Two Link Arm Control

Group 13 || Tyler J. Adams | Henrietta Odiete | Pavel Shering

***Abstract*—This report describes the design, implementation and simulation of a two link robotic arm and its controller that is required to move the robot to specified destinations.**

I. Introduction

THE project contains three major components: the design of the two link robot arm, controller design and implementation, and optimization. The robot arm is required to start at point A, then move in a specific square

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$$

(as shown in Fig. 1) while constraining the oscillations of the end effector to within 4 [mm] for at least 0.5 seconds. The end effector of the arm must not collide with the environment (walls at surrounding the trajectory) and the overall mass of the arm is limited to 0.75$[kg]$. The robotic arm only contains joint angle measurements without a way to sense velocities.

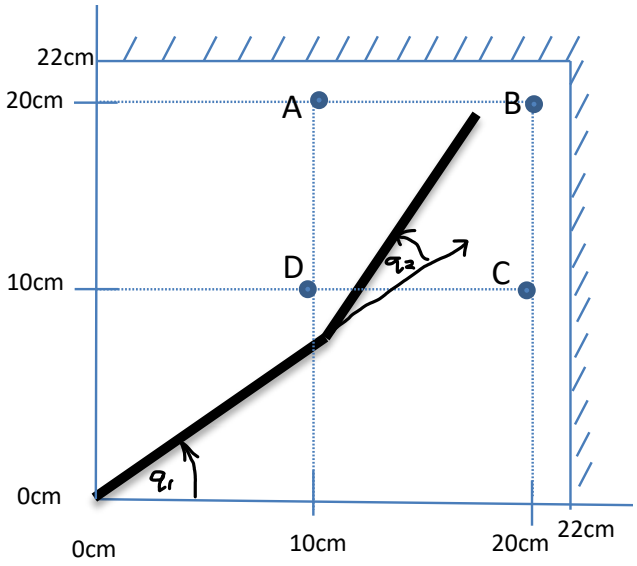

Fig. 1. Environment set up for the robotic arm

II. Robot Arm

The two link robot arm motion equations were provided by the project description which are used for the non linear model is created in MATLAB (displayed in Listing 3 located in the Source Code section). The model is used during the simulation to test the controller implementation and the ability of the robot to perform the required tasks. Its a design choice to make the link lengths approximately equal ($l_1 = l_2$) and out of titanium with 1.4$[kg/m]$ density and $6 \pm 2[N/(rad/s)]$ friction at the joints to optimize for time and energy consumption based on testing preformed after final controller was finished.

III. Controller

Design: Two main controller designs are considered for the robotic arm. The push and pull controllers (Fig. 2), each with state feedback and an estimator, as well as a regulator. The pull controller components are linearized about the destination point and the robot arm is pulled towards the target applying input. While for push control, the linearization is about the staring point and the robot arm is given a step input to push towards the target.

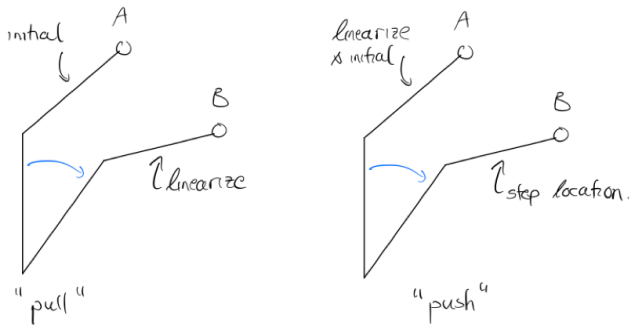

Fig. 2. Push vs Pull controller

The general control architecture is shown in Fig. 3. The controller takes in the desired angles for the robot arm and outputs the required torque for the robot arm to achieve the destination. The robot non linear motion equations are linearized into the form of

$$\dot{\hat{x}} = A\hat{x} + Bu + F(y - \hat{y})$$

$$y = Cx + Du$$

$$u = -K\hat{x}$$

(shown in Listing 5).

Due to dusty environment, friction model in the system varies the mass of the two links by 10% with total mass remaining the same. Additionally, small amounts of Gaussian noise is present in the joint angles. Optimal control approach is used to design the controller, since pole placement state feedback and state estimation techniques have limited performance in presence of noise, and the fact that there no measurements of join velocities. To achieve zero steady state error and minimize the energy used by the system the Linear Quadratic Regulator (LQR) is used in conjunction with Kalman filter to predict joint velocities and reduce noise in the controller.

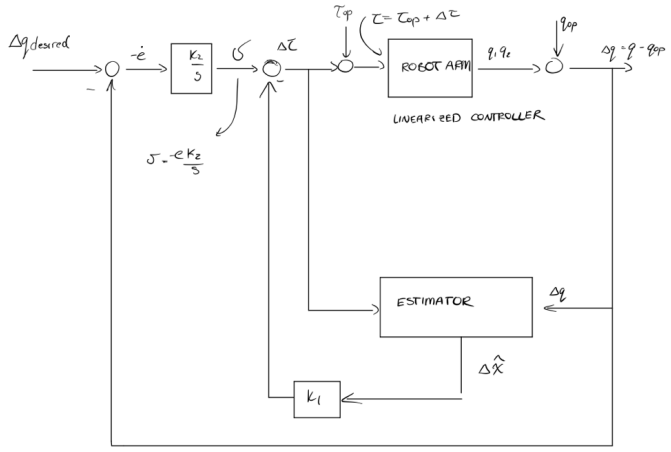

Fig. 3. Overall Controller Architecture

Implementation: The system is linearized about every trajectory angle (shown in Listing 2) to decrease implementation testing as this decreases computation time and stored in a list to be used in the RobotController script. The trajectory angles are obtained from inverse kinematics of the path the robot is required to take (Listing 1)). The number of in-between points is explained later in the optimization section.

Listing 1. Creating angle trajectories out of distance trajectories

```
target_A = [0.10 0.20];
target_B = [0.20 0.20];
target_C = [0.20 0.10];
target_D = [0.10 0.10];

% milestones = [ A B C D A ]
milestones = [ target_A;
               target_B;
               target_C;
               target_D;
               target_A];

my_inbw_points = 2 + 10;
my_traj_xy = [];

% delete adjacent duplicates
for k = 1:length(milestones)-1
    xvalues = linspace(milestones(k,1), milestones(k+1,1), my_inbw_points)
        ↪ ;
    yvalues = linspace(milestones(k,2), milestones(k+1,2), my_inbw_points)
        ↪ ;
    for i = 1:my_inbw_points
        temp(i, :) = [xvalues(i) yvalues(i)];
    end
    my_traj_xy = [my_traj_xy; temp];
end

c2_ik = (my_traj_xy(:,1).^2 + my_traj_xy(:,2).^2 - l1^2 - l2^2)/(2*l1*l2);
s2_ik = sqrt(1 - c2_ik.^2);
THETA2D = atan2(s2_ik,c2_ik);
k1_ik = l1 + l2*c2_ik;
k2_ik = l2*s2_ik;
THETA1D = atan2(my_traj_xy(:,2),my_traj_xy(:,1)) - atan2(k2_ik,k1_ik);
my_traj_angles = [THETA1D THETA2D];
```

The detailed implementation of both push and pull controllers is shown in Listings 5 and 6, the pull controller is within the "my_pull" if statement.

Listing 2. Linearization about every angle of the robot trajectory

```
for w = 1:length(my_traj_angles)

    u1r = double(subs(T1, [x1 x2 x3 x4], [my_traj_angles(w,1), 0,
        ↪ my_traj_angles(w,2), 0]));
    u2r = double(subs(T2, [x1 x2 x3 x4], [my_traj_angles(w,1), 0,
        ↪ my_traj_angles(w,2), 0]));

    A = double(subs(A_jacobian, [x1 x2 x3 x4 u1 u2], [my_traj_angles(w,1),
        ↪  0, my_traj_angles(w,2), 0, u1r, u2r]));
    B = double(subs(B_jacobian, [x1 x2 x3 x4 u1 u2], [my_traj_angles(w,1),
        ↪  0, my_traj_angles(w,2), 0, u1r, u2r]));

    % kalman filter
    [F P_se ev_se] = lqr(A.', C.', Q_kal, R_kal);
    F = F';
    % optimal control
    [K P_sf ev_sf] = lqr(A, B, Q_lqr, R_lqr);

    % regulator
    A_aug = [ A zeros(4,2);
              C zeros(2,2)];
    B_aug = [B ; zeros(2,2)];

    [K_aug P_sf ev_sf] = lqr(A_aug, B_aug, Q_lqr_aug, R_lqr_aug);
    k1 = K_aug(:,1:end-2);
    k2 = K_aug(:,end-1:end);

    A_lst(:,:,w) = A;
    B_lst(:,:,w) = B;
    F_lst(:,:,w) = F;
    K_lst(:,:,w) = K;
    K1_lst(:,:,w) = k1;
    K2_lst(:,:,w) = k2;
    Tss_lst(:,:,w) = [u1r u2r];
end
```

Furthermore, the pausing functionality for each point A, B, C, D is the same for each controller. It is a requirement that the end effector must settle and wait at each point to within 4 [mm]. The waiting code checks to make sure that the end effector distance is within the 4 [mm] and starts the timer. If the end effector is still settling and leaves the 4 [mm] diameter, the timer restarts. Fig. 4 and Fig. 5 show the settling of the end effector within the required distance and waiting the required time at target B (others are not shown to save space).

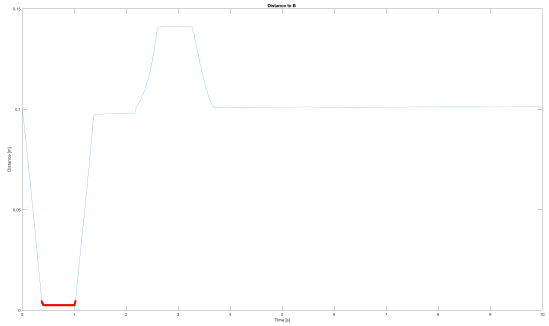

Fig. 4. Robot Distance to Target B throughout the entire trajectory

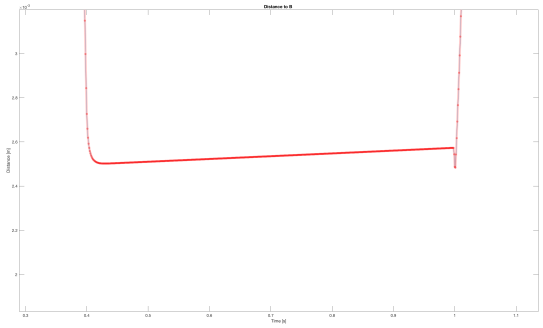

Fig. 5. Robot Distance to Target B zoomed on the region for settling time and distance within 4 [mm]

IV. Optimization

The controller is optimized for minimizing time to completion, as well as for minimizing energy used to complete the task.

Minimizing the time to completion has to be balanced with energy consumption as the faster the robot moves the more torque is applied to drive the robot arm from start to target. First the push controller is abandoned as it is more sluggish than the pull and has overshoot in its response. Fig. 6 and Fig. 7 display the drastic difference in controller behaviour for pull vs push. The final result comparison can be seen in Fig. 8 and Fig. 9.

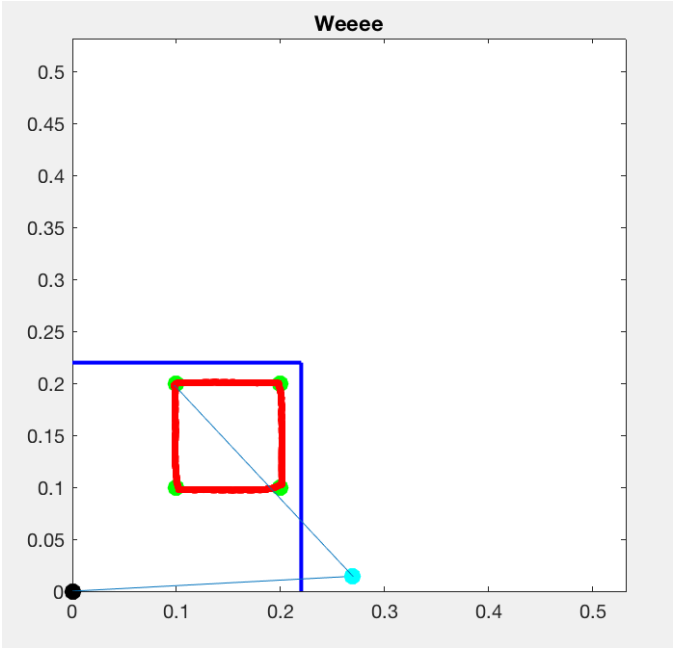

Fig. 6. Pull Controller Behaviour

First, the pull controller was tested with and without intermediate points between the targets (A,B,C,D,A). Without intermediate points the controller takes more time for completion of the task. Thus, 10 intermediate milestones are

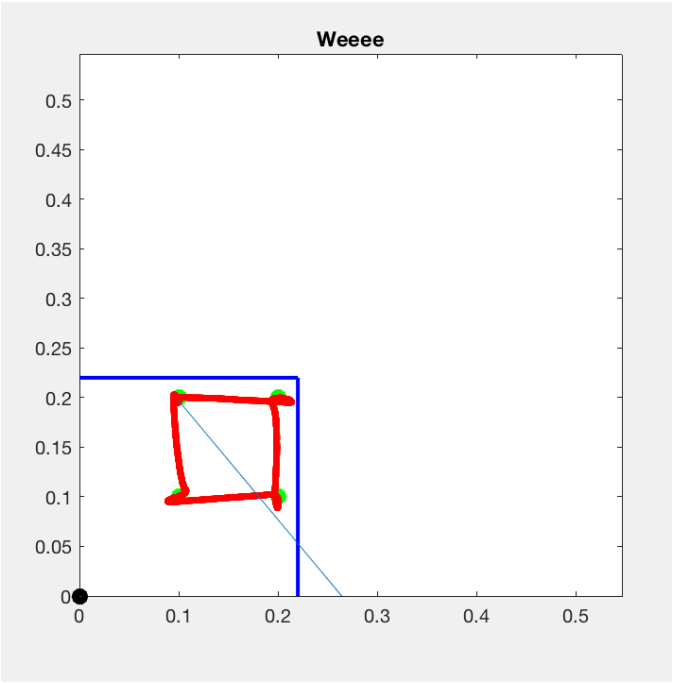

Fig. 7. Push Controller Behaviour

added to speed up the controller, ultimately allowing the robot to use a more accurate controller as it follows the trajectory, as the linearization is done at every point.

The robot itself is chosen to be made out of titanium as it lessens the link lengths which allows for less movement of the arm and therefore uses less energy. However, steel is not used due to issues with the ODE solver in MATLAB as short link lengths create singularity A matrices which prevent inversion and blow up the system. This problem can be fixed using the Stiff ODEs solver, but due to time constraints it was not attempted.

Next, the pull controller is further optimized by adjusting the R_kal, Q_kal, and Q_lqr matrices. The R_lqr matrix is left untouched as an identity. The R_kal is determined by the noise specification given Gaussian noise with standard deviation of a $1/3$ of a degree with mean of zero. The matrix is further multiplied by a factor of 1.7 to increase the controller robustness to measurement noise. The Q_kal is also determined by the uncertainty of the system model, however, due to missing specifications this was determined experimentally. Lastly, the R_lqr is a 6x6 matrix containing the weighting for each state as well as the error. Increasing any of the weights will cause the LQR to minimize that component. Thus the weights for angles and error are increased by 45,000x and 10,000 respectively. The values are chosen experimentally to balance energy consumption and the time to competition. The final controller is a pull controller with the following Kalman filter and LQR parameters are listed in Eq. 1, 2, 3, 4.

$$R_{kal} = 1.0e-04 * \begin{bmatrix} 0.5754 & 0 \\ 0 & 0.5754 \end{bmatrix} \quad (1)$$

$$Q_{kal} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad (2)$$

$$R_{lqr} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3)$$

$$Q_{lqr} = \begin{bmatrix} 45000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 45000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix} \quad (4)$$

V. Results

With noise and no variations in parameters in terms of mass, friction and link lengths, with $l_1 = 0.3$ [m] the robot is able to complete the course within 3.7460 [s] and 80.17s [J] of energy consumed (Fig. 9). Further, testing the controller with $l_1 = 0.27$ [m] and variation is robot components the system is able to compete the course within 4.13 [s] with 78.24 [J] consumed (Fig. 10). Overall the controller has good performance and robustness as can be seen from self testing. However, the definition of "good" cannot be full bench marked until the results from competition. Further LQR parameter tuning is possible through entropy search and cost evaluation technique, however, due to time constraints, this is not implemented.

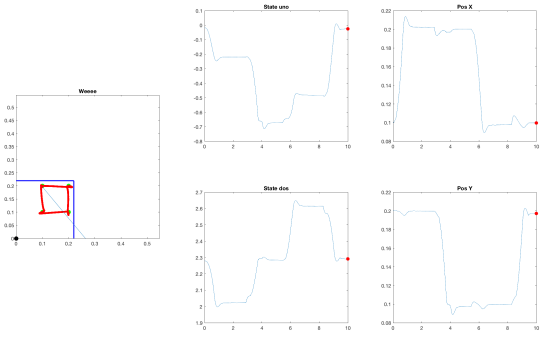

Fig. 8. Push Controller Final Task Completion Results

VI. Source Code

Listing 3. Non linear model of the two link arm

```
function Xdot = non_lin_roboarm(t, Xnl, T, sl1, sl2, sm1, sm2, sg, sc1,
    ↪ sc2)
    u2 = T(2);
    u1 = T(1);

    x1 = Xnl(1);
```

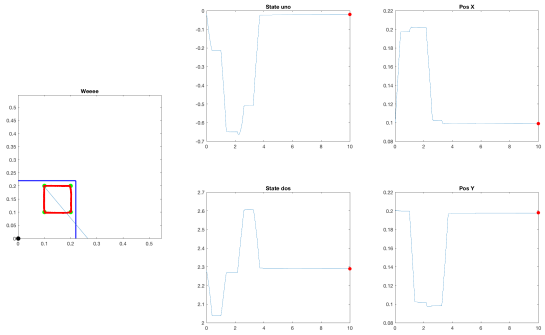

Fig. 9. Pull Controller Final Task Completion Results

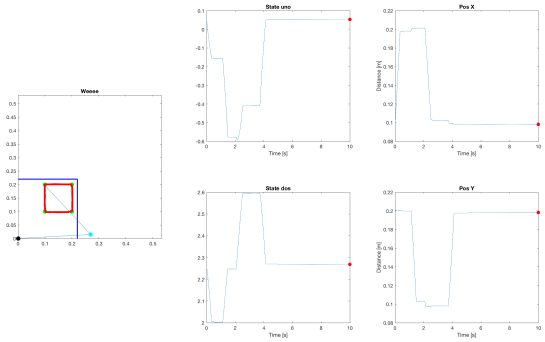

Fig. 10. Pull Controller Final Task Completion Results with Parameter Variations

```
    x2 = Xnl(2);
    x3 = Xnl(3);
    x4 = Xnl(4);

    Xdot = [ Xnl(2);
            (3*(4*sl2*u1 - 4*sl2*u2 - 6*sl1*u2*cos(x3) - 4*sc1*sl2*x2 +
                ↪ ...
            4*sc2*sl2*x4 + 6*sc2*sl1*x4*cos(x3) - 2*sg*sl1*sl2*sm1*cos(x1)
                ↪ ...
            - 4*sg*sl1*sl2*sm2*cos(x1) + 2*sl1*sl2^2*sm2*x2^2*sin(x3) +...
            2*sl1*sl2^2*sm2*x4^2*sin(x3) + 3*sl1^2*sl2*sm2*x2^2*cos(x3)*
                ↪ sin(x3)...
            + 3*sg*sl1*sl2*sm2*cos(x1 + x3)*cos(x3) + ...
            4*sl1*sl2^2*sm2*x2*x4*sin(x3)))/(4*sl1^2*sl2*sm1 + ...
        12*sl1^2*sl2*sm2 - 9*sl1^2*sl2*sm2*cos(x3)^2);
             Xnl(4);
            -(3*(4*sl2^2*sm2*u1 - 12*sl1^2*sm2*u2 - 4*sl1^2*sm1*u2 - ...
            4*sl2^2*sm2*u2 - 4*sc1*sl2^2*sm2*x2 + 4*sc2*sl1^2*sm1*x4 + ...
            12*sc2*sl1^2*sm2*x4 + 4*sc2*sl2^2*sm2*x4 + 6*sl1*sl2*sm2*u1*
                ↪ cos(x3) ...
            - 12*sl1*sl2*sm2*u2*cos(x3) + 2*sl1*sl2^3*sm2^2*x2^2*sin(x3) +
                ↪ ...
            6*sl1^3*sl2*sm2^2*x2^2*sin(x3) + 2*sl1*sl2^3*sm2^2*x4^2*sin(x3
                ↪ ) + ...
            6*sg*sl1^2*sl2*sm2^2*cos(x1 + x3) - 4*sg*sl1*sl2^2*sm2^2*cos(
                ↪ x1) + ...
            2*sl1^3*sl2*sm1*sm2*x2^2*sin(x3) + 4*sl1*sl2^3*sm2^2*x2*x4*sin
                ↪ (x3) + ...
            6*sl1^2*sl2^2*sm2^2*x2^2*cos(x3)*sin(x3) + 3*sl1^2*sl2^2*sm2
                ↪ ^2*x4^2*cos(x3)*sin(x3) ...
            + 3*sg*sl1*sl2^2*sm2^2*cos(x1 + x3)*cos(x3) - 6*sc1*sl1*sl2*
                ↪ sm2*x2*cos(x3) + ...
            12*sc2*sl1*sl2*sm2*x4*cos(x3) - 6*sg*sl1^2*sl2*sm2^2*cos(x1)*
                ↪ cos(x3) + ...
```

```
            2*sg*sl1^2*sl2*sm1*sm2*cos(x1 + x3) - 2*sg*sl1*sl2^2*sm1*sm2*
                ↪ cos(x1) + ...
            6*sl1^2*sl2^2*sm2^2*x2*x4*cos(x3)*sin(x3) - 3*sg*sl1^2*sl2*sm1
                ↪ *sm2*cos(x1)*cos(x3)))...
            /(12*sl1^2*sl2^2*sm2^2 - 9*sl1^2*sl2^2*sm2^2*cos(x3)^2 + 4*sl1
                ↪ ^2*sl2^2*sm1*sm2)];
end
```

Listing 4. Constants

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %You NEED these constants
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% titanium
l1 = 0.27; %link 1 length
% noise testing
% m1 = (1.4*l1 - 0.1*1.4*l1) + rand(1)*(0.2*1.4*l1); %link 1 mass
% m2 = 0.75 - m1; %link 2 mass
% l2 = m2/1.4; %link 2 length
% c1 = 4 + rand(1)*4; % damping of link 1
% c2 = 4 + rand(1)*4; % damping of link 2
% actual params without variations
m1 = 1.4*l1; %l1 mass
m2 = 0.75-m1;
l2 = m2/1.4; %link 2 length
c1 = 6; % damping of link 1
c2 = 6; % damping of link 2
g=3.7;%acceleration due to gravity m/s^2 on mars

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
my_time = 0;
my_params = [m1, m2, l1, l2, c1, c2];
my_delta_xhat = [0 0 0 0]';
my_error = [0; 0];
my_delta_t = 0.001;
my_energy = 0;
my_pull = 1;
my_wait = 1;
my_sd = (1/3)*(pi / 180);

if my_pull
    my_traj_count = 1;
else
    my_traj_count = 2;
end
my_traj_threshold = 0.05;
my_tspan = 0:0.001:10; % set time interval

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRACJECTORY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
target_A = [0.10 0.20];
target_B = [0.20 0.20];
target_C = [0.20 0.10];
target_D = [0.10 0.10];

% milestones = [ A B C D A ]
milestones = [ target_A;
               target_B;
               target_C;
               target_D;
               target_A];

my_inbw_points = 2 + 10;
my_traj_xy = [];

% This variable is used to check if within 4mm of milestone
my_milestone_ctr = 2;

% Global boolean for if we are waiting for 0.5s
recordedMilestoneStartTime = 0;
% Global variabble for waiting for 0.5s
milestoneStartTime = 0;

% delete adjacent duplicates
for k = 1:length(milestones)-1
    xvalues = linspace(milestones(k,1), milestones(k+1,1), my_inbw_points)
        ↪ ;
    yvalues = linspace(milestones(k,2), milestones(k+1,2), my_inbw_points)
        ↪ ;
    for i = 1:my_inbw_points
        temp(i, :) = [xvalues(i) yvalues(i)];
    end

    my_traj_xy = [my_traj_xy; temp];
end

c2_ik = (my_traj_xy(:,1).^2 + my_traj_xy(:,2).^2 - l1^2 - l2^2)/(2*l1*l2);
s2_ik = sqrt(1 - c2_ik.^2);
THETA2D = atan2(s2_ik,c2_ik);

k1_ik = l1 + l2*c2_ik;
k2_ik = l2*s2_ik;
THETA1D = atan2(my_traj_xy(:,2),my_traj_xy(:,1)) - atan2(k2_ik,k1_ik); %
    ↪ theta1 is deduced

% forward kinematics test
% X_pred = l1 * cos(THETA1D) + l2 * cos(THETA1D + THETA2D); % compute x
    ↪ coordinates
% Y_pred = l1 * sin(THETA1D) + l2 * sin(THETA1D + THETA2D);

my_traj_angles = [THETA1D THETA2D];
x_0 = [my_traj_angles(1,1) 0 my_traj_angles(1,2) 0]';
```

Listing 5. Constants2

```
%% TOTAL SYSTEM
% symbolic model variables
syms q1 q1d q1dd T1
syms q2 q2d q2dd T2

syms x1 x2 u1 %q1
syms x3 x4 u2 %q2

% change of variables
%   [q1 q1d q2 q2d]
x = [x1 x2 x3 x4];
%   [T1 T2]
u = [u1 u2];

syms sg sm1 sm2 sl1 sl2 sc1 sc2

eqn1 = 0 == -T1 + [(sm1*sl1.^2)/3 + (sm2*sl2.^2)/12 + sm2*(sl1.^2 + sl2
    ↪ .^2/4 + sl1*sl2*cos(q2))]*q1dd + ...
    [(sm2*sl2.^2)/3 + (sm2*sl1*sl2)/2*cos(q2)]*q2dd - ...
    sm2*sl1*sl2*sin(q2)*q1d*q2d - (sm2*sl1*sl2*sin(q2))/2*q2d.^2 + (sm1*
        ↪ sl1/2+sm2*sl1)*sg*cos(q1) + ...
    (sm2*sl2)/2*sg*cos(q1+q2)+sc1*q1d;

eqn2 = 0 == -T2 + [(sm2*sl2.^2)/3 + (sm2*sl1*sl2)/2*cos(q2)]*q1dd + (sm2*
    ↪ sl2.^2)/3*q2dd + ...
    (sm2*sl1*sl2*sin(q2))/2*q1d.^2 + (sm2*sl2)/2*sg*cos(q1+q2) + sc2*q2d;

% change of variables
eqn1 = subs(eqn1, [q1 q1d q2 q2d T1 T2], [x1 x2 x3 x4 u1 u2]);
eqn2 = subs(eqn2, [q1 q1d q2 q2d T1 T2], [x1 x2 x3 x4 u1 u2]);

% solve for q1dd and q2dd
sol = solve([eqn1 eqn2], [q1dd q2dd]);

% solve for Steady State Torque where velocities are zero
T = solve([eqn1 eqn2], [u1 u2]);
SST1 = subs(T.u1, [x2, x4, q1dd, q2dd], [0, 0, 0, 0]);
SST2 = subs(T.u2, [x2, x4, q1dd, q2dd], [0, 0, 0, 0]);

%formulating the state space
y = [x1 x3];
xdot = [x(2) sol.q1dd x(4) sol.q2dd].';

%% LINEAR SYSTEM
xdot = subs(xdot, [sg, sm1, sm2, sl1, sl2, sc1, sc2], [g, m1, m2, l1, l2,
```

```
      ↪ c1, c2]);
A_jacobian = jacobian(xdot, x.');
B_jacobian = jacobian(xdot, u.');
% C = jacobian(y, x.');
% D = jacobian(y, u.');
C = [1 0 0 0;
     0 0 1 0];

T1 = subs(SST1, [sg, sm1, sm2, sl1, sl2, sc1, sc2], [g, m1, m2, l1, l2, c1
      ↪ , c2]);
T2 = subs(SST2, [sg, sm1, sm2, sl1, sl2, sc1, sc2], [g, m1, m2, l1, l2, c1
      ↪ , c2]);

% EKF and LQR design
sd = (1/3)*(pi / 180); % third of degree measurement error

if my_pull
    R_kal = my_sd^2 .* eye(2) *1.7;
    Q_kal = eye(4) .* ([1 2 1 2]');

    R_lqr = eye(2);
    Q_lqr = eye(4) .* [250000 1 250000 1]';

    R_lqr_aug = eye(2);
    Q_lqr_aug = eye(6) .* [45000 1 45000 1 10000 10000]';
else
    R_kal = my_sd^2 .* eye(2);
    Q_kal = eye(4) .* ([1 1 1 1]'*1.5);

    R_lqr = eye(2);
    Q_lqr = eye(4) .* [25000 1 25000 1]';

    R_lqr_aug = eye(2);
    Q_lqr_aug = eye(6) .* ([1 1 1 1 100000 100000]');
end

for w = 1:length(my_traj_angles)

    u1r = double(subs(T1, [x1 x2 x3 x4], [my_traj_angles(w,1), 0,
          ↪ my_traj_angles(w,2), 0]));
    u2r = double(subs(T2, [x1 x2 x3 x4], [my_traj_angles(w,1), 0,
          ↪ my_traj_angles(w,2), 0]));

    A = double(subs(A_jacobian, [x1 x2 x3 x4 u1 u2], [my_traj_angles(w,1),
          ↪  0, my_traj_angles(w,2), 0, u1r, u2r]));
    B = double(subs(B_jacobian, [x1 x2 x3 x4 u1 u2], [my_traj_angles(w,1),
          ↪  0, my_traj_angles(w,2), 0, u1r, u2r]));

    % kalman filter
    [F P_se ev_se] = lqr(A.', C.', Q_kal, R_kal);
    F = F';
    % optimal control
    [K P_sf ev_sf] = lqr(A, B, Q_lqr, R_lqr);

    % regulator
    A_aug = [ A zeros(4,2);
              C zeros(2,2)];
    B_aug = [B ; zeros(2,2)];

    [K_aug P_sf ev_sf] = lqr(A_aug, B_aug, Q_lqr_aug, R_lqr_aug);
    k1 = K_aug(:,1:end-2);
    k2 = K_aug(:,end-1:end);

    A_lst(:,:,w) = A;
    B_lst(:,:,w) = B;
    F_lst(:,:,w) = F;
    K_lst(:,:,w) = K;
    K1_lst(:,:,w) = k1;
    K2_lst(:,:,w) = k2;
    Tss_lst(:,:,w) = [u1r u2r];
end

%% set the steady state torques
tau_0 = Tss_lst(1,:,1)';
```

Listing 6. Robot Controller Implementation

```
my_mu = [0 0];
sigma = [my_sd my_sd];
noise = mvnrnd(my_mu,sigma.^2)';

if noise(1) > 1
   noise = [1 noise(2)]';
end
if noise(2) > 1
   noise = [noise(1) 1]';
end

% q = q + noise;

if my_pull
    %% pull
    A = A_lst(:,:,my_traj_count);
    B = B_lst(:,:,my_traj_count);
    F = F_lst(:,:,my_traj_count);
    K = K_lst(:,:,my_traj_count);
    k1 = K1_lst(:,:,my_traj_count);
    k2 = K2_lst(:,:,my_traj_count);
    U_op =  Tss_lst(:,:,my_traj_count)';
    q_op = [my_traj_angles(my_traj_count,1) 0 my_traj_angles(my_traj_count
          ↪ ,2) 0]';

    my_delta_xhat = (my_delta_xhat) + ( (A-F*C)*(my_delta_xhat) ...
        + B*(U-U_op) + F*(q - my_traj_angles(my_traj_count,:)')  ) *
            ↪ 0.001;

    % state estimate
%     U = -K * (my_delta_xhat) + U_op;

    % regulator
    my_error  = my_error + (q - my_traj_angles(my_traj_count,:)') * 0.001;
    delta_U = -k1 * my_delta_xhat - k2 * my_error;
    U = delta_U + U_op;

    cur_pos = my_delta_xhat + q_op;
    X_pred = l1 * cos(cur_pos(1)) + l2 * cos(cur_pos(1) + cur_pos(3)); %
          ↪ compute x coordinates
    Y_pred = l1 * sin(cur_pos(1)) + l2 * sin(cur_pos(1) + cur_pos(3));

    X = l1 * cos(my_traj_angles(my_traj_count,1)) + l2 * cos(
          ↪ my_traj_angles(my_traj_count,1) +...
        my_traj_angles(my_traj_count,2));
    Y = l1 * sin(my_traj_angles(my_traj_count,1)) + l2 * sin(
          ↪ my_traj_angles(my_traj_count,1) + ...
        my_traj_angles(my_traj_count,2));

    if (sqrt((X_pred - X)^2 + (Y_pred - Y)^2) <= 0.004)
        if my_wait
            if my_milestone_ctr <= length(milestones) &&  ...
                    sqrt((X_pred - milestones(my_milestone_ctr, 1))^2 +...
                    (Y_pred - milestones(my_milestone_ctr, 2))^2) < 0.004
                if ~recordedMilestoneStartTime
                    milestoneStartTime = t;
                    recordedMilestoneStartTime = 1;
                    disp('Start time:')
                    t
                    return
                elseif recordedMilestoneStartTime
                    if (t - milestoneStartTime >= 0.6)
                        % Set up future check for next milestone
                        my_milestone_ctr = my_milestone_ctr + 1
                        % Reset variable and go do the torque of the next
                        % point
                        recordedMilestoneStartTime = 0;
                    elseif (t - milestoneStartTime < 0.6)
                        % Output the same torque as last time
                        % and exit the script to avoid
                        % outputting torque of the next point in traj
                        U = U_op;
                        return
                    end
                end
            end
        end
```

```
        my_traj_count = my_traj_count+1;
        if (my_traj_count == length(my_traj_angles)+1)
            my_traj_count = length(my_traj_angles);
            U = U_op;
            return
        else
            my_delta_xhat = [my_traj_angles(my_traj_count-1,1) 0
                ↪ my_traj_angles(my_traj_count-1,2) 0]' - ...
            [my_traj_angles(my_traj_count,1) 0 my_traj_angles(
                ↪ my_traj_count,2) 0]';
        end
    else
        if recordedMilestoneStartTime == 1
            % You left the 0.004 after you entered it
            milestoneStartTime = 0;
            recordedMilestoneStartTime = 0;
        end
    end

else
    %% push
    A = A_lst(:,:,my_traj_count-1);
    B = B_lst(:,:,my_traj_count-1);
    F = F_lst(:,:,my_traj_count-1);
    K = K_lst(:,:,my_traj_count-1);
    k1 = K1_lst(:,:,my_traj_count-1);
    k2 = K2_lst(:,:,my_traj_count-1);
    U_op = Tss_lst(:,:,my_traj_count-1)';
    q_op = [my_traj_angles(my_traj_count-1,1) 0 my_traj_angles(
        ↪ my_traj_count-1,2) 0]';
    q_dest = [my_traj_angles(my_traj_count,1) 0 my_traj_angles(
        ↪ my_traj_count,2) 0]';

    my_delta_xhat = (my_delta_xhat)  + ( (A-F*C)*(my_delta_xhat) ...
        + B*(U-U_op) + F*(q  - q_op([1,3],:)) )*0.001;

    my_error = my_error + (q - q_dest([1,3],:)) * 0.001;
    delta_U = -k1 * (my_delta_xhat) - k2 * my_error;
    U = delta_U + U_op;

    cur_pos = my_delta_xhat + q_op;
    X_pred = l1 * cos(cur_pos(1)) + l2 * cos(cur_pos(1) + cur_pos(3)); %
        ↪ compute x coordinates
    Y_pred = l1 * sin(cur_pos(1)) + l2 * sin(cur_pos(1) + cur_pos(3));

    X = l1 * cos(my_traj_angles(my_traj_count,1)) + l2 * cos(
        ↪ my_traj_angles(my_traj_count,1) +...
        my_traj_angles(my_traj_count,2));
    Y = l1 * sin(my_traj_angles(my_traj_count,1)) + l2 * sin(
        ↪ my_traj_angles(my_traj_count,1) +...
        my_traj_angles(my_traj_count,2));


    if (sqrt((X_pred - X)^2 + (Y_pred - Y)^2) <= 0.004)
        if my_wait
            if my_milestone_ctr <= length(milestones) &&  ...
                    sqrt((X_pred - milestones(my_milestone_ctr, 1))^2 +...
                    (Y_pred - milestones(my_milestone_ctr, 2))^2) < 0.004
                if ~recordedMilestoneStartTime
                    milestoneStartTime = t;
                    recordedMilestoneStartTime = 1;
                    disp('Start time: ')
                    t
                    return
                elseif recordedMilestoneStartTime
                    if (t - milestoneStartTime >= 0.6)
                        % Set up future check for next milestone
                        my_milestone_ctr = my_milestone_ctr + 1
                        % Reset variable and go do the torque of the next
                        % point
                        recordedMilestoneStartTime = 0;
                    elseif (t - milestoneStartTime < 0.6)
                        % Output the same torque as last time
                        % and exit the script to avoid
                        % outputting torque of the next point in traj
                        U = U_op;
                        return
                    end
                end
            end
        end
    end

    my_traj_count = my_traj_count+1;
    if (my_traj_count == length(my_traj_angles)+1)
        my_traj_count = length(my_traj_angles);
        U = U_op;
        return
    else
        my_delta_xhat = [my_traj_angles(my_traj_count-1,1) 0
            ↪ my_traj_angles(my_traj_count-1,2) 0]' - ...
        [my_traj_angles(my_traj_count,1) 0 my_traj_angles(
            ↪ my_traj_count,2) 0]';
    end
else
    if recordedMilestoneStartTime == 1
        % You left the 0.004 after you entered it
        milestoneStartTime = 0;
        recordedMilestoneStartTime = 0;
    end
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ENERGY CALCULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
my_energy = my_energy + U'*U*0.001;
```

Listing 7. The script that is used to run the entire simulation

```
% ENERGY: 76.5326J without 10% mass stuff and noise at links
% TIME: 3.7460s with 30 l1
% with noise and varitions l1 = 27 i get 4.1 s and energry = 60.33J
%
%     R_kal = sd^2 .* eye(2) *1.5;
%     Q_kal = eye(4) .* ([1 2 1 2]'*1);
%
%     R_lqr = eye(2);
%     Q_lqr = eye(4) .* [250000 1 250000 1]';
%
%     R_lqr_aug = eye(2);
%     Q_lqr_aug = eye(6) .* [25000 1 25000 1 10000 10000]';

%this is the general format of the simulator script
close all; clear all; clc;

%% initializing code
% provides system parameters and and Eqm_point
Constants;
% solves for the state space system and provides S.S torques
Constants2;

%initial conditions
X0 = x_0;
U = tau_0; % init torque

[tout,qout] = ode45(@(time,x)non_lin_roboarm(time,x,U,l1,l2,m1,m2,...
    g,c1,c2),[0 my_delta_t],X0);
q=qout(end,[1,3])';

q_start = qout(:,[1,3])';
t_start = tout;
T=1;

history_q = zeros(length(my_tspan), 2);
history_torques = zeros(length(my_tspan), 2);

for t=my_tspan
   t;
   % check if robot meets requirements

   RobotControllerScript %your script is used here.
   [tout,qout] = ode45(@(time,x)non_lin_roboarm(time,x,U,l1,l2,m1,m2,...
```

```
        g,c1,c2),[t t+my_delta_t],qout(end,:));
    q=qout(end,[1,3])';

    history_q(T,:) = q;
    history_torques(T,:) = U;
    hist_my_est(T,:) = my_delta_xhat;
    T = T + 1;
end

my_tspan1 = [t_start' my_tspan];

history_q1 = [q_start'; history_q];
visualize(my_params, my_tspan1', history_q1(:,1), history_q1(:,2), '')

my_energy
```

Listing 8. The script that is used to visualize the entire simulation

```
%%Copyright Ilarion Kvitnevskiy 2018

function [ ] = visualize( params, t, q1, q2, file)

m1 = params(1);
m2 = params(2);
l1 =  params(3);
l2 =  params(4);
c1 = params(5);
c2 = params(6);

s = 0.8;
figure('units','normalized','outerposition',[(1-s)/2 (1-s)/2 s s]);

framerate = 5;
frameperiod = 1/framerate;

curtime = -frameperiod;
i = 1;

x1 = l1*cos(q1);
x2 = x1 + l2*cos(q1+q2);
y1 = l1*sin(q1);
y2 = y1 + l2*sin(q1+q2);

display(max(x2));
display(max(y2));

while i < size(t,1)
    curtime = min(curtime + frameperiod, t(end));
    while curtime > t(i)
        i = i + 1;
        continue
    end
    subplot(2,3,[1,4]);

    plot(0,0,'MarkerSize',30,'Marker','.', 'Color', 'black');
    title('Weeee');
    a1 = (l1+l2)*1.02;
    axis equal;
    axis([0, a1, 0, a1]);

    hold on;
    line( [0.22, 0.22],  [0, 0.22], 'LineWidth', 2, 'Color', 'blue')
    line( [0, 0.22],  [0.22, 0.22], 'LineWidth', 2, 'Color', 'blue')
    plot( 0.1, 0.2,'MarkerSize',30,'Marker','.', 'Color', 'green');
    plot( 0.2, 0.2,'MarkerSize',30,'Marker','.', 'Color', 'green');
    plot( 0.2, 0.1,'MarkerSize',30,'Marker','.', 'Color', 'green');
    plot( 0.1, 0.1,'MarkerSize',30,'Marker','.', 'Color', 'green');

    plot(x2(1:i), y2(1:i), 'MarkerSize', 10, 'Marker', '.', 'Color', 'red'
        ↪ )

    line([0, x1(i)], [0 y1(i)], 'LineWidth',0.5*m1+0.1);
    plot(x1(i),y1(i),'MarkerSize',30,'Marker','.', 'Color', 'cyan');

    line([x1(i), x2(i)], [y1(i), y2(i)], 'LineWidth',0.5*m2+0.1);

    hold off;

    subplot(2,3, 2);
    plot(t, q1);
    hold on;
    plot(t(i), q1(i), 'MarkerSize', 20, 'Marker', '.', 'Color', 'red');
    hold off;
    title('State uno');
    xlabel('Time [s]');

    subplot(2,3, 5);
    plot(t, q2);
    hold on;
    plot(t(i), q2(i), 'MarkerSize', 20, 'Marker', '.', 'Color', 'red');
    hold off;
    title('State dos');
    xlabel('Time [s]');

    subplot(2,3, 3);
    plot(t, x2);
    hold on;
    plot(t(i), x2(i), 'MarkerSize', 20, 'Marker', '.', 'Color', 'red');
    hold off;
    title('Pos X')
    ylabel('Distance [m]');
    xlabel('Time [s]');

    subplot(2,3, 6);
    plot(t, y2);
    hold on;
    plot(t(i), y2(i), 'MarkerSize', 20, 'Marker', '.', 'Color', 'red');
    hold off;
    title('Pos Y')
    ylabel('Distance [m]');
    xlabel('Time [s]');


    drawnow;
    if ~strcmp(file, '')
        frame = getframe(1);
        im = frame2im(frame);
        [imind,cm] = rgb2ind(im,256);
        dt = 0.05;
        if i == 1
            imwrite(imind,cm,file,'gif', 'Loopcount',inf, 'DelayTime', dt)
                ↪ ;
        else
            imwrite(imind,cm,file,'gif','WriteMode','append', 'DelayTime',
                ↪ dt);
        end
    end

end

    figure()
    plot(t, ((x2 - 0.10).^2 +(y2-0.2).^2).^(1/2));
    title('Distance to A ');
    ylabel('Distance [m]');
    xlabel('Time [s]');
    figure()
    plot(t, ((x2 - 0.20).^2 +(y2-0.2).^2).^(1/2));
    title('Distance to B ');
    ylabel('Distance [m]');
    xlabel('Time [s]');
    figure()
    plot(t, ((x2 - 0.20).^2 +(y2-0.1).^2).^(1/2));
    title('Distance to C ');
    ylabel('Distance [m]');
    xlabel('Time [s]');
    figure()
    plot(t, ((x2 - 0.10).^2 +(y2-0.1).^2).^(1/2));
    title('Distance to D ');
    ylabel('Distance [m]');
    xlabel('Time [s]');
end
```