Final Project: Implementing Adaptive RLS for Payload Agnostic Input Shaping

ME 6404: Advanced Control System Design and Implementation

Andrew Hubbard

Jerry Lu

Ulagarjun Ulaga Narasimhan

Yadu Sunil

Dr. William Singhose

Tyler Rome

December 2, 2024

**Introduction**

Crane operations require effective suppression of payload oscillations to ensure safety and efficiency. Traditional input shapers rely on fixed system parameters, failing to adapt to varying payloads and cable lengths, leading to oscillations and manual recalibration. Such limitations can compromise the precision, safety, and efficiency of crane operations. To address these limitations, this project applies the Adaptive Recursive Least Squares (RLS) method to a tower crane's double-pendulum system. The RLS dynamically adjusts input shaping parameters in real-time, ensuring stability and minimal vibrations across varying payloads and cable lengths, eliminating the need for recalibration. The results indicate that RLS effectively attenuates oscillations for payloads lighter than the hook mass but becomes less effective as payload mass exceeds the hook mass.

**Methodology**

*Derivation of Equations of Motion*

We defined the state space of the pendulum using the trolley position , the angle of the upper pendulum from vertical, and the angle from vertical of the lower pendulum . The physical parameters of the system are the lengths (upper, lower) of the pendulums, and the masses and of the hook and payload. The hook and payload were modeled as point masses, with massless cords. Additionally, we modeled the damping of the system as viscous air resistance on each of the masses, according to . We derived the kinematics of the system, and applied Newton’s law to derive equations of motion (1) and (2). (1) is Newton’s law applied to the payload, perpendicular to the lower cord, and (2) is Newton’s law applied to both hook and payload, perpendicular to the upper cord.

|  |  |
| --- | --- |
|  | (1)  (2) |

*Derivation of Recursive Least Squares Update Equation*

To simplify the equations of motion into a form suitable for estimation via RLS, we applied the small-angle approximation. We neglected the damping terms of the system and neglected the centrifugal stiffness terms and . This results in the simplified equations of motion (3) and (4), where.

|  |  |
| --- | --- |
|  | (3)  (4) |

Since the tower crane can sense the position of the hook but not the payload, it is necessary to formulate the equations of motion in terms of only and . Thus, we solved the system of equations for and substituted back, resulting in (5).

|  |  |
| --- | --- |
|  | (5) |

For the RLS update law, we modeled the system as a generic linear system of the same form as (5), shown in (6). This allowed us to formulate deviation from the equations of motion in terms of a parameter vector and regression vector in the standard form for RLS, , as shown in (7).

|  |  |
| --- | --- |
|  | (6) |
|  | (7) |

The RLS update equation is standard, using and as shown above.

*Adaptive Input Shaper*

To apply the estimation from RLS to an input shaper, we calculate the estimated natural frequencies of the system using (8). The period of oscillation for each mode is then determined from the frequencies.

|  |  |
| --- | --- |
|  | (8) |

We used a two-mode convolved ZV shaper for this project. While more capable shapers exist, this choice was made to highlight the robustness of frequency adaptation, rather than the robustness of a simple shaper.

*Simulation Implementation*

The inputs used to perform RLS are the position of the hook, and the velocity of the cart . To derive the several derivatives necessary, we used digital filters on the measured values. These filters are shown below. The filter was chosen iteratively to have no zeros and poles at 0.7, 0.7, 0.6, and 0.5. The filter is a centered, discrete th order derivative, with an accuracy of . The delay makes the derivative filters causal.

A math equations on a white background

Description automatically generatedA math equations on a white background

Description automatically generated A math equations on a white background

Description automatically generated

A math equations on a white background

Description automatically generated A math equations on a white background

Description automatically generated

The result is that the resulting filtered values are all effectively filtered from their true values by . Since the filter for each input to RLS is identical, the RLS update equation still works by minimizing the filtered equation-of-motion error. These filters were implemented by keeping a buffer of the most recent values of their inputs and outputs, and taking a weighted sum as determined by the difference equation form of the filters.

For the RLS itself, we specified the initial gain matrix to be a multiple of the identity. We did not use a forgetting factor, because in testing we found that this causes the gain matrix to be unstable when is zero.

The input shaper was implemented by keeping a history of inputs to the system. Each impulse of the shaper then looks back by the specified delay, and the sum of these produces the shaped input for the system.

*Hardware Validation*

The hardware testing procedure begins by initializing the RLS algorithm with known values for hoist length, hook mass, and initial estimates for payload length and mass. The crane is then moved back and forth along the jib, allowing the RLS algorithm to dynamically adapt and learn payload parameters, simulating real-world manual operation. To mimic practical variability, no predefined trajectory was used.

Using the frequencies estimated by RLS, a shaped trajectory is designed with a two-mode ZV shaper to minimize oscillations. The residual oscillations are measured at the end of the trajectory to assess the effectiveness of the RLS-predicted shaper. These results are compared to an unshaped trajectory and another shaped trajectory based on frequencies calculated using the following theoretical equation.

**Results**

*Effect of Initial Guesses on RLS-Predicted Frequencies*

The RLS algorithm applied in the experiments has no forgetting factor. This means that all collected data points, including the initial guess of parameters, have equal rather than diminishing weight. As such, it is important to explore how initial guesses impact the algorithm’s performance.

The experiment was run using a hook of mass 0.21 kg at a hoist length of 0.7 m. The payload had a mass of 0.078 kg, suspended from the hook at a length of 0.7 m. The initial and converged guesses for the parameters are shown in Table 1. Figure 1 displays the convergence of several initial guesses of different payload mass and length combinations. The lower mode frequency estimates vary by up to 84% from the theoretical value of 3.10 rad/s. However, the higher mode frequency exceeds the theoretical value of 5.10 rad/s across all tests.

These discrepancies could arise due to the simplifying assumptions used in both the theoretical and the RLS update equation. The theoretical equation assumes idealized conditions, such as perfect small-angle approximations, negligible damping, and point-mass payloads, which may not fully capture the complexities of the real system. On the other hand, the RLS algorithm dynamically updates frequency estimates based on real-time motion data but is sensitive to noise, inaccuracies in initial parameter guesses, and the approximations used in its adaptive update rule. Table 1 summarizes the converged frequency estimates and their initial guesses. Test 5, which uses the exact payload parameters for the initial guess, gives the best estimate for at 3.18 rad/s, while the estimate of 8.51 rad/s shows no significant improvement over other tests. This suggests that accurate initial guesses can lead to better algorithm performance. More tests are needed to more precisely describe this behavior; these tests should incrementally manipulate one initial guess parameter and keep the other as a control.

A graph of different colored lines

Description automatically generated*Figure 1: Frequency estimates for different initial guesses*

*Table 1: Frequency estimates for different initial guesses – numeric values*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Parameters** | **Theory** | **Test 1** | **Test 2** | **Test 3** | **Test 4** | **Test 5** | **Avg** |
| (m) | 0.7 | 0.5 | 0.1 | 0.1 | 0.8 | 0.7 | -- |
| (kg) | 0.078 | 0.5 | 0.1 | 0.5 | 0.01 | 0.078 | -- |
| (rad/s) | 3.10 | 3.95 | 2.42 | 2.68 | 5.77 | 3.18 | 3.6 |
| (rad/s) | 5.10 | 9.11 | 7.86 | 8.04 | 10.91 | 8.51 | 8.89 |

To assess the degree to which physical implementation errors affected these results, an identical test was run using the simulation. The convergence of the estimated natural frequencies is shown in Figure 2.



*Figure 2: Frequency Estimates for Different Initial Guesses in Simulation*

In each case, the simulation quickly identifies the higher mode as 13.23 rad/s. However, the lower mode is not consistently identified as any particular value. This indicates that the system is dominated by a single mode for this payload, which is a likely cause of inconsistency in the physical experiments.

The consistency of the RLS algorithm for a given initial guess was also tested on the physical system. The results are displayed in Figure 3. Using the same initial guess for all 3 tests, estimates for both frequencies converged to very similar values for Tests 1 and 2, but Test 3 resulted in higher values.



*Figure 3: Frequency estimates for identical initial guesses*

Single-mode domination is the most likely reason for the convergence to behave inconsistently. If only one frequency has a significant effect on the system behavior, it follows that trying to find two frequencies which best match the behavior may have several valid solutions.

Another potential cause of this inconsistency is the lack of forgetting factor, where the oldest data may affect the resulting converged values. This is a detriment when old data contains outliers such as initialization errors or accidental disturbances to the system. Some of the variance in results obtained in Table 1 may be attributed to this effect. Two methods expected to decrease this inconsistency are to implement a forgetting factor or allow the system to spend more time collecting oscillation data to offset the weight of the old data. More tests are needed to more precisely describe inconsistency in the algorithm’s performance.

*Effect of Payload Mass on RLS Performance*

The performance of the RLS algorithm on the physical system depends heavily on the payload mass. Figure 4 shows that RLS effectively attenuates oscillations when the payload mass is smaller than the 0.21 kg hook mass. However, as the payload mass approaches or exceeds the hook mass, RLS performance declines, with frequency predictions deviating more from theoretical values.

A graph of a graph of a person

Description automatically generated with medium confidence

*Figure 4: Maximum residual radial deflection for four different payload masses across unshaped, theoretically shaped, and RLS-shaped trajectories*

This behavior can be attributed to the system dynamics transitioning as the payload mass increases. For lighter payloads, the system exhibits double-pendulum-like behavior, where both the hook and payload dynamics are prominent. Under these conditions, RLS effectively identifies the two natural frequencies of the system, (lower mode) and (higher mode), enabling precise oscillation attenuation. However, as the payload mass increases, its influence dominates the dynamics, causing the system to behave more like a single pendulum. This shift reduces the significance of the hook dynamics, making it more challenging for RLS to accurately estimate ω₁, which is the slower, dominant mode frequency. This trend is seen in Figure 4, as the from theory and RLS diverge as mass increases.

Additionally, RLS relies on feedback from the crane's motion to adapt its estimates. As the payload mass increases, the oscillation amplitudes become more pronounced, which may introduce non-linear effects not accounted for by the RLS update equation. This can exacerbate the deviation of RLS-predicted frequencies from theoretical values, particularly for the lower mode .

Finally, Figure 4 also clearly shows that neither the theoretical nor the RLS-based shapers completely attenuate oscillations, which means that both approaches deviate from the true system dynamics to some extent. By redesigning the RLS update equation to account for more realistic system behaviors such as incorporating damping, non-linear coupling between modes, or the effects of cable flexibility, it may be possible to achieve better frequency estimates and, consequently, more effective oscillation attenuation.

For comparison, this experiment was also run on the simulated pendulum. The results of this are shown in Figure 5. Note that while the while the theoretical input shaper does significantly reduce the oscillation compared to the unshaped input, especially when the system is naturally attenuated by the preprogrammed input for a given payload mass, the RLS shaper performs extremely well regardless of the payload.

**

*Figure 5: Maximum residual oscillation for different payload masses for unshaped, theoretically shaped, and RLS shaped trajectories*

The difference in performance of the RLS shaper across the several masses is surprising. While it works very well in simulations, it does not perform well in physical testing. This could be caused by a few differences between the physical and simulated system, such as input delays between the hook and trolley and quantization of the shaper input times, among others.

*Effect of Payload Cable Length on RLS Performance*

Figure 6 shows the maximum residual radial deflection for three cable lengths (0.45m, 0.65m, 0.85m) at a 0.19 kg payload mass, and two cable lengths (0.45m, 0.85m) at a 0.078 kg payload mass. Using a two-mode ZV input-shaper, RLS-predicted frequencies are compared to theoretical frequencies.

At a payload mass of 0.19 kg, the theoretical shaper generally outperforms the RLS shaper, as the system approaches single-pendulum dynamics where fixed theoretical parameters align better. However, at 0.85m, the RLS shaper surpasses the theoretical shaper due to its ability to adapt to parameter variations caused by the increased sensitivity of longer cables.

For a 0.078 kg payload, RLS consistently outperforms the theoretical shaper for both 0.45m and 0.85m cable lengths. The smaller payload mass, compared to the hook mass, makes the system highly sensitive to parameter changes, reducing the theoretical shaper's effectiveness. The RLS method dynamically adjusts to these variations, allowing for superior oscillation suppression.

A graph of different colored lines

Description automatically generated with medium confidence

*Figure 6: Maximum residual radial deflection for different payload masses and cable lengths across unshaped, theoretically shaped, and RLS-shaped trajectories*

**Conclusions**

The implementation of the RLS algorithm successfully attenuated payload oscillations, particularly when the payload mass was less than the hook mass. However, several limitations hindered its full potential. Simplifying assumptions in the update equation facilitated implementation but likely impacted prediction accuracy. A more rigorous formulation could improve estimates. Additionally, the physical system introduced unexpected factors, such as hook tilt and non-constant timesteps, which created challenges for the RLS update process. Further improvements could include optimizing trolley movement patterns to enhance RLS learning and validating frequency predictions through FFT analysis. To increase practicality, incorporating a forgetting factor could significantly reduce learning time. Finally, evaluating RLS with more robust shaping methods would provide valuable insights into its broader applicability. These further advancements could establish RLS-based adaptive shaping as a viable solution for real-world crane operations.

**Appendix**

**Final Project Code**

***RLS class for tower crane Arduino:***

#ifndef RLS\_HPP

#define RLS\_HPP

// #include <string.h> // Needed for memcpy

#include <cmath>

class RLS {

private:

float m1 = 0.23;

float T = 0.016667; //60 Hz

float L1;

float lambda = 1;

float g = 9.81;

float THETA[4];

float P[4][4];

float t1\_buffer[7] = {0};

float v\_buffer[7] = {0};

float Af[5] = {0, 0, 0, 0, 1}; // Numerator coefficients

float Bf[5] = {1, -2.5, 2.33, -0.959, 0.147}; // Denominator coefficients

int filter\_order = 5;

// Buffers for filtered data

float t1f\_buffer[5] = {0}; // Filtered angle buffer

float ddt1f\_buffer[5] = {0}; // Filtered first derivative

float ddddt1f\_buffer[5] = {0}; // Filtered second derivative

float ddxf\_buffer[5] = {0}; // Filtered velocity buffer

float ddddxf\_buffer[5] = {0}; // Filtered acceleration buffer

float t1f = 0.0; // Filtered angle

float ddt1f = 0.0; // Filtered first derivative

float ddddt1f = 0.0; // Filtered fourth derivative

float ddxf = 0.0; // Filtered velocity

float ddddxf = 0.0; // Filtered acceleration

float applyDotProduct(const float\* a, const float\* b, int length) {

float result = 0.0;

for (int i = 0; i < length; i++) {

result += a[i] \* b[i];

}

return result;

}

void filterInputs(float t1, float v) {

for (int i = 6; i > 0; i--) {

t1\_buffer[i] = t1\_buffer[i - 1];

v\_buffer[i] = v\_buffer[i - 1];

}

t1\_buffer[0] = t1;

v\_buffer[0] = v;

// Shift buffers and calculate the new value

for (int i = filter\_order - 1; i > 0; i--) {

t1f\_buffer[i] = t1f\_buffer[i - 1];

ddt1f\_buffer[i] = ddt1f\_buffer[i - 1];

ddddt1f\_buffer[i] = ddddt1f\_buffer[i - 1];

ddxf\_buffer[i] = ddxf\_buffer[i - 1];

ddddxf\_buffer[i] = ddddxf\_buffer[i - 1];

}

float Ad\_t1[] = {0, 0, 0, 1, 0, 0, 0};

float Cd\_t1 = 1;

t1f\_buffer[0] = (applyDotProduct(Ad\_t1, t1\_buffer, 7) \* Cd\_t1 -

applyDotProduct(&Bf[1], &t1f\_buffer[1], filter\_order - 1)) /

Bf[0];

float Ad\_ddt1[] = {1.0 / 90, -3.0 / 20, 3.0 / 2, -49.0 / 18, 3.0 / 2, -3.0 / 20, 1.0 / 90};

float Cd\_ddt1 = 1 / (T \* T);

ddt1f\_buffer[0] = (applyDotProduct(Ad\_ddt1, t1\_buffer, 7) \* Cd\_ddt1 -

applyDotProduct(&Bf[1], &ddt1f\_buffer[1], filter\_order - 1)) /

Bf[0];

float Ad\_ddddt1[] = {-1.0 / 6, 2.0, -13.0 / 2, 28.0 / 3, -13.0 / 2, 2.0, -1.0 / 6};

float Cd\_ddddt1 = 1 / (T \* T \* T \* T);

ddddt1f\_buffer[0] = (applyDotProduct(Ad\_ddddt1, t1\_buffer, 7) \* Cd\_ddddt1 -

applyDotProduct(&Bf[1], &ddddt1f\_buffer[1], filter\_order - 1)) /

Bf[0];

float Ad\_ddxf[] = {-1.0 / 60, 3.0 / 20, -3.0 / 4, 0.0, 3.0 / 4, -3.0 / 20, 1.0 / 60};

float Cd\_ddxf = 1 / T;

ddxf\_buffer[0] = (applyDotProduct(Ad\_ddxf, v\_buffer, 7) \* Cd\_ddxf -

applyDotProduct(&Bf[1], &ddxf\_buffer[1], filter\_order - 1)) /

Bf[0];

float Ad\_ddddxf[] = {1.0 / 8, -1.0, 13.0 / 8, 0.0, -13.0 / 8, 1.0, -1.0 / 8};

float Cd\_ddddxf = 1 / (T \* T \* T);

ddddxf\_buffer[0] = (applyDotProduct(Ad\_ddddxf, v\_buffer, 7) \* Cd\_ddddxf -

applyDotProduct(&Bf[1], &ddddxf\_buffer[1], filter\_order - 1)) /

Bf[0];

// Compute filtered outputs (t1f, ddt1f, etc.)

t1f = applyDotProduct(Af, t1f\_buffer, filter\_order);

ddt1f = applyDotProduct(Af, ddt1f\_buffer, filter\_order);

ddddt1f = applyDotProduct(Af, ddddt1f\_buffer, filter\_order);

ddxf = applyDotProduct(Af, ddxf\_buffer, filter\_order);

ddddxf = applyDotProduct(Af, ddddxf\_buffer, filter\_order);

}

void solveRLS() {

float PHI[4] = {-ddt1f, -t1f, ddddxf, ddxf};

float e = ddddt1f;

for (int i = 0; i < 4; i++) { //e = y-THETA'\*PHI

e -= THETA[i] \* PHI[i];

}

float PHI\_P[4] = {0};

for (int i = 0; i < 4; i++) { //P\*PHI

for (int j = 0; j < 4; j++) {

PHI\_P[i] += P[i][j] \* PHI[j];

}

}

float denominator = 1;

for (int i = 0; i < 4; i++) { //1+PHI'\*P\*PHI

denominator += PHI[i] \* PHI\_P[i];

}

for (int i = 0; i < 4; i++) { //THETA+P\*PHI\*e/(1+PHI'\*P\*PHI)

THETA[i] += PHI\_P[i] \* e / denominator;

}

float P\_new[4][4];

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

P\_new[i][j] = P[i][j] - PHI\_P[i] \* PHI\_P[j] / denominator;

P\_new[i][j] /= lambda;

}

}

// memcpy(P, P\_new, sizeof(P)); //copies P\_new to P

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

P[i][j] = P\_new[i][j];

}

}

}

public:

RLS(float L1, float L2\_est, float m2\_est) : L1(L1) { //cart velocity and hook deflection

THETA[0] = (L1 + L2\_est) \* g \* (m1 + m2\_est) / (L1 \* L2\_est \* m1);

THETA[1] = (g \* g \* (m1 + m2\_est)) / (L1 \* L2\_est \* m1);

THETA[2] = -1 / L1;

THETA[3] = g \* (m1 + m2\_est) / (L1 \* L2\_est \* m1); // init theta

for (int i = 0; i < 4; i++) { //init P as Identity

for (int j = 0; j < 4; j++) {

P[i][j] = (i == j) ? 1000 : 0;

}

}

}

void update(float t1, float v) {

t1 = (t1 - 13.5)/1000;

t1 = atan(t1/L1);

v = v/1000;

// Serial.print("t1: ");

// Serial.println(t1);

// Serial.print("v: ");

// Serial.println(v);

filterInputs(-t1, v);

solveRLS();

// printTheta();

printFreq();

}

float\* getTheta() {

return THETA;

}

float\* getPRow(int row) {

return P[row];

}

void printTheta() {

Serial.print("Theta: ");

for (int i = 0; i < 4; i++) {

Serial.print(THETA[i]);

Serial.print(" ");

}

Serial.println();

}

void printP(){

Serial.println("Matrix P:");

for(int i = 0; i<4; i++) {

for (int j = 0; j < 4; j++){

Serial.print(P[i][j]);

Serial.print(" ");

}

Serial.println();

}

}

void printFreq(){

float term = sqrt(fabs(THETA[0] - 4\*THETA[1]));

float wn1 = sqrt(fabs(THETA[0] - term) / 2);

float wn2 = sqrt(fabs(THETA[0] + term) / 2);

// Serial.println("Natural Frequencies");

Serial.print("1");

Serial.println(wn1);

Serial.print("2");

Serial.println(wn2);

}

};

#endif

***Modified PENDULUM.m file to create double pendulum***

classdef PENDULUM < handle

properties(GetAccess = 'private', SetAccess = 'private')

m\_Haxle %handle

m\_Hbase

m\_Hmass

m\_Hrod

m\_Hrodbase

m\_Hfig

m\_Haxis

m\_Vaxle

m\_Vbase

m\_Vmass

m\_Vrod

m\_Vrodbase

m\_WS %x length

m\_Mode %Velocity or Force Control

m\_Mp %Mass of pendulum

m\_Mc %Mass of cart

m\_L %Length of pendulum

m\_b %Viscous damping coef.

m\_Hrod2 % Handle for the second rod

m\_Hmass2 % Handle for the second mass

m\_Vrod2 % Vertices for the second rod

m\_Vmass2 % Vertices for the second mass

m\_L2 % Length of the second pendulum

m\_Mp2 % Mass of the second pendulum

m\_b2

m\_Q %q1 = theta, q2 = theta\_dot, q3 = x, q4 = xdot

end

properties(GetAccess = 'public', SetAccess = 'public')

U = 0 ; %control input.

end

properties(GetAccess = 'public', SetAccess = 'private')

Q = 0;

L = 0;

Mp = 0;

Mc = 0;

b = 0;

L2 = 0; % Length of the second pendulum

Mp2 = 0; % Mass of the second pendulum

b2 = 0;

end

methods

function set.U(obj,val)

obj.U = val;

obj.UPDATE\_OTHER\_DATA;

end

function Q = get.Q(obj)

Q = obj.m\_Q;

end

function L = get.L(obj)

L = obj.m\_L;

end

function Mp = get.Mp(obj)

Mp = obj.m\_Mp;

end

function Mc = get.Mc(obj)

Mc = obj.m\_Mc;

end

function b = get.b(obj)

b = obj.m\_b;

end

function L2 = get.L2(obj)

L2 = obj.m\_L2;

end

function Mp2 = get.Mp2(obj)

Mp2 = obj.m\_Mp2;

end

function b2 = get.b2(obj)

b2 = obj.m\_b2;

end

end

methods(Access = 'public')

function obj = PENDULUM(varargin)

%default

obj.m\_WS = 3;

obj.m\_Q = [0 0 0 0 0 0];

obj.m\_Hfig = gcf;

obj.m\_Haxis = gca;

obj.m\_Mp = 1;

obj.m\_Mc = 1;

obj.m\_L = 1;

obj.m\_b = 1;

obj.m\_L2 = 1; % Length of the second pendulum

obj.m\_Mp2 = 1; % Mass of the second pendulum

obj.m\_b2 = 1;

obj.m\_Mode = 'Force'; %'Velocity'

%specified

if ~isempty(varargin)

for k = 1:2:length(varargin)

tmp\_str = varargin{k};

switch tmp\_str

case 'Mode'

obj.m\_Mode = varargin{k+1};

case 'Damping'

obj.m\_b = varargin{k+1};

case 'MassCart'

obj.m\_Mc = varargin{k+1};

case 'MassPendulum'

obj.m\_Mp = varargin{k+1};

case 'PendulumLength'

obj.m\_L = varargin{k+1};

case 'SecondPendulumLength' % New case for second pendulum length

obj.m\_L2 = varargin{k+1};

case 'SecondPendulumMass' % New case for second pendulum mass

obj.m\_Mp2 = varargin{k+1};

case 'SecondPendulumDamping' % New case for second pendulum damping

obj.m\_b2 = varargin{k+1};

case 'WorkspaceLength'

obj.m\_WS = varargin{k+1};

case 'InitialStates'

obj.m\_Q = varargin{k+1};

obj.m\_Q = obj.m\_Q(:);

case 'Fig'

obj.m\_Hfig = varargin{k+1};

case 'Axis'

obj.m\_Haxis = varargin{k+1};

otherwise

errordlg([tmp\_str,' is not a valid parameter name.'],'Syntax Error');

return

end

end

end

%initialize robot

obj.INITIALIZE

end

end

methods(Access = 'private')

function INITIALIZE(obj)

%COLORS

c\_runway = [162 20 20 ]/255;

c\_base = [190 122 66 ]/255;

c\_axle = [142 152 187]/255;

c\_rodbase = [190 122 66 ]/255;

c\_rod = [180 180 130]/255;

c\_mass = [100 100 100]/255;

c\_floor = [100 100 100]/255;

%INITIALIZE THE FIGURE

figure(obj.m\_Hfig)

axes(obj.m\_Haxis)

cla

set(gcf,'RendererMode','manual','Renderer','OpenGL')

%GET THE PARTS

% cd 'Pend\_Model'

% [F\_base, V\_base, ~] = STL2MAT('base.stl',1);

% [F\_axle, V\_axle, ~] = STL2MAT('axle.stl',1);

% [F\_rodbase, V\_rodbase, ~] = STL2MAT('rodbase.stl',1);

% [F\_rod, V\_rod, ~] = STL2MAT('rod.stl',1);

% [F\_mass, V\_mass, ~] = STL2MAT('mass.stl',1);

% [F\_runway, V\_runway, ~] = STL2MAT('runway.stl',1);

% [F\_floor, V\_floor, ~] = STL2MAT('floor.stl',1);

% cd ..

%

% save PARTS F\_base V\_base F\_axle V\_axle F\_rodbase V\_rodbase F\_rod ...

% V\_rod F\_mass V\_mass F\_runway V\_runway F\_floor V\_floor

%

load PARTS2

%DRAW THE PARTS AS PATCH OBJECTS WITH HANDLES

WS = obj.m\_WS;

del = 0.09;

%runway

V\_runway(:,1) = V\_runway(:,1)\*WS;

V\_runway(:,3) = V\_runway(:,3) - del;

patch('faces', F\_runway, 'Vertices' ,V\_runway,'EdgeColor','none','FaceC',c\_runway,...

'AmbientStrength',.6,'DiffuseStrength',.9,'SpecularStrength',.3,'SpecularExponent',30);

%base

V\_base(:,3) = V\_base(:,3) - del;

obj.m\_Hbase = patch('faces',F\_base, 'Vertices' ,V\_base,'EdgeColor','none','FaceC',c\_base,...

'AmbientStrength',.3,'DiffuseStrength',.6,'SpecularStrength',.8,'SpecularExponent',10);

obj.m\_Vbase = V\_base;

%axle

V\_axle(:,3) = V\_axle(:,3) - del;

obj.m\_Haxle = patch( 'faces',F\_axle, 'Vertices' ,V\_axle,'EdgeColor','none','FaceC',c\_axle,...

'AmbientStrength',.3,'DiffuseStrength',.6,'SpecularStrength',.8,'SpecularExponent',10);

obj.m\_Vaxle = V\_axle;

%rodbase

V\_rodbase(:,3) = V\_rodbase(:,3) - del;

obj.m\_Hrodbase = patch( 'faces',F\_rodbase, 'Vertices' ,V\_rodbase,'EdgeColor','none','FaceC',c\_rodbase,...

'AmbientStrength',.3,'DiffuseStrength',.6,'SpecularStrength',.8,'SpecularExponent',10);

obj.m\_Vrodbase = V\_rodbase;

%rod

V\_rod(:,3) = V\_rod(:,3) \* obj.m\_L;

obj.m\_Hrod = patch( 'faces',F\_rod, 'Vertices' ,V\_rod,'EdgeColor','none','FaceC',c\_rod,...

'AmbientStrength',.3,'DiffuseStrength',.6,'SpecularStrength',.8,'SpecularExponent',10);

obj.m\_Vrod = V\_rod;

%mass

V\_mass(:,3) = V\_mass2(:,3) + obj.m\_L;

obj.m\_Hmass = patch( 'faces',F\_mass, 'Vertices' ,V\_mass,'EdgeColor','none','FaceC',c\_mass,...

'AmbientStrength',.3,'DiffuseStrength',.6,'SpecularStrength',.8,'SpecularExponent',10);

obj.m\_Vmass = V\_mass;

% Second Rod

V\_rod2(:,3) = V\_rod2(:,3) \* obj.m\_L2; % Scale the rod vertices by the length of the second pendulum

obj.m\_Hrod2 = patch('faces', F\_rod, 'Vertices', V\_rod2, 'EdgeColor', 'none', 'FaceC', c\_rod, ...

'AmbientStrength', .3, 'DiffuseStrength', .6, 'SpecularStrength', .8, 'SpecularExponent', 10);

obj.m\_Vrod2 = V\_rod2;

% Second Mass

V\_mass2(:,3) = V\_mass2(:,3) + obj.m\_L2; % Position the mass at the end of the second rod

obj.m\_Hmass2 = patch('faces', F\_mass, 'Vertices', V\_mass2, 'EdgeColor', 'none', 'FaceC', c\_mass, ...

'AmbientStrength', .3, 'DiffuseStrength', .6, 'SpecularStrength', .8, 'SpecularExponent', 10);

obj.m\_Vmass2 = V\_mass2;

%floor

V\_floor(:,1:2) = V\_floor(:,1:2)\*0.5/0.2;

V\_floor(:,3) = V\_floor(:,3) - 3;

for i = 1:round(WS/.500)

for j = 1:round(1/.500)

floor(i,j) = patch('faces', F\_floor, 'Vertices' ,V\_floor,'EdgeColor','none',...

'FaceC',c\_floor,'FaceAlpha',1,'Visible','off','AmbientStrength',.5,...

'DiffuseStrength',.3,'SpecularStrength',.1,'SpecularExponent',50);

tmp = get(floor(i,j),'Vertices');

tmp(:,1) = tmp(:,1) + (i-1)\*.500 + -WS/2;

tmp(:,2) = tmp(:,2) + (j-1)\*.500 + -0.5;

set(floor(i,j),'Vertices',tmp,'Visible','on')

end

end

%MAKE AXIS LOOK NICE

axis equal

axis([-obj.m\_WS/2-0.1 obj.m\_WS/2+0.1 -0.6 0.6 -3 1])

light

lighting gouraud

%perspective

camproj('perspective')

view(20,32)

end

function UPDATE\_OTHER\_DATA(obj)

persistent THEN u\_old

if isempty(THEN)

THEN = obj.GETTIME;

u\_old = 0;

end

%KEEP TRACK OF TIME

now = obj.GETTIME;

del\_t = now - THEN;

THEN = now;

%CALCULATE THE NEW STATES

T = [0 del\_t];

Q = obj.m\_Q;

L = obj.m\_L;

Mp = obj.m\_Mp;

Mc = obj.m\_Mc;

b = obj.m\_b;

L2 = obj.m\_L2;

Mp2 = obj.m\_Mp2;

b2 = obj.m\_b2;

Mode = obj.m\_Mode;

if del\_t ~= 0

switch Mode

case 'Velocity'

u = (obj.U - u\_old)/del\_t;

u\_old = obj.U;

otherwise

u = obj.U;

end

[~, Q] = ode45(@(T,Q)dequations(T,Q,u,L,Mp,Mc,b,L2, Mp2, b2, Mode),T,Q);

Q = Q(end,:);

while Q(1) > 2\*pi

Q(1) = Q(1) - 2\*pi;

end

while Q(1) < -2\*pi

Q(1) = Q(1) + 2\*pi;

end

obj.m\_Q = Q(:);

end

%NOW UPDATE THE PATCH OBJECTS

obj.DRAW

end

function DRAW(obj)

POS = obj.m\_Q(5);

%CART

V = obj.m\_Vbase;

V(:,1) = V(:,1) + POS;

set(obj.m\_Hbase, 'Vertices',V)

%AXLE

V = obj.m\_Vaxle;

V(:,1) = V(:,1) + POS;

set(obj.m\_Haxle, 'Vertices',V)

theta1 = obj.m\_Q(1) + pi;

%ROD BASE

V = obj.m\_Vrodbase;

V = [obj.Ry(-theta1) \* V']';

V(:,1) = V(:,1) + POS;

set(obj.m\_Hrodbase, 'Vertices',V)

%ROD

V = obj.m\_Vrod;

V = [obj.Ry(-theta1) \* V']';

V(:,1) = V(:,1) + POS;

x\_off = V(:, 1);

set(obj.m\_Hrod, 'Vertices',V)

%MASS

V = obj.m\_Vmass;

V = [obj.Ry(-theta1) \* V']';

V(:,1) = V(:,1) + POS;

set(obj.m\_Hmass, 'Vertices',V)

% Second Pendulum

% Calculate the rotation and translation for the second rod

theta2 = obj.m\_Q(2) + pi; % Assume theta2 is the angle of the second pendulum

V = obj.m\_Vrod2;

V = [obj.Ry(-theta2) \* V']';

% Offset position based on the first pendulum’s length and angle

x\_offset = POS - obj.m\_L \* sin(theta1);

z\_offset = obj.m\_L \* cos(theta1);

V(:,1) = V(:,1) + x\_offset; % Translate the rod

V(:,3) = V(:,3) + z\_offset;

set(obj.m\_Hrod2, 'Vertices', V);

% Update the position of the second mass

V = obj.m\_Vmass2;

V = [obj.Ry(-theta2) \* V']';

V(:,1) = V(:,1) + x\_offset;

V(:,3) = V(:,3) + z\_offset; % Translate the mass

set(obj.m\_Hmass2, 'Vertices', V);

end

function R = Ry(obj, angle)

R = [cos(angle) 0 sin(angle) ;

0 1 0 ;

-sin(angle) 0 cos(angle) ];

end

function out = GETTIME(obj)

tmp = clock;

out = 3600\*tmp(4) + 60\*tmp(5)+tmp(6);

end

end

end

***Modified dequations.m file***

function qdot = dequations(T,q,u,L,mp,mc,b,L2, mp2, b2, Mode)

%Define some parameters that we can use in our differential equations.

mt = mp+mc;

g = 9.81;

t1 = q(1);

t2 = q(2);

w1 = q(3);

w2 = q(4);

x = q(5);

v = q(6);

s1 = sin(t1);

c1 = cos(t1);

s2 = sin(t2);

c2 = cos(t2);

s21 = sin(t2-t1);

c21 = cos(t2-t1);

m1 = mp;

m2 = mp2;

L1 = L;

A = u;

qdot(1,1) = w1; %w1

qdot(2,1) = w2; %w2

prob\_mat = [L1\*c21,L2;

L1\*(m1+m2),L2\*m2\*c21];

prob\_vec = [-A\*m2\*c2-L1\*m2\*w1^2\*s21-m2\*g\*s2;

-A\*(m1+m2)\*c1+m2\*L2\*w2^2\*s21-(m1+m2)\*g\*s1];

prob\_vec\_damp = [-b2\*L2\*w2-b2\*L1\*w1\*c21-b2\*v\*c2;

-(b+b2)\*L1\*w1-(b+b2)\*v\*c1-b2\*L2\*w2\*c21];

qdot(3:4,1) = prob\_mat\(prob\_vec+prob\_vec\_damp);

qdot(5,1) = v; %v

qdot(6,1) = A; %a

end

***Modified source.m file to implement RLS update***

% source.m

% clear all

% clear classes

clc

%ADD HEBI LIBRARY FOR KEYBOARD INPUT

addpath('hebi');

%% May need to change the next line to "kb = HebiKeyboard();" if you get a keyboard not connected error

kb = HebiKeyboard('native',1); %Create a HibiKeyboard object.

%CONSTANTS

mc = 3; %Mass of Cart

g = 9.81; %Gravity

Mode = 'Acceleration'; %Velocity or Force or Acceleration

min\_cycle = 1/60; %Minimum cycle time

L1 = 0.7; %Length

m1 = 0.21; %Mass of Payload

b1 = 0.0003; %Damping factor

L2 = 0.7;

m2 = 0.078;

b2 = 0.0003;

a\_max = 0.326;

v\_max = 0.0726;

%exact frequencies

TH\_true = [(L1+L2)\*g\*(m1+m2)/(L1\*L2\*m1);

g^2\*(m1+m2)/(L1\*L2\*m1);

-1/L1;

g\*(m1+m2)/(L1\*L2\*m1)]

R\_ = m2/m1;

B = sqrt((1+R\_)^2\*(1/L1+1/L2)^2-4\*(1+R\_)/(L1\*L2));

w\_n = sqrt(g/2)\*sqrt((1+R\_)\*(1/L1+1/L2)+[-1;1]\*B)

%Setup Filtering

t1\_buffer = zeros(1,7);

v\_buffer = zeros(1,7);

[num,den] = tfdata(zpk([],[0.7,0.7,0.6,0.5],1,-1));

filter\_order = max(length(num{1}),length(den{1}));

Bf = [den{1},zeros(1,filter\_order-length(den{1}))];

Af = [num{1},zeros(1,filter\_order-length(num{1}))];

t1f\_buffer = zeros(1,filter\_order);

ddt1f\_buffer = zeros(1,filter\_order);

ddddt1f\_buffer = zeros(1,filter\_order);

ddxf\_buffer = zeros(1,filter\_order);

ddddxf\_buffer = zeros(1,filter\_order);

%Setup RLS

P = eye(4)\*0.1;

L1\_est = L1;

m1\_est = m1;

L2\_est = 0.1;

m2\_est = 0.1;

THETA = [(L1\_est+L2\_est)\*g\*(m1\_est+m2\_est)/(L1\_est\*L2\_est\*m1\_est);

g^2\*(m1\_est+m2\_est)/(L1\_est\*L2\_est\*m1\_est);

-1/L1\_est;

g\*(m1\_est+m2\_est)/(L1\_est\*L2\_est\*m1\_est)];

forgetting\_factor = 1;

%Setup Input Shaping

v\_input\_buffer = zeros(1,length(0:min\_cycle:5));

% T\_n = zeros(size(w\_n));

% T\_n = 2\*pi./w\_n\_final;

% Ai = [0.25 0.25 0.25 0.25];

% ti = [0,T\_n(2),T\_n(1),T\_n(1)+T\_n(2)]/2;

%PREPARE THE FIGURE WINDOW & GET PENDULUM OBJECT

figure(1); clf; set(gcf,'KeyPressFcn','1;');

Pend = PENDULUM('InitialStates',[0 0 0 0 0 0], 'WorkspaceLength', 8, ...

'Damping', b1, 'MassCart', mc, 'MassPendulum', m1, 'PendulumLength', L1, ...

'SecondPendulumLength', L2, 'SecondPendulumMass', m2, 'SecondPendulumDamping', b2, ...

'Mode', Mode);

%PREPARE THE STREAMING FIGURE WINDOWS

figure(2); clf

subplot(3,1,1);

Stream\_Theta = STREAM\_AXIS\_DATA(gca);

ylabel('\theta')

subplot(3,1,2);

Stream\_Pos = STREAM\_AXIS\_DATA(gca);

ylabel('Position')

subplot(3,1,3);

Stream\_Vel = STREAM\_AXIS\_DATA(gca);

ylabel('Velocity')

figure(3); clf

subplot(4,1,1);

Stream\_TH1 = STREAM\_AXIS\_DATA(gca);

ylabel('\Theta\_1')

subplot(4,1,2);

Stream\_TH2 = STREAM\_AXIS\_DATA(gca);

ylabel('\Theta\_2')

subplot(4,1,3);

Stream\_wn1 = STREAM\_AXIS\_DATA(gca);

ylabel('wn\_1')

subplot(4,1,4);

Stream\_wn2 = STREAM\_AXIS\_DATA(gca);

ylabel('wn\_2')

% setup pre-programmed input and history

% time\_quit = 20;

% t\_plot = 0:min\_cycle:time\_quit;

% U\_plot = zeros(size(t\_plot));

% U\_plot(t\_plot<4) = 1;

% U\_plot(t\_plot>=6&t\_plot<10) = -1;

%

% t1\_plot = zeros(size(t\_plot));

% w1\_plot = zeros(size(t\_plot));

% w2\_plot = zeros(size(t\_plot));

%BEGINNING OF SIMULATION LOOP

quit = 0;

del = min\_cycle;

now = 0;

cycle\_idx = 0;

while quit == 0

cycle\_idx = cycle\_idx+1;

%KEEP TRACK OF TIME

% now = GETTIME;

now = now+del;

%CHECK IF THE ESC, F1 KEY IS BEING PRESSED

kbinput = read(kb);

if kbinput.ESC

quit = 1;

end

if now>=time\_quit

quit=1;

end

%GET VR FROM THE KEYBOARD

U\_manual = GET\_MANUAL\_COMMAND('LEFT','RIGHT',kb); % Use LEFT and RIGHT arrow keys

%The States

Q = Pend.Q;

t1 = Q(1); % Angle of first pendulum

t2 = Q(2); % Angle of second pendulum

w1 = Q(3); % Angular velocity of first pendulum

w2 = Q(4); % Angular velocity of second pendulum

x = Q(5);

v = Q(6);

c1 = cos(t1);

s1 = sin(t1);

c2 = cos(t2);

s2 = sin(t2);

c21 = cos(t2-t1);

s21 = sin(t2-t1);

% Filter inputs to RLS

T = min\_cycle;

t1\_buffer(2:end) = t1\_buffer(1:end-1);

t1\_buffer(1) = t1;

v\_buffer(2:end) = v\_buffer(1:end-1);

v\_buffer(1) = v;

t1f\_buffer(2:end) = t1f\_buffer(1:end-1);

Ad = [0 0 0 1 0 0 0];

Cd = 1;

t1f\_buffer(1) = (Ad\*t1\_buffer'\*Cd-Bf(2:end)\*t1f\_buffer(2:end)')/Bf(1);

ddt1f\_buffer(2:end) = ddt1f\_buffer(1:end-1);

Ad = [1/90 -3/20 3/2 -49/18 3/2 -3/20 1/90];

Cd = 1/T^2;

ddt1f\_buffer(1) = (Ad\*t1\_buffer'\*Cd-Bf(2:end)\*ddt1f\_buffer(2:end)')/Bf(1);

ddddt1f\_buffer(2:end) = ddddt1f\_buffer(1:end-1);

Ad = [-1/6 2 -13/2 28/3 -13/2 2 -1/6];

Cd = 1/T^4;

ddddt1f\_buffer(1) = (Ad\*t1\_buffer'\*Cd-Bf(2:end)\*ddddt1f\_buffer(2:end)')/Bf(1);

ddxf\_buffer(2:end) = ddxf\_buffer(1:end-1);

Ad = [-1/60 3/20 -3/4 0 3/4 -3/20 1/60];

Cd = 1/T;

ddxf\_buffer(1) = (Ad\*v\_buffer'\*Cd-Bf(2:end)\*ddxf\_buffer(2:end)')/Bf(1);

ddddxf\_buffer(2:end) = ddddxf\_buffer(1:end-1);

Ad = [1/8 -1 13/8 0 -13/8 1 -1/8];

Cd = 1/T^3;

ddddxf\_buffer(1) = (Ad\*v\_buffer'\*Cd-Bf(2:end)\*ddddxf\_buffer(2:end)')/Bf(1);

t1f = Af\*t1f\_buffer';

ddt1f = Af\*ddt1f\_buffer';

ddddt1f = Af\*ddddt1f\_buffer';

ddxf = Af\*ddxf\_buffer';

ddddxf = Af\*ddddxf\_buffer';

%Solve recursive least squares

y = ddddt1f;

PHI = [-ddt1f;-t1f;ddddxf;ddxf];

% lam\_min = 0.7;

% lam\_gain = 0.001;

% forgetting\_factor = lam\_min+(1-lam\_min)\*exp(-lam\_gain\*(PHI'\*diag([1,100,0.01,1])\*PHI));

e = y-THETA'\*PHI;

THETA = THETA+P\*PHI\*e/(1+PHI'\*P\*PHI);

P = (P-P\*(PHI\*PHI')\*P/(1+PHI'\*P\*PHI))/forgetting\_factor;

wn1 = sqrt(abs(THETA(1)-sqrt(abs(THETA(1)^2-4\*THETA(2))))/2);

wn2 = sqrt(abs(THETA(1)+sqrt(abs(THETA(1)^2-4\*THETA(2))))/2);

T\_n = 2\*pi./[wn1,wn2];

Ai = [0.25 0.25 0.25 0.25];

ti = [0,T\_n(2),T\_n(1),T\_n(1)+T\_n(2)]/2;

ti\_idx = min(round(ti/min\_cycle)+1,length(v\_input\_buffer));

%Actuate the Pendulum

%get input

v\_input = v\_max\*U\_manual;

% v\_input = v\_max\*U\_plot(cycle\_idx);

%input shaper

v\_input\_buffer(2:end) = v\_input\_buffer(1:end-1);

v\_input\_buffer(1) = v\_input;

v\_shaped = sum(Ai.\*v\_input\_buffer(ti\_idx));

%motor input

a\_des = max(min((v\_shaped-v)/del,a\_max),-a\_max);

Pend.U = a\_des;

%Trace states

Stream\_Theta.stream(now, Q(1));

Stream\_Pos.stream(now, Q(5));

Stream\_Vel.stream(now, Q(6));

Stream\_TH1.stream(now, THETA(1));

Stream\_TH2.stream(now, THETA(2));

Stream\_wn1.stream(now, wn1);

Stream\_wn2.stream(now, wn2);

% t1\_plot(cycle\_idx) = t1;

% w1\_plot(cycle\_idx) = wn1;

% w2\_plot(cycle\_idx) = wn2;

%update screen

drawnow

%HANDLE ANY IDLE TIME

% finish = GETTIME;

% while finish <= now + min\_cycle

% finish = GETTIME;

% end

% del = finish - now;

end

w\_n\_final = [wn1,wn2];

%--------------------------------------------------

function Uout = GET\_MANUAL\_COMMAND(xm, xp, kb)

umax = 1;

U = 0;

kbinput = read(kb);

if isfield(kbinput,xm) && isfield(kbinput,xp) % if using meta keys such as arrows

if kbinput.(xp) % move right with specified meta key (e.g. RIGHT arrow)

U = U + umax;

end

if kbinput.(xm) % move left with specified key (e.g. LEFT arrow)

U = U - umax;

end

else % using regular keys

if sum(kbinput.keys) ~= 0

if kbinput.keys(xp)% move right with specified key (e.g. 'd')

U = U + umax;

end

if kbinput.keys(xm) % move left with specified key (e.g. 'a')

U = U - umax;

end

end

end

Uout = U;

end

%--------------------------------------------------

function out = GETTIME

%time = GETTIME returns the current cpu clock time in milliseconds

tmp = clock;

out = 3600\*tmp(4) + 60\*tmp(5)+tmp(6);

end

***Tower crane RLS trajectory creator***

clear; clc

%% Orange Bottle Parameters

% Hoist Length = 700 mm

% Payload Length = 0.77

% Payload Mass = 0.078

% Theoretical: w1 = 2.9781; w2 = 5.1922

% RLS: w1 = 3.1; w2 = 9.0

%% 0.5kg Mass Parameters

% Hoist Length = 960 mm

% Payload Length = 0.96

% Payload Mass = 0.5

% Theoretical: w1 = 2.5398; w2 = 8.3942

% RLS: w1 = 5.00; w2 = 8.80

%% Disk Mass Parameters

% Hoist Length = 700 mm

% Payload Length = 0.675

% Payload Mass = 0.19 kg

% Theoretical: w1 = 2.92; w2 6.603

% RLS: w1 = 4.54; w2 = 9.00

%% Combined Mass Parameters

% Hoist Length = 700 mm

% Payload Length = 0.660

% Payload Mass = 0.132 kg

% Theoretical: w1 = 2.99; w2 6.0381

% RLS: w1 = 2.89; w2 = 7.87

A = [.25 .25 .25 .25]; % Shaper amplitudes

% T = [0 .6051 1.0547 1.66]; % Orange Theory Shaper times

% T = [0 .3491 1.0134 1.3625]; % Orange RLS Shaper times

% T = [0 .4758 1.0756 1.5514]; % Disc Theory Shaper times

% T = [0 .3491 0.6359 0.9850]; % Disc RLS Shaper times

% T = [0 .3743 1.2369 1.6112]; % 0.5kg Mass Theory Shaper times

% T = [0 .357 0.6283 0.9853]; % 0.5kg Mass RLS Shaper times

% T = [0 .5203 1.0476 1.5679]; % Combined Theory Shaper times

T = [0 .3992 1.0871 1.4862]; % Combined RLS Shaper times

timeStep = .05; % ms

vCommand = [ones(80, 1);zeros(40,1);-ones(80,1)]\*100; % unshaped step command

v1 = zeros(1000, 1); v2 = zeros(1000, 1); v3 = zeros(1000, 1); v4 = zeros(1000, 1);

v1(1:length(vCommand)) = A(1)\*vCommand;

v2(floor(T(2)/timeStep) + 1:floor(T(2)/.05) + length(vCommand)) = A(2)\*vCommand;

v3(floor(T(3)/timeStep) + 1:floor(T(3)/.05) + length(vCommand)) = A(3)\*vCommand;

v4(floor(T(4)/timeStep) + 1:floor(T(4)/.05) + length(vCommand)) = A(4)\*vCommand;

v = v1 + v2 + v3 + v4; % shaped command