Contents


```
function [ dx ] = ode1linkTracking_passitvity_robust( t,x,param )
```

Implementation

```
        %Extracting the coefficients of the trajectory
        a1=param(1,:);

        %Extract the approximated dynamics
        alpha=x(3:5,:);

        %Create the actual trajectory
        vec_t = [1; t; t^2; t^3];
        theta_d= [a1*vec_t];

        % compute the velocity and acceleration in both theta 1 and theta2.
        a1_vel = [a1(2), 2*a1(3), 3*a1(4), 0];
        a1_acc = [2*a1(3), 6*a1(4),0,0 ];


        % compute the desired trajectory (assuming 3rd order polynomials for trajectories)
        dtheta_d =[a1_vel*vec_t];
        ddtheta_d =[a1_acc*vec_t];
        theta= x(1);
        dtheta= x(2);
```

```
Not enough input arguments.

Error in ode1linkTracking_passitvity_robust (line 5)
        a1=param(1,:);
```

Passivity Robust controller

True dynamics of the system

```
        I=7.2;
        mgd=1;
        fv=1;
        alpha_actual=[mgd;fv;I];

        % Calculating the r in the system
        lambda=1;
        e=theta-theta_d;
        e_dot=dtheta-dtheta_d;
        r=e_dot+lambda*e;
```

```
        v=dtheta_d-r;
        a=ddtheta_d-r;

       %Calculate the dalpha
        gamma=[1 0 0;0 1 0;0 0 1];
        P=[1 0;0 1];
        phi=[sin(theta_d) v a];

        Kv=1;
        eps=5;
        rho=10;
        if(norm(transpose(phi)*r)>eps);
          dalpha=-rho*(transpose(phi)*r)/norm(transpose(phi)*r);
        else
           dalpha=-rho*(transpose(phi)*r)/eps;
        end


        u=phi*(dalpha-alpha_actual)-Kv*r;

        ddtheta=(1/I)*(u-fv*x(2)-mgd*sin(x(1)));
```

Calculate dx based on the dynamics of the robot and the error

```
          dx=[dtheta;ddtheta;dalpha];
```

```
end
```

Published with MATLAB® R2018b