

```

1  %% prob_3c.m
2  %
3  % this is a script to plot the valid configuration space of 2-DoF robot arm
4  %
5  % - written by: Dimitri Lezcano
6
7  %% Set-Up
8  global po r l1 l2
9
10 % run params
11 check_fwd_kin = false;
12 check_constraints = false;
13
14
15 % physical parameters
16 l1 = 1;
17 l2 = 1;
18 po = 1/2*[1;1];
19 r = 1/4;
20
21
22 % possible angles to iterate over
23 N_angles = 150;
24 theta1_v = linspace(0, 2*pi, N_angles);
25 theta2_v = linspace(0, 2*pi, N_angles);
26
27
28 %% Determine which angles to keep
29 % part a
30 theta_valid_a = zeros(2, N_angles^2);
31 num_valid_configs_a = 0;
32
33 % part b
34 theta_valid_b = zeros(2, N_angles^2);
35 num_valid_configs_b = 0;
36 for theta1 = theta1_v
37     for theta2 = theta2_v
38         % check if valid configuration: part a
39         c_a = constrainta(theta1, theta2);
40         if (c_a <= 0)
41             % increment_num_valid_configs
42             num_valid_configs_a = num_valid_configs_a + 1;
43
44             % add [theta1; theta2] to theta_valid
45             theta_valid_a(:, num_valid_configs_a) = [theta1; theta2];
46
47         end
48
49         % check if valid configuration: part b
50         c_b = constraintb(theta1, theta2);
51         if all(c_b <= 0)
52             % increment num_valid_configs
53             num_valid_configs_b = num_valid_configs_b + 1;
54
55             % add [theta1; theta2] to theta_valid
56             theta_valid_b(:, num_valid_configs_b) = [theta1; theta2];
57
58         end
59     end
60 end
61
62 % remove unused points
63 theta_valid_a = theta_valid_a(:, 1:num_valid_configs_a);
64 theta_valid_b = theta_valid_b(:, 1:num_valid_configs_b);
65
66 %% Plotting
67 % plot the points: part b
68 fb = figure(1);
69 theta_valid_deg_b = rad2deg(theta_valid_b);

```

```

70 plot(theta_valid_deg_b(1,:), theta_valid_deg_b(2,:), '.', 'MarkerSize', 10);
71 xlabel('theta1 (deg)'); ylabel('theta2 (deg)');
72 xlim([0, 360]); ylim([0, 360]);
73 title('Problem 3.c | Part b)')
74 grid on;
75
76 % plot the points: part a
77 fa = figure(2);
78 theta_valid_deg_a = rad2deg(theta_valid_a);
79 plot(theta_valid_deg_a(1,:), theta_valid_deg_a(2,:), '.', 'MarkerSize', 10);
80 xlabel('theta1 (deg)'); ylabel('theta2 (deg)');
81 xlim([0, 360]); ylim([0, 360]);
82 title('Problem 3.c | Part a)')
83 grid on;
84
85 % check the forward kinematics
86 if check_fwd_kin
87     theta1 = pi/4; theta2 = 0;
88     p1 = calculate_joint1(theta1, theta2);
89     p2 = calculate_tool(theta1, theta2);
90     p = [zeros(2,1) p1 p2];
91
92     figure(4);
93     plot(p(1,:), p(2,:), '-'); hold on;
94     % check the constraints
95     if check_constraints
96         pts_obs = plot_obstacle_pts(po, r);
97         plot(pts_obs(1,:), pts_obs(2,:), 'r'); hold off;
98         disp('constraint');
99         disp(constraintb(theta1, theta2));
100
101     end
102     hold off;
103     xlim([- (l1 + l2 + 1) (1 + l1 + l2)]);
104     ylim([- (l1 + l2 + 1) (1 + l1 + l2)]);
105     grid on;
106 end
107
108
109 %% Saving
110 % part a figure
111 saveas(fa, 'prob_3c-a.png');
112 disp('Saved: prob_3c-a.png');
113
114 % part b figure
115 saveas(fb, 'prob_3c-b.png');
116 disp('Saved: prob_3c-b.png');
117
118 %% Functions
119 % implementation of the constraint function: Part a
120 function c = constrainta(theta1, theta2)
121     global po r
122     p_t = calculate_tool(theta1, theta2); % tool position
123
124     c = r^2 - (p_t - po)'*(p_t - po);
125
126
127 end
128 % implementation of the constraint function: Part b
129 function c = constraintb(theta1, theta2)
130     global po r
131     % get the joint position
132     p_base = zeros(2, 1);
133     p_jt1 = calculate_joint1(theta1, theta2);
134     p_t = calculate_tool(theta1, theta2);
135
136     % get the lambda conditions
137     % joint 1
138     A1 = calculate_A(po, p_base, p_jt1, r);

```

```

139     B1 = calculate_B(po, p_base, p_jt1, r);
140     C1 = calculate_C(po, p_base, p_jt1, r);
141
142     % joint2
143     A2 = calculate_A(po, p_jt1, p_t, r);
144     B2 = calculate_B(po, p_jt1, p_t, r);
145     C2 = calculate_C(po, p_jt1, p_t, r);
146
147     % perform analysis
148     % discriminant
149     discrim1 = B1^2 - 4 * A1 * C1;
150     discrim2 = B2^2 - 4 * A2 * C2;
151
152     c = zeros(4,1);
153     % joint1
154     if discrim1 <= 0
155         c(1:2) = discrim1;
156
157     else % 2 real roots
158         lambda_p = (-B1 + sqrt(discrim1))/(2*A1);
159         lambda_m = (-B1 - sqrt(discrim1))/(2*A1);
160         c(1) = (-B1 + sqrt(discrim1))*(2*A1 + B1 - sqrt(discrim1));
161         c(2) = (-B1 - sqrt(discrim1))*(2*A1 + B1 + sqrt(discrim1));
162     end
163
164     % joint 2
165     if discrim2 <= 0
166         c(3:4) = discrim2;
167
168     else % 2 real roots
169         lambda_p = (-B2 + sqrt(discrim2))/(2*A2);
170         lambda_m = (-B2 - sqrt(discrim2))/(2*A2);
171         c(3) = (-B2 + sqrt(discrim2))*(2*A2 + B2 - sqrt(discrim2));
172         c(4) = (-B2 - sqrt(discrim2))*(2*A2 + B2 + sqrt(discrim2));
173     end
174
175
176 end
177
178 % 2x2 rotation matrix
179 function R = rotate(theta)
180     R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
181
182 end
183
184 % calculate A value for lambda root
185 function A = calculate_A(p0, p1, p2, r)
186     A = (p2 - p1)' * (p2 - p1);
187
188 end
189
190 % calculate B value for lambda root
191 function B = calculate_B(p0, p1, p2, r)
192     B = 2*(p1'*p2 - p2'*p2 - p0'*p1 + p0'*p2);
193
194 end
195
196 % calculate C value for lambda root
197 function C = calculate_C(p0, p1, p2, r)
198     C = (p0 - p2)' * (p0 - p2) - r^2;
199
200 end
201
202 % calculate joint 1 position
203 function p = calculate_joint1(theta1, theta2)
204     global l1 l2
205
206     p = l1 * rotate(theta1) * [1; 0];
207

```

```

208 end
209
210 % calculate tool position
211 function p = calculate_tool(theta1, theta2)
212     global l1 l2
213
214     p1 = calculate_joint1(theta1, theta2);
215     p = p1 + l2 * rotate(theta1 + theta2) * [1; 0];
216
217
218 end
219
220 % obstacle plot points
221 function pts = plot_obstacle_pts(po, r)
222     theta = linspace(0, 2*pi, 100);
223     pts = po + r * [cos(theta); sin(theta)];
224
225 end
226
227
228

```