```matlab
%% prob_3c.m
%
% this is a script to plot the valid coniguration space of 2-DoF robot arm
%
% - written by: Dimitri Lezcano

%% Set-Up
global po r l1 l2

% run params
check_fwd_kin = false;
check_constraints = false;


% physical parameters
l1 = 1;
l2 = 1;
po = 1/2*[1;1];
r = 1/4;


% possible angles to iterate over
N_angles = 150;
theta1_v = linspace(0, 2*pi, N_angles);
theta2_v = linspace(0, 2*pi, N_angles);


%% Determine which angles to keep
% part a
theta_valid_a = zeros(2, N_angles^2);
num_valid_configs_a = 0;

% part b
theta_valid_b = zeros(2, N_angles^2);
num_valid_configs_b = 0;
for theta1 = theta1_v
    for theta2 = theta2_v
        % check if valid configuration: part a
        c_a = constrainta(theta1, theta2);
        if (c_a <= 0)
            % increment num_valid_configs
            num_valid_configs_a = num_valid_configs_a + 1;

            % add [theta1; theta2] to theta_valid
            theta_valid_a(:, num_valid_configs_a) = [theta1; theta2];

        end

        % check if valid configuration: part b
        c_b = constraintb(theta1, theta2);
        if all(c_b <= 0)
            % increment num_valid_configs
            num_valid_configs_b = num_valid_configs_b + 1;

            % add [theta1; theta2] to theta_valid
            theta_valid_b(:, num_valid_configs_b) = [theta1; theta2];

        end
    end
end

% remove unused points
theta_valid_a = theta_valid_a(:,1:num_valid_configs_a);
theta_valid_b = theta_valid_b(:,1:num_valid_configs_b);

%% Plotting
% plot the points: part b
fb = figure(1);
theta_valid_deg_b = rad2deg(theta_valid_b);
```

```matlab
plot(theta_valid_deg_b(1,:), theta_valid_deg_b(2,:), '.', 'MarkerSize', 10);
xlabel('theta1 (deg)'); ylabel('theta2 (deg)');
xlim([0, 360]); ylim([0, 360]);
title('Problem 3.c | Part b)')
grid on;

% plot the points: part a
fa = figure(2);
theta_valid_deg_a = rad2deg(theta_valid_a);
plot(theta_valid_deg_a(1,:), theta_valid_deg_a(2,:), '.', 'MarkerSize', 10);
xlabel('theta1 (deg)'); ylabel('theta2 (deg)');
xlim([0, 360]); ylim([0, 360]);
title('Problem 3.c | Part a)')
grid on;

% check the forward kinematics
if check_fwd_kin
    theta1 = pi/4; theta2 = 0;
    p1 = calculate_joint1(theta1, theta2);
    p2 = calculate_tool(theta1, theta2);
    p = [zeros(2,1) p1 p2];

    figure(4);
    plot(p(1,:), p(2,:),'.-'); hold on;
    % check the constraints
    if check_constraints
        pts_obs = plot_obstacle_pts(po, r);
        plot(pts_obs(1,:), pts_obs(2,:), 'r'); hold off;
        disp('constraint');
        disp(constraintb(theta1, theta2));

    end
    hold off;
    xlim([-(l1 + l2 + 1) (1 + l1 + l2)])
    ylim([-(l1 + l2+ 1) (1 + l1 + l2)])
    grid on;
end


%% Saving
% part a figure
saveas(fa, 'prob_3c-a.png');
disp('Saved: prob_3c-a.png');

% part b figure
saveas(fb, 'prob_3c-b.png');
disp('Saved: prob_3c-b.png');

%% Functions
% implementation of the constraint functin: Part a
function c = constrainta(theta1, theta2)
    global po r
    p_t = calculate_tool(theta1, theta2); % tool position

    c = r^2 - (p_t - po)'*(p_t - po);


end
% implementation of the constraint function: Part b
function c = constraintb(theta1, theta2)
    global po r
    % get the joint position
    p_base = zeros(2, 1);
    p_jt1 = calculate_joint1(theta1, theta2);
    p_t = calculate_tool(theta1, theta2);

    % get the lambda conditions
    % joint 1
    A1 = calculate_A(po, p_base, p_jt1, r);
```

```matlab
    B1 = calculate_B(po, p_base, p_jt1, r);
    C1 = calculate_C(po, p_base, p_jt1, r);

    % joint2
    A2 = calculate_A(po, p_jt1, p_t, r);
    B2 = calculate_B(po, p_jt1, p_t, r);
    C2 = calculate_C(po, p_jt1, p_t, r);

    % perform analysis
    % discriminant
    discrim1 = B1^2 - 4 * A1 * C1;
    discrim2 = B2^2 - 4 * A2 * C2;

    c = zeros(4,1);
    % joint1
    if discrim1 <= 0
        c(1:2) = discrim1;

    else % 2 real roots
        lambda_p = (-B1 + sqrt(discrim1))/(2*A1);
        lambda_m = (-B1 - sqrt(discrim1))/(2*A1);
        c(1) = (-B1 + sqrt(discrim1))*(2*A1 + B1 - sqrt(discrim1));
        c(2) = (-B1 - sqrt(discrim1))*(2*A1 + B1 + sqrt(discrim1));
    end

    % joint 2
    if discrim2 <= 0
        c(3:4) = discrim2;

    else % 2 real roots
        lambda_p = (-B2 + sqrt(discrim2))/(2*A2);
        lambda_m = (-B2 - sqrt(discrim2))/(2*A2);
        c(3) = (-B2 + sqrt(discrim2))*(2*A2 + B2 - sqrt(discrim2));
        c(4) = (-B2 - sqrt(discrim2))*(2*A2 + B2 + sqrt(discrim2));
    end


end

% 2x2 rotation matrix
function R = rotate(theta)
    R = [cos(theta), -sin(theta); sin(theta), cos(theta)];

end

% calculate A value for lambda root
function A = calculate_A(p0, p1, p2, r)
    A = (p2 - p1)'* (p2 - p1);

end

% calculate B value for lambda root
function B = calculate_B(p0, p1, p2, r)
    B = 2*(p1'*p2 - p2'*p2 - p0'*p1 + p0'*p2);

end

% calculate C value for lambda root
function C = calculate_C(p0, p1, p2, r)
    C = (p0 - p2)'*(p0 - p2) - r^2;

end

% calculate joint 1 position
function p = calculate_joint1(theta1, theta2)
    global l1 l2

    p = l1 * rotate(theta1) * [1; 0];

```

```
end

% calculate tool position
function p = calculate_tool(theta1, theta2)
    global l1 l2

    p1 = calculate_joint1(theta1, theta2);
    p = p1 + l2 * rotate(theta1 + theta2) * [1; 0];


end

% obstacle plot points
function pts = plot_obstacle_pts(po, r)
    theta = linspace(0, 2*pi, 100);
    pts = po + r * [cos(theta); sin(theta)];

end
```