```
classdef manipulator < handle

    properties
        DH          %DH parameters that the user inputs
        A_Mats      %3D array to hold all the A matrices
        Trans       %General transformation matrix for manipulator
        numJoints   %Number of joints in the system
        config      %Type of joints and their order from base to EE
        sing        %singularities
        links       %link lengths

    end

    methods

        %Constructor
        function self = manipulator(DH,l)

            self.DH = DH;
            self.numJoints = size(self.DH,1);
            self.links = l;
            self.Trans = self.fkine();

        end

        %Prints all the current manipulator parameters
        function currentProperties(self)

            self.DH
            self.A_Mats
            self.Trans
            self.numJoints
            self.config

        end

        %This function calculates the forward kinematics of the robotic
        %manipulator and return the general tranformation matrix or the
        %transformation matrix for a given set of joint positions and link
        %lengths
        function Trans = fkine(self,q)
            %q is an optional arguments
            arguments
                self
                q = []
            end
            %Needs to be here so we can use subs function later
            syms l1 l2 l3
```

```
            if isempty(self.Trans)

                numJoints = self.numJoints;
                %Have to setup array with syms otherwise MATLAB will complain about ↙
using
                %numeric vs variables
                A_Mats = sym(zeros(4,4,numJoints));



                for index = 1:numJoints
                    %User must define the DH table for a given robot
                    a = self.DH(index,1);
                    d = self.DH(index,2);
                    alpha = self.DH(index,3);
                    theta = self.DH(index,4);

                    % Build the A_i matrix
                    A_Mats(:,:,index) = createMatrix(a,d,alpha,theta);
                end



                self.A_Mats = A_Mats(:,:,:);


                % Multiply all transformation matrices to get the end effector
                % frame with respect to the base frame
                Trans = eye(4);
                for index = 1:numJoints
                    Trans = Trans * A_Mats(:,:,index);
                end

                sympref('AbbreviateOutput', false);
                %Simplifies the transformation matrix
                Trans = simplify(Trans,"Steps",400);
                %This removes the pi/180 that appears when you just use the simplify
                %function to display
                %Trans = subs(Trans,pi/180,1);
                Trans = subs(Trans, {l1, l2, l3}, {self.links(1,1), self.links(1,2), ↙
self.links(1,3)});
                %Saving the general transformation matrix as a property of the
                %manipulator
                self.Trans = Trans;
```

```
            else %If Trans has already been calculated
                Trans = self.Trans;
            end

            %If the variable for q isn't empty, then sub them into the
            %transformation matrix
            if ~isempty(q)
                vars = symvar(Trans);
                q_index = 1;
                l_index = 1;
                %Loop through all the unknowns in the transformation matrix
                for index = 1:length(vars)
                    %If the varible name doesn't start with l
                    if ~startsWith(string(vars(index)), "l")
                        Trans = subs(Trans,vars(index),q(q_index));
                        q_index = q_index + 1;
                    %If the variable name does start with l
                    else
                        Trans = subs(Trans,vars(index),self.links(l_index));
                        l_index = l_index + 1;
                    end
                end
            end

        end
        %Given a position q, return necessary joint values to achieve
        %that position
        function [d1, theta2, theta3] = ikine(self, q)
            % q = [x y z] (desired EE position in base frame)
            x = q(1); y = q(2); z = q(3);
            l1 = self.links(1);
            l2 = self.links(2);
            l3 = self.links(3);

            d1 = z;                 % prismatic first joint


            D = (y^2 + (x-l1)^2 - l2^2 - l3^2)/(2*l2*l3);
            theta3_minus = atan2(-sqrt(1-D^2),D);
            theta3_plus = atan2(sqrt(1-D^2),D);

            theta3 = [theta3_plus theta3_minus];

            %Changed the plus theta2 to addition and it makes more sense now
            theta2_minus = atan2(y,x-l1) - atan2(l3*sin(theta3_minus),l2+l3*cos ↙
(theta3_minus));
            %theta2_plus_test = atan((l2+l3*cos(theta3_plus))/(l3*sin(theta3_plus))) ↙
- atan((x-l1)/y)
            theta2_plus = atan2(y,(x-l1)) - atan2(l3*sin(theta3_plus),l2+l3*cos ↙
```

```
(theta3_plus));

            theta2 = [theta2_plus theta2_minus];
        end


        function J = Jacobian(self,q)
            %q is an optional argument
            arguments
                self
                q = []
            end
            %Needs to be here so we can use subs function later
            syms l1 l2 l3 d1 theta2 theta3

            %This assumes that the user has already run fkine
            numJoints = self.numJoints;
            A_Mats = self.A_Mats;
            links = self.links;

            %This identifies the configuration based on the unknowns in the
            %transformation matrix
            vars = symvar(self.Trans);
            var_index = 1;
            self.config = strings(1,numJoints); %preallocate as string array

            %Loop through all the unknowns in the transformation matrix
            for index = 1:length(vars)
                %If the varible name doesn't start with l
                if startsWith(string(vars(index)), "theta")
                    self.config(var_index) = "R";
                    var_index = var_index + 1;
                elseif startsWith(string(vars(index)),"d")
                    self.config(var_index) = "P";
                    var_index = var_index + 1;
                end
            end

            % Preallocate
            z = sym(zeros(3,numJoints+1));
            O = sym(zeros(3,numJoints+1));
            T = sym(eye(4));

            % Base frame
            z(:,1) = [0;0;1];
            O(:,1) = [0;0;0];
            self.A_Mats(:,:,1);
            % Compute cumulative transforms
            for i = 1:numJoints
```

```
                T = T * A_Mats(:,:,i);
                z(:,i+1) = T(1:3,3);
                O(:,i+1) = T(1:3,4);
            end

            % Build Jacobian
            J = sym(zeros(6,numJoints));
            for i = 1:numJoints
                if self.config(i) == "R"

                    Jv = cross(z(:,i), O(:,end) - O(:,i));
                    Jw = z(:,i);
                    J(:,i) = [Jv; Jw];

                elseif self.config(i) == "P"

                    Jv = z(:,i);
                    Jw = sym([0;0;0]); %Using syn incase Jv is symbolic
                    J(:,i) = [Jv; Jw];

                else

                    disp(["Issue with configuration:", self.config]);

                end

            end

            J = simplify(J);
            %Substituting values for links into the jacobian
            J = subs(J, {l1 l2 l3}, {links(1) links(2) links(3)});
            %If values for q were passed, sub them into the jacobian
            if ~isempty(q)

                J = subs(J, {d1, theta2, theta3}, {q(1) q(2) q(3)});

            end
            %Don't need these atm
            %{
            Singularity = simplify(det(J));
            subs(Singularity,pi/180,1);
            fprintf("Determinant of J: %s\n", Singularity);
            %}
        end
    end
end

%General form of the homgeneous transformation matrix
function A = createMatrix(a,d,alpha,theta)
```

```
    A = [cos(theta) -sin(theta)*cos(alpha)  sin(theta)*sin(alpha)  a*cos(theta);
         sin(theta)  cos(theta)*cos(alpha) -cos(theta)*sin(alpha)  a*sin(theta);
         0           sin(alpha)             cos(alpha)             d;
         0           0                      0                      1];
end
```