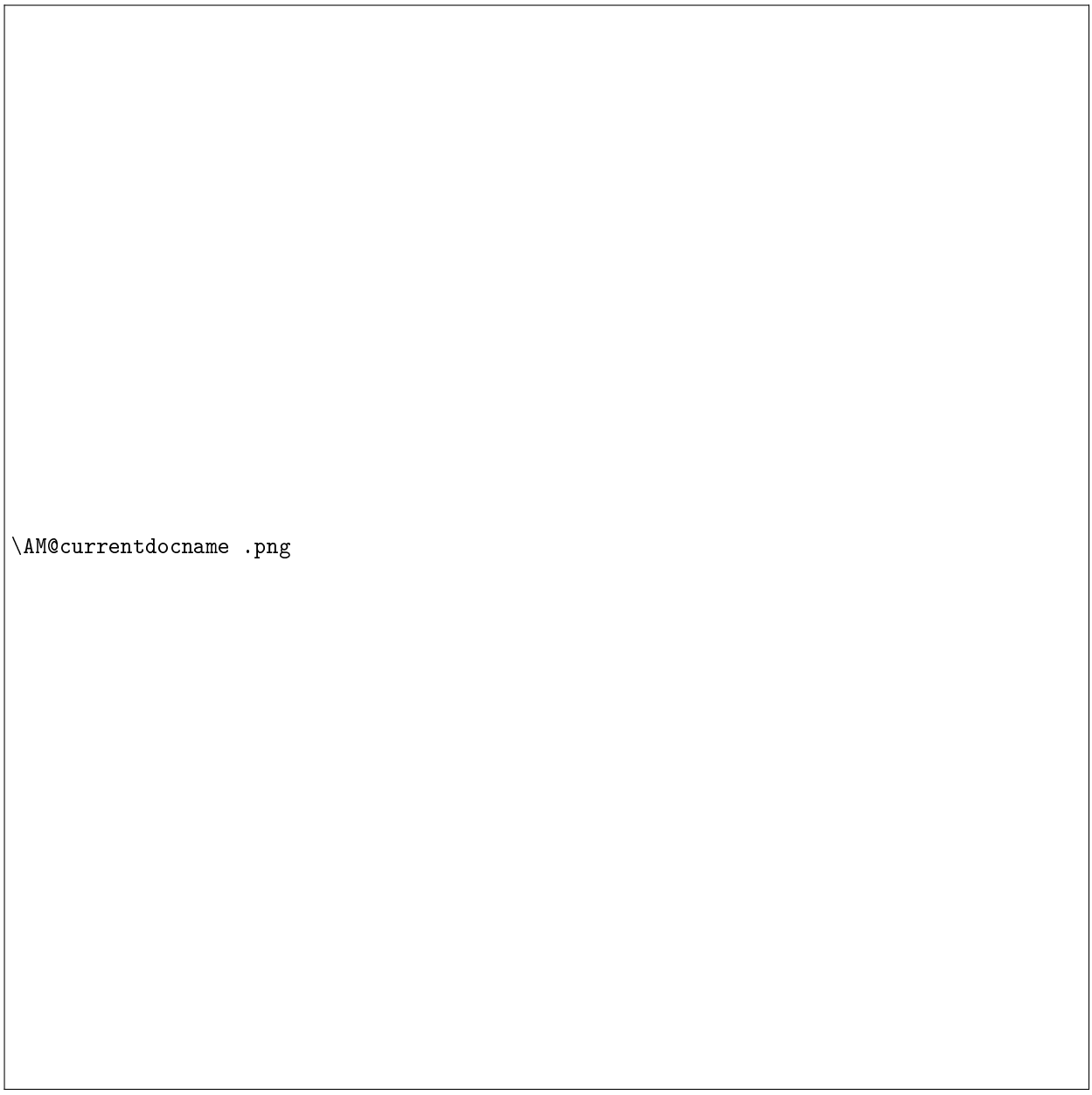

.png

4M20 - Coursework One

Candidate Number: 5584F

December 15, 2019

Abstract

Three link robots are extremely versatile and can be used in applications ranging from robotic arms to moving robots. Throughout this investigation the three link robot was tasked with catching a ball at various trajectories. Kinematic control was considered for the majority of tasks due to the decreased complexity in computation compared to a dynamic model. The robot consists of three joints to move around 2D space, this creates lots of options for path trajectory, and robot shape. Planning algorithms were used to heuristically solve this problem to find a good solution quickly.

I. Introduction

Theoretical and numeric investigations of a 3 link robotic manipulator are described in this report. The manipulator analysed is shown in **??** and is designed to catch a thrown ball.

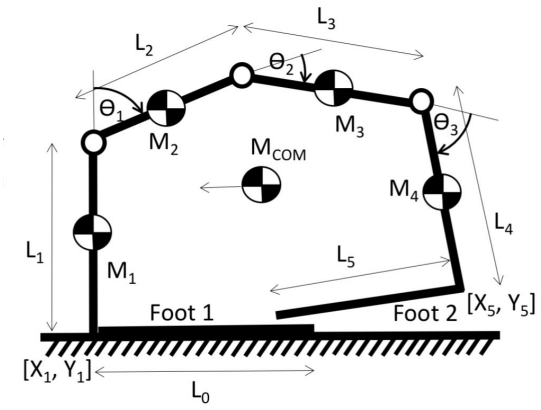

Figure 1: *Diagram of robotic manipulator. L1=0.8m, L2=0.8m, L3=0.3m, θ1=[-90,90], θ2=[-160,160], θ3=[-160,160]*

II. Question 1

i. Kinematic Control

Kinematic control involves using the system geometry and knowledge of the world without sensing to build a desired motion through time. This motion defines the joint positions in our task. Kinematic control doesn't take forces or accelerations into account. This simplification reduces the complexity of the task drastically and allows for efficient planning methods to be implemented.

Two key aspects of kinematic control are forward kinematics, and inverse kinematics. In forward kinematics the manipulator joint angles $\boldsymbol{\theta}$ are used to calculate end effector position. The reverse occurs during inverse kinematics. In our case there are three control angles, and only two position co-ordinates. This leads to multiple viable solutions to inverse kinematics, which the best is chosen using planning algorithms.

ii. Feasible Region

Figure 2: *Reachable range for the 3 link manipulator, ball trajectories are shown for questions 2 and 5*

iii. Feasible Region

Figure 3: *Viable joint angles to achieve x=y=0.5*

Figure **??** shows the reachable range for the robotic manipulator. With the constrains on angular position described in figure **??** this creates a semi circle region of radius 1.9m above y=0. The region below y=0 is also in accessible due to the floor, but is formed of two circles of radius 1.1m about [$\pm$0.8,0]. To calculate the feasible region the forward kinematics in equation **??** were used and the variables were parametrically plotted. Different areas of the feasible region are accessible with differing combinations of joint angle, the region edge is only possible for one specific set of $\boldsymbol{\theta}$ for example.

To discover viable joint angles that result in [x,y]=[0.5,0.5] the inverse kinematics are required. However, our problem involves a redundant free variable not specified in the desired position. This variable is the end orientation and is $\theta_2 + \theta_3 - \theta_1$. This means that there will exist multiple curves of solutions that satisfy the end co-ordinates. Figure **??** shows some of these solutions found by setting the desired orientation to lots of values. The solutions form a curved line, that signifies the intersection between the two complicated planes defined by the forward dynamics. Some of the solutions and resulting end co-ordinates are listed in table **??**. Figure **??** visualises some of these possible positions.

Figure 4: *Three possible joint angles that achieve x=y=0.5*

While each set of joint angles provides a valid solution to the positioning problem several factors make each solution more or less effective. This is down to factors such as required joint movement to reach end state, obstacles in trajectory, and required end orientation to catch ball. Throughout our problem no obstacles exist other then the floor which will never be in the way for a shot path so isn't considered. However, the minimisation of angle change will be considered. This directly effects both energy required and maximum required angular velocity. The l_1 norm of all required $\delta\boldsymbol{\theta}$ for the trajectory is compared to consider energy, or the l_∞ norm for reducing maximum velocity.

θ_1	θ_2	θ_3	x	y
2.377	102.5	137.9	0.4999	0.4998
43.49	127.9	37.63	0.4999	0.4992
42.58	148.3	-40.99	0.5001	0.5004
-0.1004	103.6	144.8	0.4995	0.5006

Table 1: *Some possible joint angles for [x,y]=[0.5,0.5]*

III. Question 2-3

i. Ball Dynamics

For a ball thrown without drag equations **??** can be used to generate the dynamics. The parameters take the values u=10m/s, α=45, x_0=3m, y_0=0m, g=-9.81m/s^2. This is plotted in figure **??** and is shown to just clip the reachable range at between [x,y] = [1.09,1.55] and [0.85,1.69]. To catch the ball the end effector must be in the trajectory at the time of passing. Aim to meet the ball in the middle at [x,y]=[0.956,1.634]. the time the ball passes through is calculated using equation **??** and is 0.289s.

The position of the end effector at this time is $\boldsymbol{\theta}$= [-25.65, 5.85, 8.256] , this is the solution that minimises $\Delta\boldsymbol{\theta}$ from the initial position. This solution was particularly easy to locate due to the end effectors close proximity to the feasible region edge, where θ_1, θ_2 are approximately 0. Figure **??** shows the possible solutions for this point and the chosen point that minimises the l_1 norm from the start position. Figure **??** shows the final position of the arm.

Figure 5: *Optimised position to catch the ball thrown at 45 degrees*

Figure 6: *Optimised position*

$$\begin{bmatrix} x_b(t) \\ y_b(t) \end{bmatrix} = \begin{bmatrix} -uCos(\alpha)t + x_0 \\ uSin(\alpha)t + 0.5gt^2 + y_0 \end{bmatrix} \quad (1)$$

$$t = \frac{-uSin\alpha \pm \sqrt{u^2Sin^2\alpha + 2gy}}{g} \qquad (2)$$

ii. Joint Trajectories and Planning

To calculate time series trajectories of joint angles the profile of angular speed must be considered. Figure **??** shows multiple possible profiles. Each profile assumes zero velocity at the start and end, this might not be desirable if catching required matching the ball velocity. A step function is the ideal motion to evenly move the joint. However, in practice this is impossible due to the instantaneous change in speed and acceleration delta function. To fix this third or higher order polynomial functions could be implemented. These functions guarantee smooth accelerations and velocities but they don't take into account the physical limits of motors. A trapezoid function (smoothed in real application) is therefore used to represent the movement. Figure **??** shows this velocity profile with two parameters V_{max} and a_{max} these are set by the motors used. In order to reach the desired position to catch the ball the integral of the velocity profile must equal the required angular change of each joint.
The time series equations for the velocity profile are listed in equation **??** and shown for each angle in figure **??**. The acceleration time was optimised to minimise the sum $|\ddot{\theta}_{max}T/2 + \dot{\theta}_{max}|$. These optimised acceleration and velocity maxima are listed in table **??**. For true theoretical minima of $\dot{\theta}_{max}$ and $\ddot{\theta}_{max}$ the velocity profiles are a step function, and triangular pulse respectively. If the arm can catch while moving then a ramp profile can be used to minimise acceleration further. The results for minimum $\dot{\theta}_{max}$ and $\ddot{\theta}_{max}$ for the step and ramp function can also be seen in table **??**.

$$\dot{\theta}_i = \begin{cases} -\ddot{\theta}_{max}t & : 0 < t < \frac{\dot{\theta}_{max}}{\ddot{\theta}_{max}} \\ \dot{\theta}_{max} & : \frac{\dot{\theta}_{max}}{\ddot{\theta}_{max}} < t < T - \frac{\dot{\theta}_{max}}{\ddot{\theta}_{max}} \\ \ddot{\theta}_{max}(t-T) & : T - \frac{\dot{\theta}_{max}}{\ddot{\theta}_{max}} < t < T \\ 0 & : otherwise \end{cases} \qquad (3)$$

Joint	$\lvert\dot{\theta}_{max}\rvert$	$\lvert\ddot{\theta}_{max}\rvert$
1	303.9	2870
2	623.5	5888
3	391.8	3700
1	192.6	∞
2	395	∞
3	248.2	∞
1	385	1333
2	790	2733
3	496	1718

Table 2: *Minimum $|\dot{\theta}_{max}|$ and $|\ddot{\theta}_{max}|$ when optimising a weighted sum, step function, and ramp function*

Another method of designing the angular time series is to first plan the end effector path. This can lead to minimising energy if for example the end effector is heavy, by taking the shortest route. This is achieved by dividing the desired path into small time steps and applying a trapezoid velocity profile to the path to allow smooth end movement. These small time steps result in required positions throughout time. This can be used in equation **??** to calculate joint movement in time. This equation utilises the Jacobian which is seen in equation **??**. In order to invert J($\boldsymbol{\theta}$) and determine angular movement the matrix must be square, this requires the addition of the end effector orientation (ϕ). The orientation through time can be calculated to linearly change between start and end, or to fit other criteria in different tasks.

$$\dot{\boldsymbol{\theta}}(t) = \boldsymbol{J}(\boldsymbol{\theta})^{-1}(\mathbf{X}(t+1) - \mathbf{X}(t) \qquad (4)$$

IV. Question 4

i. Dynamic Control

In dynamic control a model is build that uses forces and accelerations. For this to work we need to know the following:

- Mass and inertial matrix of each arm, link and the ball

(a) θ_1 **(b)** θ_2 **(c)** θ_3

Figure 7: *Velocity profiles for each joint, red is the unit step, blue is minimising accelerations, green is an intermediate profile and black is optimised*

- Motor torques and accelerations
- Friction at joints
- System latency

These are required so that the dynamics can be accurately modelled and turned into a time series to calculate the desired action and position. As with kinematics there exists forward and inverse dynamics. The forward model takes the torques and forces and calculates robot trajectory. The inverse model reverses this and can be used to realise desired trajectories.
A kinematic model is far less complicated then a dynamic model due to these extra specifications that aren't required. This makes computation time and ease far better in a kinematic model. However, the extra complexity comes with the benefit of improved modelling and reaction to factors such as disturbances, balance and softening landings (ball catch or walking). Damping can be added to reach targets in a smooth way, without having to design a whole velocity profile as the dynamic model knows the extremes of the motors.

V. Question 5

To catch a ball thrown at a variable angle a planning algorithm must occur. The algorithm helps to determine the required action to achieve the set goal. In order to know the goal and criteria for a good plan additional constrains must be applied to the problem.

- The ball must be caught within the feasible region, without bouncing
- The ball must be caught at zero velocity
- The path taken by the end effector should be minimised when locating the catching point
- Planning algorithm should minimise joint velocity once a desired location has been found
- Planning algorithm has knowledge of α and the thrown position

Figure 8: *Flow chart of discrete planning*

Figure **??** shows a planning algorithm for catching the ball under these constants. This algorithm splits the problem into multiple optimisations for computational ease, this won't generate the best solution but will generate a good one. The first problem is locating the interception point, this can be done using an l_2 norm from the start position to the ball path. This can be analytically computed as there aren't multiple answers. The ball enters the feasible region when α =78 at 1.8s and hits the floor at 2s. The ball never enters the feasible region when α=60 so the planning algorithm stops. Figure **??** shows these trajectories. The point on the trajectory that's closest to the start position of [x,y]=[0.4521 0.3974] from forward kinematics, is at [x,y]=[-0.9919,0.6986] the ball passes here at t=1.92s. Figure **??** shows this optimum point.

Figure 9: *Velocity profile for the joins when minimising* $|\dot{\theta}_{max}|$ *and* $|\ddot{\theta}_{max}|$

The next problem is to efficiently find a trajectory that minimises the join velocity. This can be done by assuming trapezoid velocity profiles for all the small discrete routes. Then for each small discrete step calculate the l_∞ norm of the velocities, and choose the direction that minimises this. The search through discrete space can be done using a depth first search algorithm. Once all the steps have been optimised they can be combined to form the whole joint trajectory.
By discretionary the search space you greatly simplify the problem and reduce computation

Figure 10: *Closest location to catch ball*

time, however it doesn't guarantee the best path as you cant move in all directions to find the direction of minimum joint velocity.
The desired end position was found to be $\boldsymbol{\theta}$ =[88.4947,50.8256,119.1692] this can be seen in figure **??**.
The time series trajectories of the joints follow equation **??** with $\dot{\boldsymbol{\theta}}_{max}$=[48.09,-56.87,32.2], $\ddot{\boldsymbol{\theta}}_{max}$=[68.33,-80.82,45.76], and T=1.92s.

VI. Appendix

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -L_1 Sin(\theta_1) + L_2 Sin(\theta_2 - \theta_1) + L_3 Sin(\theta_2 + \theta_3 - \theta_1) \\ L_1 Cos(\theta_1) + L_2 Cos(\theta_2 - \theta_1) + L_3 Cos(\theta_2 + \theta_3 - \theta_1) \end{bmatrix} \tag{5}$$

$$\boldsymbol{J(\theta)} = \begin{bmatrix} -l_1 cos(\theta_1) - l_2 cos(\theta_2 - \theta_1) - l_3 cos(\phi) & l_2 cos(\theta_2 - \theta_1) + l_3 cos(\phi) & l_3 cos(\phi) \\ -l_1 sin(\theta_1) + l_2 sin(\theta_2 - \theta_1) + l_3 sin(\phi) & -l_2 sin(\theta_2 - \theta_1) - l_3 sin(\phi) & -l_3 sin(\phi) \\ -1 & 1 & 1 \end{bmatrix} \tag{6}$$

i. Code

```
l1=0.8;
l2=0.8;
l3=0.3;

plot_region(l1,l2,l3);
alphas=[60,78]%,60,78];
trajectory(alphas);
% draw_position([2.377,102.5,137.9],l1,l2,l3,'k');
% draw_position([43.49,127.9,37.63],l1,l2,l3,'r');
% draw_position([42.58,148.3,-40.99],l1,l2,l3,'m');

% draw_position([-25.65,5.85,8.256],l1,l2,l3,'k');
draw_position([30,120,80],l1,l2,l3,'r');
draw_position([88.4947,50.8256,119.1692],l1,l2,l3,'k');

title('');
xlabel('x/m');
ylabel('y/m');
hold on
scatter3(-0.9919,0.6986,0)
function plot_region(l1,l2,l3)
    syms t1 t2
    for t3 = -160:10:160
        x=-l1*sind(t1)+l2*sind(-t1+t2)+l3*sind(-t1+t2+t3);
        y=l1*cosd(t1)+l2*cosd(-t1+t2)+l3*cosd(-t1+t2+t3);
        z=x*0;
        h=ezsurf(x,y,z,[-90,90,-160,160]);
        h.EdgeColor = 'none';
        hold on
    end
end


function trajectory(alphas)
    g= 9.81;
    u=10;
    x0=3;
```

```
    y0=0;
    colours=['m','g','r','k']
    syms t
    for i =1:length(alphas)
        hold on
        alpha=alphas(i)
        x=-u*cosd(alpha)*t+x0;
        y=u*sind(alpha)*t+0.5*-g*t^2+y0;
        h=ezplot(x,y,[0,2.037*sind(alpha)]);
        h.Color = colours(i)
    end
    title('Feasible␣Region␣&␣Trajectories');
    xlabel('x/m');
    ylabel('y/m');
    %legend('alpha=45','alpha=60','alpha=78');
end



function draw_position(t,l1,l2,l3,c)

    x1=-l1*sind(t(1));
    y1=l1*cosd(t(1));

    x2=-l1*sind(t(1))+l2*sind(-t(1)+t(2));
    y2=l1*cosd(t(1))+l2*cosd(-t(1)+t(2));

    x3=-l1*sind(t(1))+l2*sind(-t(1)+t(2))+l3*sind(-t(1)+t(2)+t(3));
    y3=l1*cosd(t(1))+l2*cosd(-t(1)+t(2))+l3*cosd(-t(1)+t(2)+t(3));

    line([0 x1],[0 y1],[0 0],'color',c,'LineWidth',2);
    line([x1 x2],[y1 y2],[0 0],'color',c,'LineWidth',2);
    line([x2 x3],[y2 y3],[0 0],'color',c,'LineWidth',2);
end
```

```
x0=-0.9919;
y0=0.6986;
l1=0.8;
l2=0.8;
l3=0.3;

xs(1)=0;
ys(1)=0;
zs(1)=0;

count=1;
for phi = 0:0.25:360
```

```
    syms x y z
    Eq1 = -l1*sind(x)+l2*sind(y-x)+l3*sind(y+z-x) == x0 ;
    Eq2 = l1*cosd(x)+l2*cosd(y-x)+l3*cosd(y+z-x) == y0 ;
    Eq3 = y + z - x == phi;

    result = vpasolve(Eq1,Eq2,Eq3);
    s = size(result.x);
    for sol = 1:s(1)
        tempx =result.x(sol);
        tempy =result.y(sol);
        tempz =result.z(sol);
        if abs(tempx)<= 90 & abs(tempy)<=160 & abs(tempz)<=160
            xs(count)=tempx;
            ys(count)=tempy;
            zs(count)=tempz;
            count=count+1
        end

    end
end

scatter3(xs,ys,zs);
```