Robot of choice:

UFACTORY 850 Robotic Arm (6 DoF)

Joint Range/Limits:

Joint	Range
J1	$\pm 360°$
J2	$\pm 132°$
J3	$-242°\sim 3.5°$
J4	$\pm 360°$
J5	$\pm 124°$
J6	$\pm 360°$

D-H Table:

Kinematics	θ	d(mm)	α(deg)	a(mm)
Joint1	0	364	90	0
Joint2	90	0	180	390
Joint3	90	0	-90	150
Joint4	0	426	-90	0
Joint5	0	0	90	0
Joint6	0	90	0	0

Equations:

$$Sphere\ Center: < h, k, l >$$

$$Sphere\ Radius: R$$

$$Circle\ Radius: r$$

Sphere equation: $(x-h)^2 + (y-k)^2 + (z-l)^2 = R^2$

Normal to sphere: $[2 * (x-h),\ 2 * (y-k),\ 2 * (z-l)] = End\ effector\ orientation$

Distance into sphere: $\sqrt{R^2 - r^2}$

Parameterized circle: $= \left[(rcos\theta + rsin\theta) + \left(R - \sqrt{R^2 - r^2}\right)\right] \times unitNormal\ vector\ : 0 \le \theta \le 2\pi$

```matlab
%% --- Robot Model Implementation and FK Verfication ---
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Robot Model Implementation and FK Verfication...');

% constant for unit conversion
deg = pi/180;

% define links using DH params for UFACTORY 850
L(1) = Link([0,  0.364, 0,     90*deg],  'offset', 0,      'R');
L(2) = Link([0,  0,     0.390, 0], 'offset', 90*deg, 'R');
L(3) = Link([0,  0,     0.150, -90*deg], 'offset', 90*deg, 'R');
L(4) = Link([0,  0.426, 0,     90*deg], 'offset', 0,      'R');
L(5) = Link([0,  0,     0,     -90*deg],  'offset', 0,      'R');
L(6) = Link([0,  0.090, 0,     0],       'offset', 0,      'R');

% apply joint limits
L(1).qlim = [-360*deg, 360*deg];
L(2).qlim = [-132*deg, 132*deg];
L(3).qlim = [-242*deg, 3.5*deg];
L(4).qlim = [-360*deg, 360*deg];
L(5).qlim = [-124*deg, 124*deg];
L(6).qlim = [-360*deg, 360*deg];

% create SerialLink object
robot = SerialLink(L, 'name', 'UFACTORY 850');

% verify FK
q_test = zeros(1,6);

T_fk_object = robot.fkine(q_test);
T_fk_matrix = T_fk_object.T;

disp('Forward Kinematics Verification at q = [0, 0, 0, 0, 0, 0]:');
disp('End-Effector Pose (T_fk):');
disp(T_fk_matrix);

position_xyz = T_fk_matrix(1:3, 4)';
fprintf('End-Effector Position (x,y,z) in m: [%.4f, %.4f, %.4f]\n', position_xyz);

%% --- Trajectory Planning ---
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Trajectory Planning...');

%sphere params in meters
sphere_center = [-0.6; -0.4; 0.8];
sphere_radius = 0.15;

%circle params
circle_normal = [1; 1; 0.45];
circle_normal = circle_normal / norm(circle_normal);
circle_radius = 0.04;

plane_offset = sqrt(sphere_radius^2 - circle_radius^2);%sphere_radius * 0.5;

%calculations
```

```
circle_center = sphere_center + plane_offset * circle_normal;

%make 2 orthogonal basis vectors (u & v) for circle plane
if abs(circle_normal(1)) < 0.9
    temp = [1; 0; 0];
else
    temp = [0; 1; 0];
end
u = cross(circle_normal, temp);
u = u / norm(u);
v = cross(circle_normal, u);

num_points = 100;
time_parameterization = linspace(0, 2*pi, num_points);

% trajectory generation
positions = zeros(3, num_points);
orientations = zeros(3, 3, num_points);

for i = 1:num_points
    %calculate position on circle
    positions(:, i) = circle_center + circle_radius * cos(time_parameterization(i)) *
u + circle_radius * sin(time_parameterization(i)) * v;

    %calculate normal vector
    normal_vector = positions(:,i ) - sphere_center;
    normal_vector = normal_vector / norm(normal_vector);

    %z-axis pointing to sphere center
    z_axis = -normal_vector;

    %x-axis perpendicular to normal and circle normal
    x_axis = cross(circle_normal, z_axis);
    if norm(x_axis) < 1e-6
        x_axis = cross([1; 0; 0], z_axis);
        if norm(x_axis) < 1e-6
            x_axis = cross([0; 1; 0], z_axis);
        end
    end
    x_axis = x_axis / norm(x_axis);

    %y-axis complete right handed coordinate system
    y_axis = cross(z_axis, x_axis);

    %store orientation matrix
    orientations(:, :, i) = [x_axis, y_axis, z_axis];
end

fprintf('========================================\n');
fprintf('Sphere Center: [%.3f, %.3f, %.3f] m\n', sphere_center);
fprintf('Sphere Radius: %.3f m\n', sphere_radius);
fprintf('Circle Center: [%.3f, %.3f, %.3f] m\n', circle_center);
fprintf('Circle Radius: %.3f m\n', circle_radius);
fprintf('Circle Normal: [%.3f, %.3f, %.3f]\n', circle_normal);
```

```matlab
fprintf('Num Trajectory Points: %d\n', num_points);
fprintf('======================================\n\n');

%% --- Inverse Kinematics ---
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Inverse Kinematics...');

%preallocate joint angles arr
q_trajectory = zeros(num_points, 6);

%inital guess for IK
q0 = [0, -45*deg, -45*deg, 0, -45*deg, 0];

%solve IK for each trajectory pt
for i = 1:num_points
    %create homogeneous transformation matrix
    T_desired = [orientations(:,:,i), positions(:,i); 0 0 0 1];

    %solve IK using ikine w/ mask
    q_sol = robot.ikine(T_desired, q0, 'mask', [1 1 1 1 1 1]);

    %check if solution is within joint lims
    if isempty(q_sol) || any(isnan(q_sol))
        fprintf('IK failed at point %d. using previous solution.', i);
        if i > 1
            q_sol = q_trajectory(i-1, :);
        else
            q_sol = q0;
        end
    end

    %store solution
    q_trajectory(i,:) = q_sol;

    %use curr solution as init guess for next iteration
    q0 = q_sol;

end

disp('\nInverse Kinematics Complete');

%verify joint lims
fprintf('======================================\n');
disp('Joint limit verification:');
for j = 1:6
    q_min = min(q_trajectory(:, j));
    q_max = max(q_trajectory(:, j));
    limit_min = L(j).qlim(1);
    limit_max = L(j).qlim(2);

    fprintf('Joint %d: [%.2f, %.2f] deg | Limits: [%.2f, %.2f] deg\n', j, q_min/deg, 
q_max/deg, limit_min/deg, limit_max/deg);

    if q_min < limit_min || q_max > limit_max
        fprintf('Joint %d exceeds limits', j);
```

```
    end
end
fprintf('======================================\n\n');


%% --- Animation ---
%%%%%%%%%%%%%%%%%%%%
disp('Animation...');

figure('Name', 'Robot Drawing Circle on Sphere', 'NumberTitle', 'off');
hold on;
grid on;
axis equal;
xlabel('X');
ylabel('Y');
zlabel('Z');
title('UFACTORY xARM 850 - Circular Path on Sphere Surface');

%set axis limits for better visualization
axis([sphere_center(1)-0.6, sphere_center(1)+0.6, sphere_center(2)-0.6,
sphere_center(2)+0.6, 0, sphere_center(3)+0.6]);

%draw sphere
[x_sphere, y_sphere, z_sphere] = sphere(30);
surf(sphere_center(1) + sphere_radius * x_sphere, sphere_center(2) + sphere_radius *
y_sphere, sphere_center(3) + sphere_radius * z_sphere, 'FaceAlpha', 0.3, 'EdgeAlpha',
0.1, 'FaceColor', 'red');

%draw desired circle path
plot3(positions(1,:), positions(2,:), positions(3,:), 'r--', 'LineWidth', 2,
'DisplayName', 'Desired Path');

%actual path
trace_line = plot3(nan, nan, nan, 'b-', 'LineWidth', 2, 'DisplayName', 'Actual
Path');

%%legend('Location', 'northeast');

%animation loop
for i = 1:num_points
    %plot robot at current config
    robot.plot(q_trajectory(i,:), 'workspace', [sphere_center(1)-1,
sphere_center(1)+1, sphere_center(2)-1, sphere_center(2)+1, 0, sphere_center(3)+1],
'trail', 'b-', 'nobase');

    %update trace line
    set(trace_line, 'XData', positions(1, 1:i), 'YData', positions(2,1:i), 'ZData',
positions(3,1:i));

    pause(0.05);

end

fprintf('======================================\n');
disp('Animation Complete');
```

```matlab
fprintf('======================================\n\n');

%% --- Verification ---
%%%%%%%%%%%%%%%%%%%%%%%
disp('Verification...');

%calculate actual end effector pos
actual_positions = zeros(3, num_points);
for i = 1:num_points
    T_actual = robot.fkine(q_trajectory(i, :));
    actual_positions(:,i) = T_actual.t;
end

%calculate pos errors
position_errors = vecnorm(positions - actual_positions);
mean_error = mean(position_errors);
max_error = max(position_errors);

fprintf('======================================\n');
disp('Verification Results:');
fprintf('Mean Position Error: %.4f mm\n', mean_error * 1000);
fprintf('Max Position Error: %.4f mm\n', max_error * 1000);
fprintf('======================================\n\n');
```

Workload:

Bengaly:

- Robot choice
- Equations: 85%
- Programming: 15%

Vladyslava:

- Equations: 15%
- Programming: 85%
- Animation