Exoskeleton Simulation Model GUI

Michael Cheng

Introduction

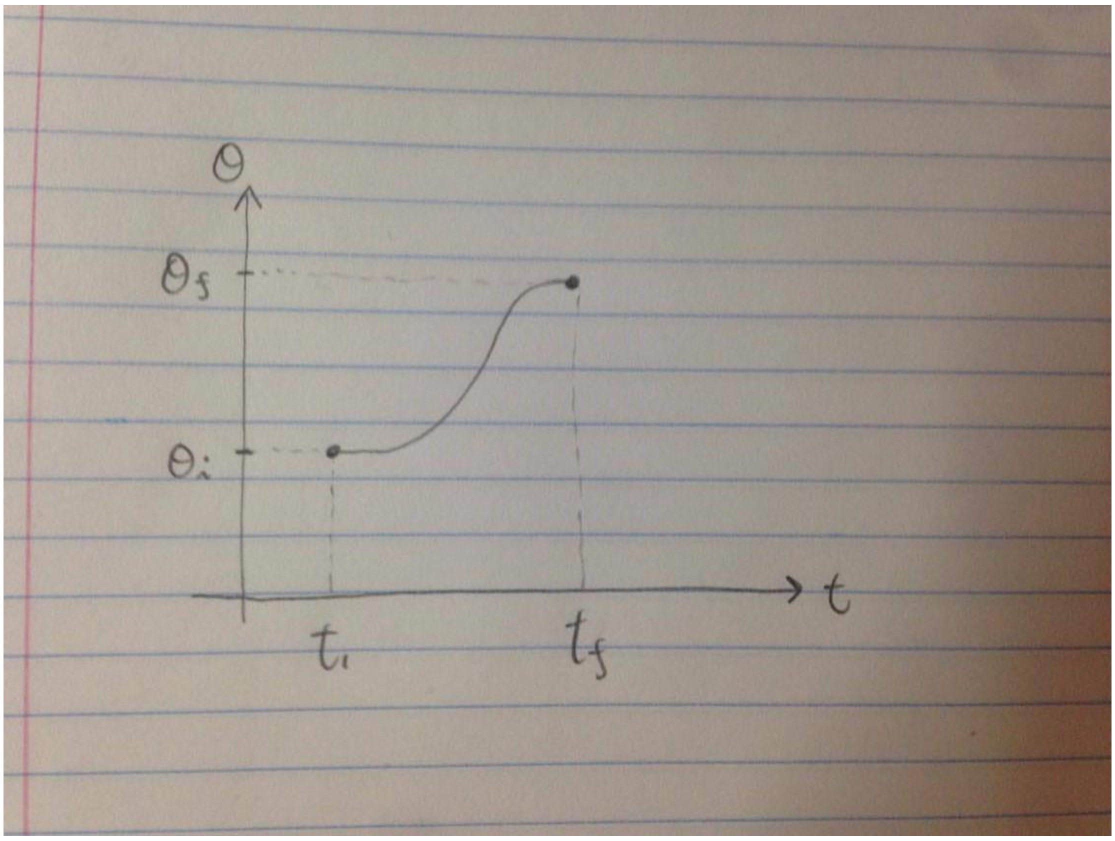
As I was working toward my initial project idea, which is simulating exoskeleton control and movement on the haptic paddle, I realize I had to change my goal. First, simulating just one motor of an exoskeleton is somewhat of a pointless objective. All motors contribute and receive torque from the whole exoskeleton system, even if I came up with a good algorithm and a robust simulation of an exoskeleton-human system, displaying the movements of one single motor in the system doesn't seem like good project material. Second, coming up with my own exoskeleton control scheme and algorithms was much more complex than I thought, building a human arm dynamic simulation to help with the research of exoskeleton movements is enough challenge within itself, let alone simulating Myoprocessor control and implementing them into an actual motor. Thus, building a simulation model of a human arm became my main goal for this project. Using the knowledge I acquired from the Robot Dynamics course and the dimensions of my own arm, I designed a MATLAB program and animation that simulates and calculates the dynamics of a human arm movement. I hope this will be a good tool in my future exoskeleton research.

Process

First, I want the program to have the ability to do "Inverse Kinematics" of an arm. Inverse kinematics, in this case, would be giving the program the coordinates and angle of the end effector (in this case, the hand), and it would generate a set of joint angles (the shoulder, elbow and wrist) that makes the end effector reach the desired coordinates. The inverse kinematics is generated from repeating the process of multiplying the error and a form of velocity Jacobian, with every multiplication, the error gets smaller and the current joint position will be closer to a configuration that satisfies the desired position.

With this concept as the basic foundation of my animation for the arm, and since the inverse kinematics calculation progress is already broken down in increments, my initial thought was to plot every increment of the inverse kinematics calculation to form an animation illusion. However, the movement generated in the animation was not smooth and logical, as shown in the "First Iteration" video, or the "test" MATLAB script.

To solve this, I implemented "3rd degree polynomial path planning" as the motion law for the simulation and the animation. That is, to generate a smooth path between the initial joint angles to the desired joint angles, I choose the initial and final angular velocities to both be zeros and the angle-time plot between the two angles to be a 3rd degree polynomial, example shown in the figure below:



With some calculations, we can derive the polynomial for this situation as below:

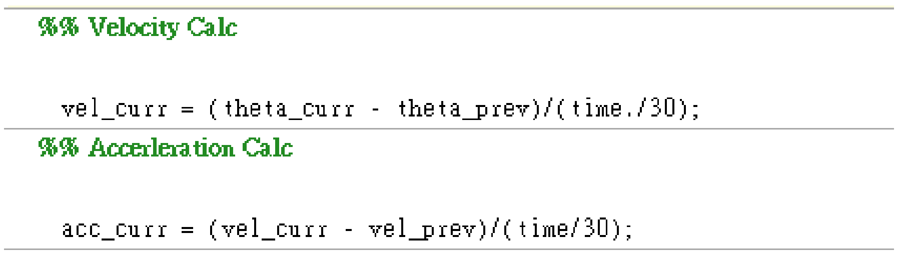
$$a = \frac{-12(\theta_1 - \theta_0)}{(t_1 - t_0)^3}$$

$$b = \frac{6(\theta_1 - \theta_0)}{(t_1 - t_0)^2}$$

$$\theta = \frac{1}{6}a(t_1 - t_0)^3 + \frac{1}{2}b(t_1 - t_0)^2 + \theta_0$$

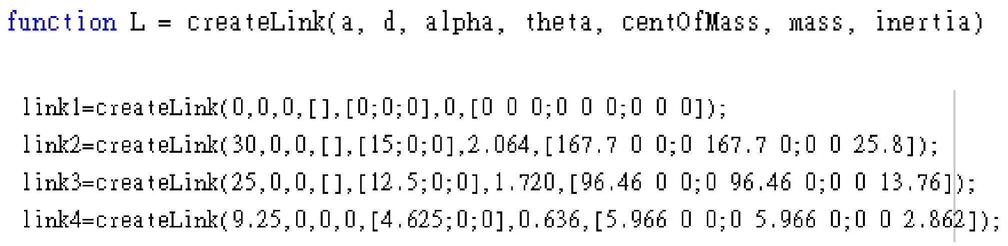
Since the initial angle position is known, the final desired angled already generated by the inverse kinematics function, and the time duration able to be set by the user, we will have the trajectory of the angle-time relations. Split the path into 30 increments and plot them with a pause time of (duration time)/30, then we will (theoretically) have an animation that simulates the full movement. In reality, MATLAB cannot run all the functions instantly, so the animation will have delays.

Having the joint angle configurations and the joint angle trajectory path would not be enough of a model for an exoskeleton research, the angular velocities, accelerations and torques of the joints will need to be known. The velocity and acceleration calculations were easy, we take the difference between the current value and the previous value, the divide it by the time increment length, as shown as the figure below:

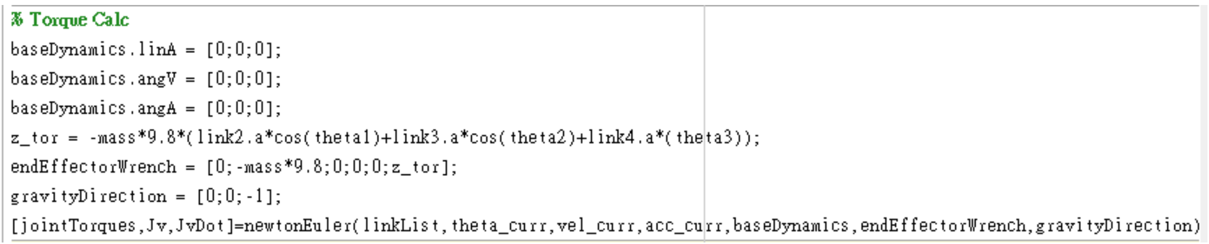


```
%% Velocity Calc

  vel_curr = (theta_curr - theta_prev)/(time./30);
%% Accerleration Calc

  acc_curr = (vel_curr - vel_prev)/(time/30);
```

The torque, on the other hand, is way more complicated. We use the "Newton Euler" method to calculate the joint torques. Methods and the codes for the Newton Euler will not be discussed here due to the complex nature of the method, the important part is knowing what the inputs for the "newtonEuler" function are. The length between each joint (which is obvious and needed for the inverse kinematics function), the mass and the center of mass of each part of the arm, the inertia tensor of the limb, the current angle, velocity, and acceleration of each joint are all the criteria for the torque calculation. The dimensions of this model are all based on my model and shown in the below MATLAB structure "links":



```
function L = createLink(a, d, alpha, theta, centOfMass, mass, inertia)

 link1=createLink(0,0,0,[],[0;0;0],0,[0 0 0;0 0 0;0 0 0]);
 link2=createLink(30,0,0,[],[15;0;0],2.064,[167.7 0 0;0 167.7 0;0 0 25.8]);
 link3=createLink(25,0,0,[],[12.5;0;0],1.720,[96.46 0 0;0 96.46 0;0 0 13.76]);
 link4=createLink(9.25,0,0,0,[4.625;0;0],0.636,[5.966 0 0;0 5.966 0;0 0 2.862]);
```

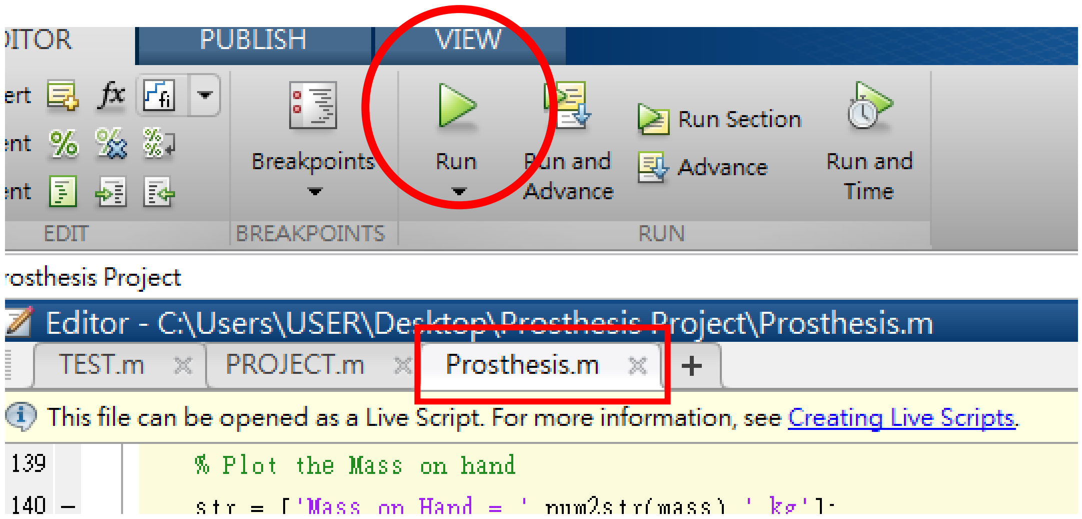


```
% Torque Calc
baseDynamics.linA = [0;0;0];
baseDynamics.angV = [0;0;0];
baseDynamics.angA = [0;0;0];
z_tor = -mass*9.8*(link2.a*cos(theta1)+link3.a*cos(theta2)+link4.a*(theta3));
endEffectorWrench = [0;-mass*9.8;0;0;0;z_tor];
gravityDirection = [0;0;-1];
[jointTorques,Jv,JvDot]=newtonEuler(linkList,theta_curr,vel_curr,acc_curr,baseDynamics,endEffectorWrench,gravityDirection)
```

With all the joint angles, joint angular velocities, joint angular accelerations and the joint torques all generated, we have a very thorough simulation of a human arm in a user friendly displayed MATLAB program. The codes for all the functions are all built for 3D use, but for the purpose of animation demo, I set my demo case to be on a 2D plane so we can have a better understanding of it.

User Manual

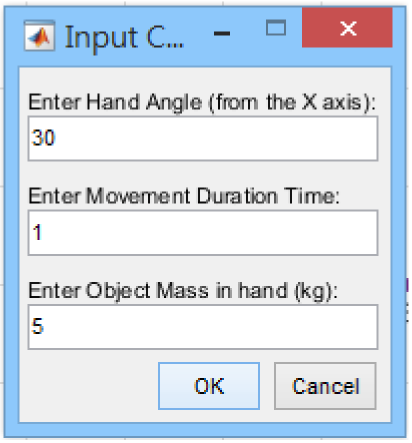
1. Run the script “Prosthesis”.



This file can be opened as a Live Script. For more information, see Creating Live Scripts.

```
% Plot the Mass on hand
str = ['Mass on Hand = ' num2str(mass) ' kg'];
```

2. Enter the desired hand angle, movement duration time, and the mass of the object being held in the hand



Input C...

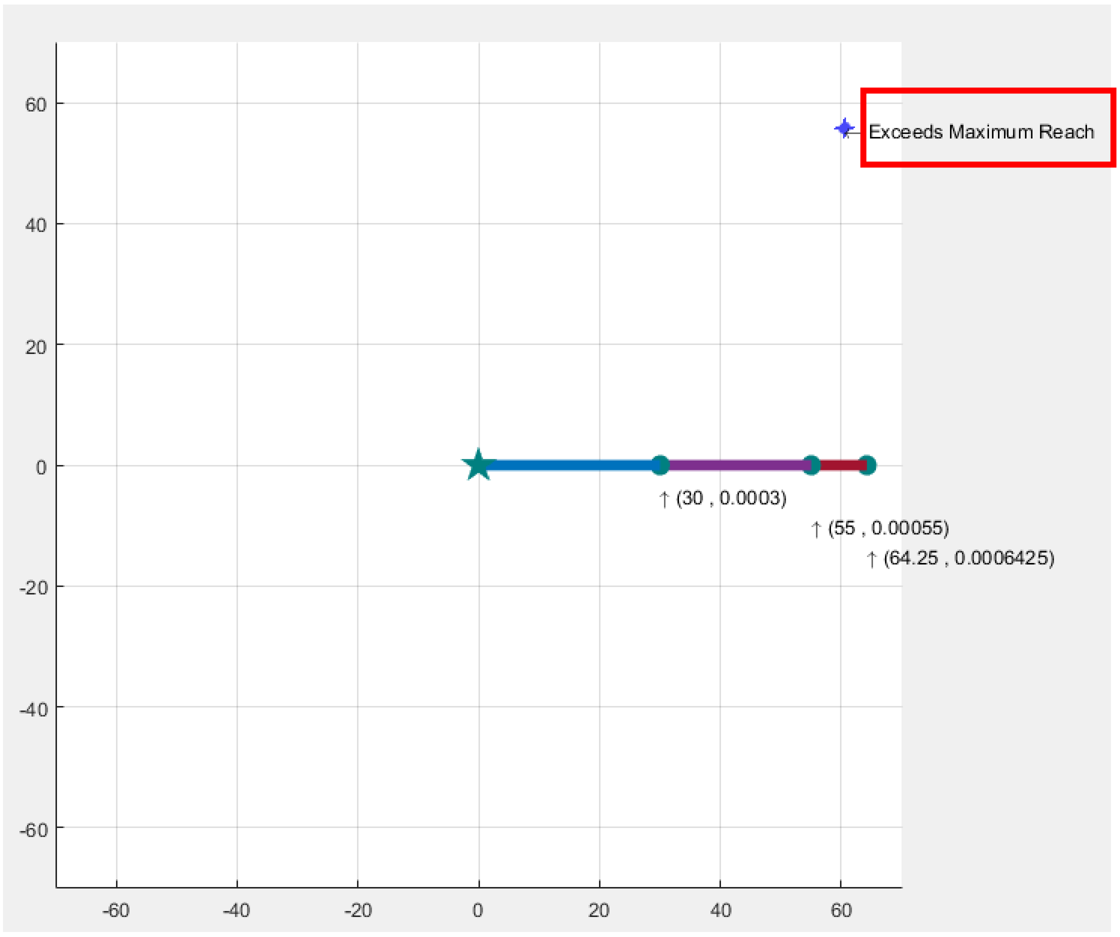
Enter Hand Angle (from the X axis): 30

Enter Movement Duration Time: 1

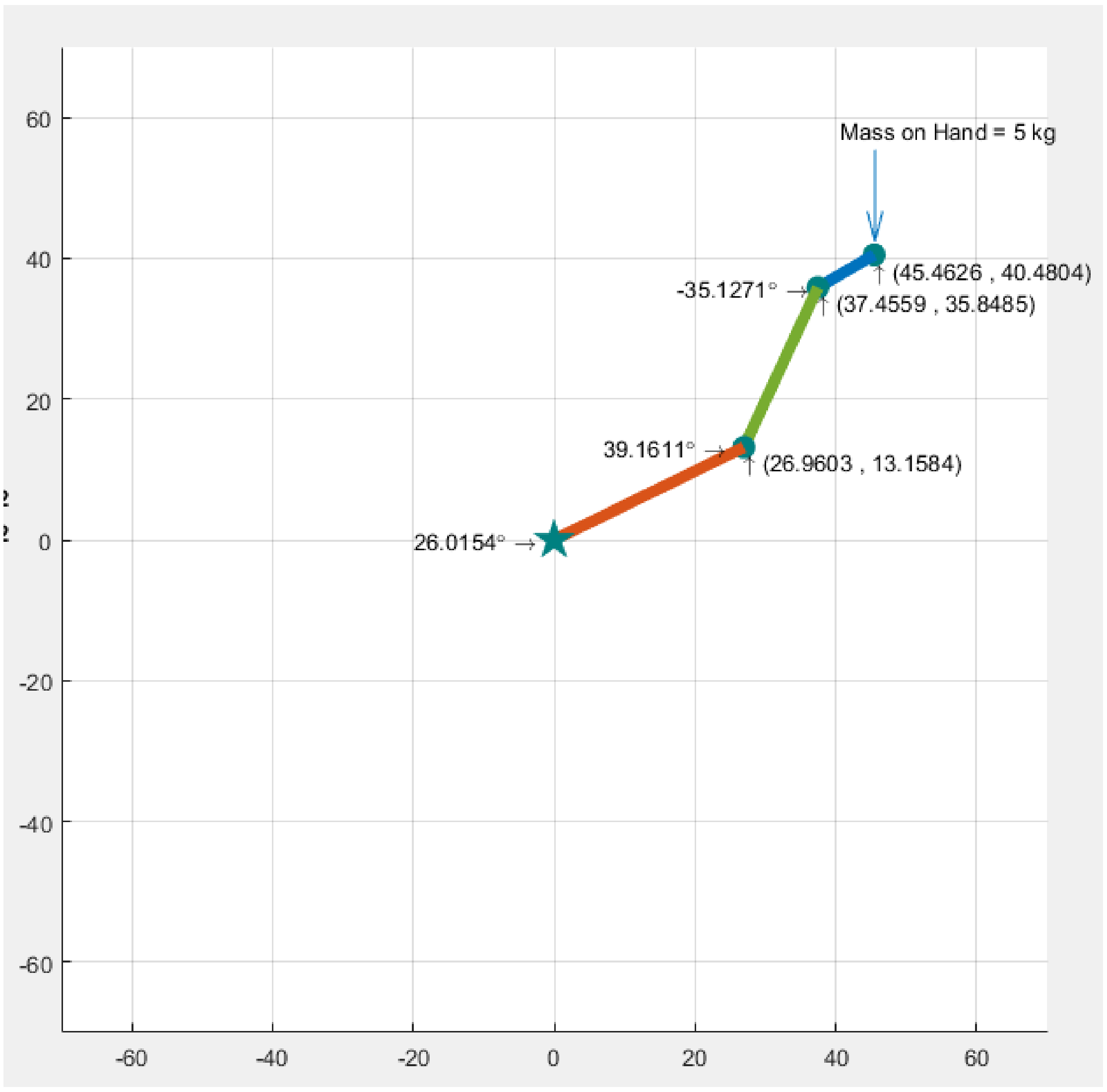
Enter Object Mass in hand (kg): 5

3. Click on the position you want the end effector to be, and the arm will settle into the desired configuration smoothly. If the arm can't reach, it will display the message of "Exceeds Maximum Reach".

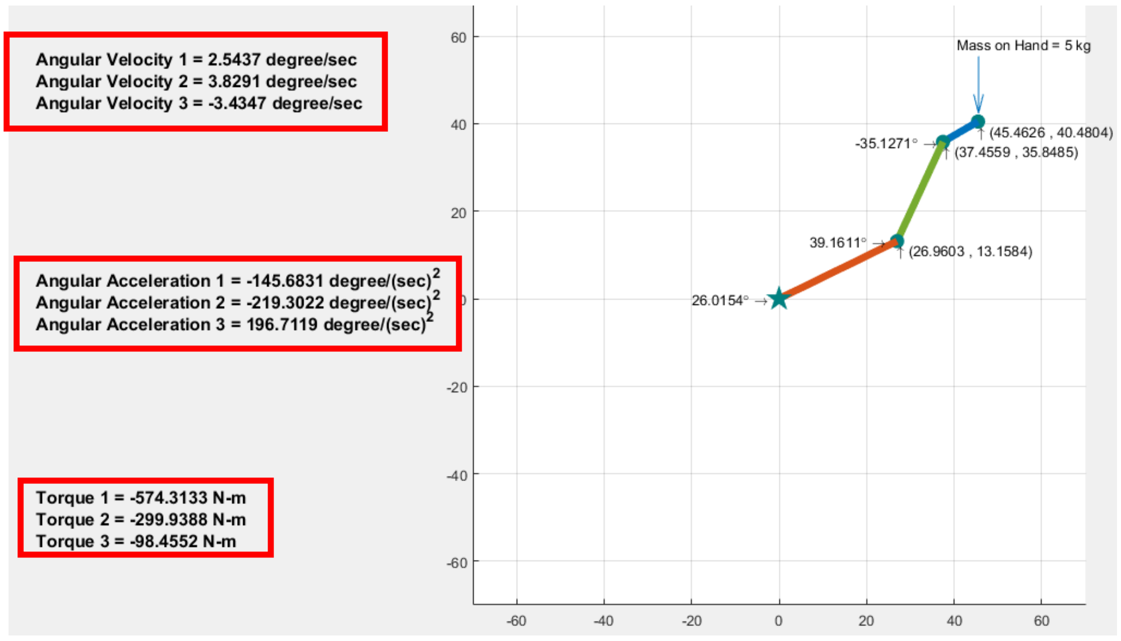
   Display of the "can't reach" situation:



Displaying the joint angle configuration of the desired end effector position:



4. Check to left side of the plot, there are the angular velocity, angular acceleration, and the torque of each joint:



```
Angular Velocity 1 = 2.5437 degree/sec
Angular Velocity 2 = 3.8291 degree/sec
Angular Velocity 3 = -3.4347 degree/sec
```

```
Angular Acceleration 1 = -145.6831 degree/(sec)^2
Angular Acceleration 2 = -219.3022 degree/(sec)^2
Angular Acceleration 3 = 196.7119 degree/(sec)^2
```

```
Torque 1 = -574.3133 N-m
Torque 2 = -299.9388 N-m
Torque 3 = -98.4552 N-m
```

5. The program is repeatable, the initial pose of the arm will be the final pose of the last configuration.

Demo

First Iteration:
https://youtu.be/xdLlZO4D3cE
Final Product:
https://youtu.be/XrjC_zOPgZw

Conclusion

I believe all the work on this MATLAB project has paid off, the displays are clear, all the information of the arm movement for exoskeleton research are all included, and the simulation animation is smooth. The only short coming of this project I would say is the capability of MATLAB is not enough to make the movement duration time the same as in the animation, though it does not affect the calculations.

This program will be really helpful for exoskeleton research or arm movement dynamics in general. In the future, I would like to continue my interest in exoskeletons, digging deeper into Myoprocessing controls.

References

[1] G.W. Lucas – *A Path Based on Third-Degree Polynomials Constrained at its Endpoints by Position, Orientation, and Speed* http://rossum.sourceforge.net/papers/CalculationsForRobotics/CubicPath.htm

[2] John J. Craig – *Introduction to Robotics Mechanics and Control 3rd Edition*

Appendix

Code for the main program named “prosthesis” (not including the functions, the other functions will be in the zip file):

```
link1=createLink(0,0,0,[],[0;0;0],0,[0 0 0;0 0 0;0 0 0]);
link2=createLink(30,0,0,[],[15;0;0],2.064,[167.7 0 0;0 167.7 0;0 0
25.8]);
link3=createLink(25,0,0,[],[12.5;0;0],1.720,[96.46 0 0;0 96.46 0;0 0
13.76]);
link4=createLink(9.25,0,0,0,[4.625;0;0],0.636,[5.966 0 0;0 5.966 0;0
0 2.862]);
linkList=[link1 link2 link3 link4];

paramListG = [0.00001;0;0;0];

figure
axis([-70 70 -70 70])
daspect([1 1 1])

%% Plot Initial Configuration of the Arm
clf;
for i = 1:1:length(paramListG)
   H(:,:,i) = dhFwdKine(linkList(1:i), paramListG(1:i));
end

for i = 1:1:length(paramListG)-1

   hold on;
   grid on;
   %plot3([H(1,4,i) H(1,4,i+1)],[H(2,4,i) H(2,4,i+1)],[H(3,4,i)
H(3,4,i+1)])

    % Plot the arms and the hand
    PLT = plot([H(1,4,i) H(1,4,i+1)],[H(2,4,i)
H(2,4,i+1)],'LineWidth',5);
    uistack(PLT, 'bottom')

    % Plot the elbow, and wrist joints
    CIR =
scatter( H(1,4,i+1),H(2,4,i+1),'o','LineWidth',5,'LineWidth',4,'Marke
rEdgeColor',[0 .5 .5],'MarkerFaceColor',[0 .5 .5]);
    uistack(CIR, 'bottom')

    % Plot the shoulder joint
    ORG =
scatter( 0,0,'p','LineWidth',5,'LineWidth',4,'MarkerEdgeColor',[0 .5
.5],'MarkerFaceColor',[0 .5 .5]);

    % Add coordinates of the joint s
    str = ['\uparrow (',num2str(H(1,4,i+1)),' ,
',num2str(H(2,4,i+1)),')'];
    TXT = text(H(1,4,i+1),H(2,4,i+1)-5*i,str);
    uistack(TXT, 'top')

    % Set plot specs
    axis([-70 70 -70 70])
    daspect([1 1 1])
end
%pause(0.5)



%% Inverse Dynamics and Animation

while(1)
    prompt = {'Enter Hand Angle (from the X axis):','Enter Movement
Duration Time:','Enter Object Mass in hand (kg):'};
    dlg_title = 'Input Conditions';
```

```matlab
    num_lines = 1;
    input = inputdlg(prompt,dlg_title,num_lines);

    % Input Hand Angle Degree
    deg = str2double(input{1,:});

    % Input Movement Time Duration
    time = str2double(input{2,:});

    % Input Mass in hand
    mass = str2double(input{3,:});

    % Point the Desired End Effector Position
    h = impoint;
    pos = getPosition(h);

    % Check if the point exceeds the max reach
    end_angle = atan(pos(2)/pos(1))*180./pi;
    diff = abs(end_angle -deg);
    if diff > 180
        diff = diff - 180;
    end
    while(-(diff/90)*9.25+9.25+55 < sqrt((pos(1))^2+(pos(2))^2))

        str = ['\leftarrow Exceeds Maximum Reach'];
        TXT_warn = text(pos(1),pos(2),str);
        pause(1)
        delete(TXT_warn);
        h = impoint;
        pos = getPosition(h);
    end

    % Setup desTransform
    desTransform = [ cos(deg.*pi./180) -sin(deg.*pi./180) 0
pos(1);...
                    sin(deg.*pi./180) cos(deg.*pi./180) 0 pos(2);...
                    0 0 1 0;...
                    0 0 0 1];
    % Inverse Dynamics
    [paramList, error] = dhInvKine (linkList, desTransform,
paramListG);



    % 3rd Degree Polynomial Path Planning

    a_1 = 6*(-2*(paramList(1)-paramListG(1))/(time^3));
    b_1 = -2*(-3*(paramList(1)-paramListG(1))/(time^2));
    a_2 = 6*(-2*(paramList(2)-paramListG(2))/(time^3));
    b_2 = -2*(-3*(paramList(2)-paramListG(2))/(time^2));
    a_3 = 6*(-2*(paramList(3)-paramListG(3))/(time^3));
    b_3 = -2*(-3*(paramList(3)-paramListG(3))/(time^2));
    a_4 = 6*(-2*(paramList(4)-paramListG(4))/(time^3));
    b_4 = -2*(-3*(paramList(4)-paramListG(4))/(time^2));

    theta_prev = paramListG;
    vel_prev = [0;0;0;0];


for t = 0:(time/30):time
    clf;
    theta1 = (1./6)*a_1*(t^3)+(1./2)*b_1*(t^2)+paramListG(1);
    theta2 = (1./6)*a_2*(t^3)+(1./2)*b_2*(t^2)+paramListG(2);
    theta3 = (1./6)*a_3*(t^3)+(1./2)*b_3*(t^2)+paramListG(3);
    theta4 = (1./6)*a_4*(t^3)+(1./2)*b_4*(t^2)+paramListG(4);

    theta_curr = [theta1;theta2;theta3;theta4];
    %% Velocity Calc

    vel_curr = (theta_curr - theta_prev)/(time./30);
    %% Accerleration Calc

    acc_curr = (vel_curr - vel_prev)/(time/30);
    %% Torque Calc
    baseDynamics.linA = [0;0;0];
    baseDynamics.angV = [0;0;0];
    baseDynamics.angA = [0;0;0];
    z_tor = -
mass*9.8*(link2.a*cos(theta1)+link3.a*cos(theta2)+link4.a*(theta3));
    endEffectorWrench = [0;-mass*9.8;0;0;0;z_tor];
    gravityDirection = [0;0;-1];

[jointTorques,Jv,JvDot]=newtonEuler(linkList,theta_curr,vel_curr,acc_
curr,baseDynamics,endEffectorWrench,gravityDirection);
    %% Animation Calc

    hold on;
    grid on;

    % Plot the Mass on hand
    str = ['Mass on Hand = ',num2str(mass),' kg'];
    TXT_mass = text(H(1,4,4)-5,H(2,4,4)+18,str);
    uistack(TXT_mass, 'top')
    p1 = [H(1,4,4) H(2,4,4)+2];
    p2 = [H(1,4,4) H(2,4,4)+15];
    dp = p1 - p2;
    quiver(p2(1),p2(2),dp(1),dp(2),0,'MaxHeadSize',2)

    for i = 1:1:length(theta_curr)
       H(:,:,i) = dhFwdKine(linkList(1:i), theta_curr(1:i));
    end

    for i = 1:1:length(theta_curr)-1

       %plot3([H(1,4,i) H(1,4,i+1)],[H(2,4,i) H(2,4,i+1)],[H(3,4,i)
H(3,4,i+1)])

       % Plot the arms and the hand
       PLT = plot([H(1,4,i) H(1,4,i+1)],[H(2,4,i)
H(2,4,i+1)],'LineWidth',5);
       uistack(PLT, 'bottom')

       % Plot the elbow, and wrist joints
```

```
        CIR = 
scatter( H(1,4,i+1),H(2,4,i+1),'o','LineWidth',4,'MarkerEdgeColor',[0
.5 .5],'MarkerFaceColor',[0 .5 .5]);
        uistack(CIR, 'bottom')

        % Plot the shoulder joint
        ORG = 
scatter( 0,0,'p','LineWidth',5,'LineWidth',4,'MarkerEdgeColor',[0 .5
.5],'MarkerFaceColor',[0 .5 .5]);

        % Add coordinates of the joints
        str = ['\uparrow (',num2str(H(1,4,i+1)),' ,
',num2str(H(2,4,i+1)),')'];
        TXT_coor = text(H(1,4,i+1),H(2,4,i+1)-2,str);
        uistack(TXT_coor, 'top')

        % Add coordinates of the joints
        str = [num2str(180*theta_curr(i)/pi),'\circ \rightarrow',];
        TXT_ang = text(H(1,4,i)-20,H(2,4,i),str);
        uistack(TXT_ang, 'top')

        % Display the velocity, accerleration and Torque of the joints
        str = ['Angular Velocity ',num2str(i),' =
',num2str(vel_curr(i)*180/pi),' degree/sec'];
        TXT_vel = text(-170,60-
5*i,str,'FontSize',12,'FontWeight','bold');
        uistack(TXT_vel, 'top')

        str = ['Angular Acceleration ',num2str(i),' =
',num2str(acc_curr(i)*180/pi),' degree/(sec)^2'];
        TXT_acc = text(-170,10-
5*(i),str,'FontSize',12,'FontWeight','bold');
        uistack(TXT_acc, 'top')

        str = ['Torque ',num2str(i),' =
',num2str(jointTorques(i)/100),' N-m'];
        TXT_tor = text(-170,-40-
5*(i),str,'FontSize',12,'FontWeight','bold');
```

```
            uistack(TXT_tor, 'top')

            % Set plot specs
            axis([-70 70 -70 70])
            daspect([1 1 1])
        end
        theta_prev = theta_curr;
        vel_prev = vel_curr;
        pause(time/30)

    end
             paramListG = paramList;
    end
```