\[ K_e = \begin{bmatrix} 0.0327 \\ 6.0000 \end{bmatrix} \]

where the J matrix represents the desired poles of the observer. This \( K_e \) is then used in the controller to implement (2.17).

### 4.3 Frequency Response

With the controller implemented, it was now possible to test the frequency response of the system. This was done using a chirp with magnitude 0.05 (5% of the control output) as input to one of the links. The resulting output of the slave link is shown in Figure 4.2. The top graph shows the response without an observer implemented (i.e. \( \hat{F}_E = 0 \)). The middle graph shows the response with an observer operating at 50 rad/s and the bottom shows the response with an observer operating at 100 rad/s.

It is clear that the addition of the observer to the controller reduces the error at lower
Figure 4.3: Simulation and hardware frequency response comparison.

frequencies. The trade-off appears to be in the slightly increased error around the 3–10 Hz range. This effect appears to reduce with increasing speed of the observer, however, and could be negligible for a sufficiently fast controller.

The next step was to compare the simulation response with the hardware performance. It was experimentally found that the fastest the observer could be run at and remain stable was 5 rad/s. This was significantly less than desirable for a human operated system, but would be sufficient to test the effectiveness of the observer.

For the identified inertias and damping of $\zeta = 0.6$, the 5 rad/s observer required

$$K_c = \begin{bmatrix} 0.0327 \\ 6 \end{bmatrix}$$

as specified by `observerdesign.m`. When implemented, this produced the slave manipulator output shown in Figure 4.3.

The simulation output shows a much smaller error reduction than those of Figure 4.2. The slow observer does, however, have an effect and this is reflected in the hardware implementation. The presence of the observer reduces the coordination error of the physical system.
Human interaction with the hardware also illustrates the presence of the observer. Forcing the manipulators to become uncoordinated (transferring energy into or out of the system) causes the controller to respond by increasing the control force until the error returns to zero (remains energetically passive). This effect is not present without the observer implemented.
Chapter 5

Conclusion

Simulation of the system implementing the force observer illustrates its effectiveness in reducing coordination error. For the periodic and relatively predictable chirp input, the force prediction appears to maintain the accuracy necessary for a significant error reduction. It must be remembered, however, that the probable motion of a teleoperator in human-environment interaction would not be as periodic or predictable.

The simulation results imply that faster observers produce better error attenuation. It was somewhat disappointing that the observer implemented on the physical system could only be run at 5 rad/s (roughly less than one update per second). This made the range of error reduction quite small and did not properly illustrate the potential usefulness of this technique on a real-world system.

The reasons for this limitation in the observer's speed could be attributed to the slow sample time of the controller—a side effect of using Simulink rather than just MATLAB for implementation. It could also have been the result of resonance effects from the anomaly in the force estimated responses that causes the output to increase slightly before decreasing with higher frequency.

From the results presented here, it appears as though observer-based force estimation could be applicable to systems whose inputs are relatively predictable. This would include applications like scraping, carving, etc., where a basic motion is repeated. More random motion may not be advisable. This is an area that may require future investigation.
Bibliography


Appendix A

Program Code

The following programs were written as MATLAB M-files.

A.1  acceltest.m

function data = acceltest(motor,command,time,trials)
% ACCELTEST  Captures motor's acceleration curve.
% DATA = ACCELTEST(MOTOR,COMMAND,TIME,TRIALS) returns a 100x3xTRIALS
% set of matrices containing the COMMAND in the first column, the elapsed
% time in the second column and the motor's position in the third column
% for output MOTOR.  One hundred data points are taken at even intervals
% from 0 to TIME seconds.  The third dimension is used to store multiple
% data sets, the number of which is determined by TRIALS.  DATA is
% processed using ACCELER.

rtclear; rturnload;

% Initialize hardware
Ts = 0.003;  % hardware sample time
rtload('ad512');
rtdef(motor,'out',Ts,motor);
rtstart all;
init;  % zero the encoders

% Record the position versus time
N = 100;  % number of data points
ts = time/N;  % data logging sample time
for i = 1:trials,  % Record each acceleration curve
fprintf('Press a key to start trial %d
', i)
pause
Posi = ReadPosi;
t0 = Posi(3);
rtwr(motor,command);
n = 0;
while(n=N)  % Record each point in the curve
n=n+1;
Posi = ReadPosi;
data(n,:,i) = [command Posi(3)-t0 Posi(motor)];
pause(ts);
end
rtwr(motor,0);
end

% Shutdown hardware
rtclear; rturnload;
zero;

% Review data for accuracy
figure(1)
cif
hold on
for i = 1:trials,
   plot(data(:,2,i),data(:,3,i),'k')
end
xlabel('Time (sec)')
ylabel('Arm Position (rad)')

A.2 accelcalc.m

function [T,a,error] = accelcalc(data)
\% ACCEL CALC Finds angular acceleration versus motor command.
\% [T,a,error] = ACCEL CALC(DATA) returns the vector of motor commands, T,
\% and the corresponding vector of angular accelerations, a, to be used to
\% find the arm's moment of inertia. The matrix DATA is 100x3xtrials where
\% column n contains the input command, column n+1 contains the elapsed time
\% and column n+2 contains the arm's position. The third dimension contains
\% the multiple trials of a given command n. Each set of three columns
\% corresponds to an ACCELTEST and results in an element in T, a and error.
\% Error is calculated using the t-statistic with 95% confidence.

\% Specify which points are to be polyfitted
b = 1;
e = 50;

tstat = 2.262; % alpha/2 = 0.025, nu = 9
[r c trials] = size(data);

% Fit a second-degree polynomial to each profile and use to find a
for n=1:3:c-2,
   for i = 1:trials,
      p = polyfit(data(b:e,n+1,i),data(b:e,n+2,i),2);
      atrial(i) = 2*p(1);
   end
   T(n+2)/3,:) = data(1,n);
a((n+2)/3,:) = mean(atrial);
   error((n+2)/3,:) = std(atrial)*tstat/sqrt(trials);
end
### A.3 arm1id.m

```matlab
load sysid;

arm1data = [arm1data1 arm1data2 arm1data3 arm1data4 arm1data5 arm1data6 ... arm1data7 arm1data8 arm1data9 arm1data10 arm1data11 arm1data12];

[T1,a1,error1] = accelcalc(arm1data);

% Calculate inertia
[p1,s1] = polyfit(a1(1:12),T1(1:12),1);
M1 = p1(1);

figure(1)
hold off
errorbar(T1, a1, error1, 'ko')
hold on
plot([0; T1], [0; T1]/M1, 'k-')
text(0.03,13,['M_1 = ' num2str(M1)])
legend('command/M_1',4)
axis([0 0.14 0 18])
grid on
xlabel('Command')
ylabel('Acceleration (rad/s^2)')
```

### A.4 arm2id.m

```matlab
load sysid;

arm2data = [arm2data1 arm2data2 arm2data3 arm2data4 arm2data5 arm2data6 ... arm2data7 arm2data8 arm2data9 arm2data10 arm2data11 arm2data12 ... arm2data13 arm2data14 arm2data15 arm2data16 arm2data17 arm2data18];

[T2,a2,error2] = accelcalc(arm2data);

% Calculate inertia
[p2, s2] = polyfit(a2(1:14), T2(1:14), 1);
M2 = p2(1);

figure(2)
hold off
errorbar(T2(1:14), a2(1:14), error2(1:14), 'ko')
hold on
plot([0; T2(1:14)] ,[0; T2(1:14)]/M2, 'k-')
text(0.015, 37.5 ,['M_2 = ' num2str(M2)])
legend('command/M_2',4)
axis([0 0.085 0 65])
grid on
xlabel('Command')
ylabel('Acceleration (rad/s^2)')
```
A.5 observerdesign.m

function observerdesign(M1,M2,wn,z)
%OBSERVERDESIGN An M-file for designing a state observer for the teleoperator
% system with \( A = [0 0; 1/Me 0] \) and \( C = [0 1] \).
% % OBSERVERDESIGN(M1,M2,wn,z) gives the vector \( Ke \) needed to implement the
% observer where \( M1 \) and \( M2 \) are the masses of arms 1 and 2, \( wn \) is the desired
% natural frequency in rad/s and \( z \) is the desired damping coefficient of the
% observer.
Me = M1*M2/(M1+M2);
A = [0 0; 1/Me 0];
C = [0 1];
N = [C' A'*C']; r = rank(N);
p = poly(A);
a1 = p(2); a2 = p(3);
J = [-wn*z+wn*sqrt(1-z^2)*i 0; 0 -wn*z-wn*sqrt(1-z^2)*i]
pp = poly(J);
aa1 = pp(2); aa2 = pp(3);
W = [a1 1; 1 0];
Ke = (inv(W*N'))*[aa2-a2; aa1-a1]

A.6 controller.m

function [sys,z0,str,ts] = controller(t,x,u,flag,a,r)
%CONTROLLER An M-file S-function for implementing passive control.
% % +-------------+
% % | q1-->|     |
% % | q2-->| Controller |-->T1
% % | q1'-->|     |--->T2
% % | q2'-->|     |
% % (u) +-------------+
% % The state variables are \( f \), the flywheel velocity; \( \text{Fest} \), the estimated
% environmental force; and the estimated Edot.
% Define variables
M1 = 0.0072;
M2 = 0.0016;
z = M2/M1;

% Coordinate transform from q1,q2 to E,L
S = [ z*a -z
      r*a   z ] / (r+z);
% Don't calculate parameters during initialization
if flag ~= 0

    % Calculate variables
    U = S*u(1:2,:);
    E = U(1);
    Udot = S*u(3:4,:);
    Edot = Udot(1);
    f = x(1);
    Fest = x(2);
    K = 2.5;
    b = 1.0;

    % Evaluate control forces
    fmin = 0.001;
    if f < fmin, f = fmin; end % avoid dividing by small numbers
    Ct = [0 K 0; -K 0 -(b*Edot+Fest)/f; 0 (b*Edot+Fest)/f 0];
    %Ct = [0 K 0; -K 0 -b*Edot/f; 0 b*Edot/f 0];
    %no Fest
    T = Ct*[E; Edot; f];
    Te = T(2);
    Tf = T(3);

end

switch flag,

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Initialization %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 0,
        [sys,x0,str,ts]=mdlInitializeSizes(S);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Derivatives %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 1,
        sys = mdlDerivatives(t,x,u,Tf,Te,M1,M2,Edot);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Outputs %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 3,
        sys=mdlOutputs(t,x,u,S,Te);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Unhandled flags %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case { 2, 4, 9 },
        sys = [];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Unexpected flags %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
e otherwise
        error(["Uncontrolled flag = ",num2str(flag)]);

end
end
% end controller

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [sys,x0,str,ts]=mdlInitializeSizes(S)

sizes = simsizes;
sizes.NumContStates = 3;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 5;
sizes.NumInputs = 4;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0 = [50; 0.1; 0.1]; % initial state variables must be nonzero
str = [];
str = [0 0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlDerivatives
% This is where the flywheel velocity is integrated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sys=mdlDerivatives(t,x,u,Tf,T1,M1,M2,Edot)

Mf = 1;
fdot = Tf/Mf;

% Implement state observer
Me = M1*M2/(M1+M2);
Ke = [0.0327;6]; %w=5rad/s
Ke = [3.2727;60]; %w=50rad/s
Ke = [13.0909; 120]; %w=100rad/s
A = [0 0; 1/Me 0];
B = [0; 1/Me];
C = [0 1];
xdot = A*[x(2);x(3)] + B*Te + Ke*(Edot-C*[x(2);x(3)]);

sys = [fdot xdot(1) xdot(2)];

%end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlOutputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Return the control forces T1 and T2
%============================================================================
% function sys=mdlOutputs(t,x,u,S,Te)

sys = [S*[Te;0]; x];
% end mdlOutputs
%
%============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%============================================================================
% function sys=mdlTerminate(t,x,u)

sys = [];
% end mdlTerminate

A.7 plant_model.m

function [sys,q0,str,ts] = plant_model(t,q,u,flag)
%PLANT_MODEL An M-file S-function modeling two one-DOF teleoperators.
%
% +--------+
% | T1-->| |-->q1
% | T2-->| Plant |-->q2
% | F1-->| Model |-->q1'
% | F2-->| |-->q2'
% (u) +--------+
%
% Implements the equation:
% q' = Aq + Bu

M1 = 0.0072;
M2 = 0.0016;

A = [ 0 0 1 0;
      0 0 0 1;
      0 0 0 0;
      0 0 0 0 ];

B = [ 0 0;
      0 0;
      1/M1 0;
      0 1/M2 ];

switch flag,
    % Initialization %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 0,
[sys,q0,str,ts]=mdlInitializeSizes(A,B);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Derivatives %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case 1, 
sys=mdlDerivatives(t,q,u,A,B);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outputs %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case 3, 
sys=mdlOutputs(t,q,u,A,B);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Unhandled flags %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case { 2, 4, 9 }, 
sys = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Unexpected flags %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% end plant_model

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [sys,q0,str,ts]=mdlInitializeSizes(A,B)

sizes = simsizes;
sizes.NumContStates = 4;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 4;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
q0 = zeros(4,1);
str = [];
ts = [0 0];

% end mdlInitializeSizes
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% mdlDerivatives
% Return the derivatives for the continuous states.
%function sys=mdlDerivatives(t,q,u,A,B)
sys = A*q + B*u;
% end mdlDerivatives

%function sys=mdlOutputs(t,q,u,A,B)
sys = q;
% end mdlOutputs

%function sys=mdlTerminate(t,q,u)
sys = [];
% end mdlTerminate

A.8 plant_hardware.m

function [sys,q0,str,ts] = plant_hardware(t,q,u,flag)
%PLANT_HARDWARE an M-file S-function operating two one-DOF teleoperators.
% +--------+
% |        |
% T1-->| Plant | -->q1
% T2-->| Hardware| -->q1'
% (u) |       | -->q2'
% +--------+

switch flag,

case 0,
[sys,q0,str,ts]=mdlInitializeSizes(t);
% Update

% Outputs

case 3,
    sys = mdlOutputs(t,q,u);

% Unhandled flags

case { 2, 4, 9 },
    sys = [];

% Unexpected flags

otherwise
    error(["Unhandled flag = ",num2str(flag)]);
end

% mdlInitializeSizes

% Return the sizes, initial conditions, and sample times for the S-function.

function [sys,q0,str,ts]=mdlInitializeSizes(t)

sizes = simsizes;
sizes.NumContStates = 0;
sizes.NumDiscStates = 5;
sizes.NumOutputs = 4;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
Posi = ReadPosi;

sys = simsizes(sizes);
q0 = [zeros(4,1); Posi(3)];
str = [];
ts = [0 0];
rtclear;
rtunload;
init;

Ts = 0.003;
rtload('ad512');
rtdef(1,'out',Ts,1); % Timer 1 output command to DMC 107, Motor 415 (big)
rtdef(2,'out',Ts,2); % Timer 2 output command to DMC 109, Motor 416 (small)
rtstart all;
rtwr(1,0);
rtwr(2,0);

% end mdlInitializeSizes
%
%============================================================================
% mdlUpdate
% Send commands to motor and read the encoders to update states.
%============================================================================
% function sys=mdlUpdate(t,q,u)

% Calculate and send commands to motor
rtwr(1,u(1));
rtwr(2,u(2));

% Read arm positions and time, calculate velocity to complete q = [q1; q2; q1'; q2'; t]  
Posi = ReadPosi;
sys = [Posi(1); Posi(2); 0; 0; Posi(3)];
sys(3:4) = (Posi(1:2)-q(1:2))/(Posi(3)-q(5));

% end mdlUpdate
%
%============================================================================
% mdlOutputs
% Return first four state variables: q1, q2, q1' and q2'
%============================================================================
% function sys=mdlOutputs(t,q,u)

% Return arm positions and velocities
sys = q(1:4);

% end mdlOutputs
%
%============================================================================
% mdlTerminate
% Shutdown hardware.
%============================================================================
% function sys=mdlTerminate(t,q,u)

rtwr(1,0);
rtwr(2,0);
rtcLEAR;
rtunload;
sys = [];

% end mdlTerminate