Table of Contents

Rover Path Simulation with MPC and Dynamic Obstacles

```
clear all; close all; clc;
```

```
Execution of script MPC as a function is not supported:
/MATLAB Drive/Examples/MPC.m
```

Parameters

```
dt = 0.1;
sim_time = 30;
steps = sim_time / dt;

x_init = [0; 0; 0; 0];
target = [10; 8];

N = 10;
Q = diag([10, 10]);
R = diag([1, 5]);

static_obstacles = [
    4, 3, 0.4;
    7, 3, 0.4;
    5, 6, 0.4;
    8, 7, 0.4;
];

% Dynamic Obstacles [x, y, vx, vy, radius]
dynamic_obstacles = [
    2, 5,  1, 0,    0.4;
    6, 8, -1, 0.1,  0.4
];

max_accel = 2;
max_decel = -2;
max_steering = pi/4;
max_vel = 2;
```

Setup

```
x = x_init;
states = zeros(4, steps+1);
states(:,1) = x_init;
controls = zeros(2, steps);
reached_target = false;
target_tolerance = 0.4;

figure(1); clf;
h_ax = axes;
hold(h_ax, 'on');
axis equal; grid on;
xlim([-2 12]); ylim([-2 12]);
title('Rover MPC Path Planning');
xlabel('X Position (m)'); ylabel('Y Position (m)');

% Static Obstacles
for i = 1:size(static_obstacles, 1)
    rectangle('Position', ...
        [static_obstacles(i,1)-static_obstacles(i,3), ...
         static_obstacles(i,2)-static_obstacles(i,3), ...
         2*static_obstacles(i,3), 2*static_obstacles(i,3)], ...
         'Curvature', [1 1], 'FaceColor', [0.8 0.8 0.8]);
end

% Dynamic Obstacles (initial plot)
h_dyn_obs = gobjects(size(dynamic_obstacles,1), 1);
for i = 1:size(dynamic_obstacles,1)
    h_dyn_obs(i) = rectangle('Position', ...
        [dynamic_obstacles(i,1)-dynamic_obstacles(i,5), ...
         dynamic_obstacles(i,2)-dynamic_obstacles(i,5), ...
         2*dynamic_obstacles(i,5), 2*dynamic_obstacles(i,5)], ...
         'Curvature', [1 1], 'FaceColor', [1 0.4 0.4]);
end

h_target = plot(target(1), target(2), 'rp', 'MarkerSize', 15,
'MarkerFaceColor', 'r');
h_rover = plot(x(1), x(2), 'bo', 'MarkerSize', 10, 'MarkerFaceColor', 'b');
h_path = plot(x(1), x(2), 'b-', 'LineWidth', 2);
h_pred = plot([], [], 'g--', 'LineWidth', 1);
legend([h_rover, h_path, h_pred, h_target], 'Rover', 'Path', 'Prediction',
'Target');

drawnow;
```

Simulation Loop

```
for k = 1:steps
    if reached_target
        fprintf('Target reached at step %d (time: %.2fs)\n', k, k*dt);
        break;
    end

    % Combine static and dynamic obstacles
    current_dynamic_obs = dynamic_obstacles(:, [1 2 5]);
    all_obstacles = [static_obstacles; current_dynamic_obs];

    % Compute control via MPC
    [u_opt, pred_states] = mpc_control(x, target, all_obstacles, N, Q, R, dt,
...
        max_accel, max_decel, max_steering, max_vel);

    u = u_opt(:,1);
    controls(:,k) = u;
    x = rover_dynamics(x, u, dt);
    states(:,k+1) = x;

    % Update rover and path
    set(h_rover, 'XData', x(1), 'YData', x(2));
    set(h_path, 'XData', states(1,1:k+1), 'YData', states(2,1:k+1));

    % Prediction plot
    if ~isempty(pred_states)
        pred_x = zeros(N+1, 1);
        pred_y = zeros(N+1, 1);
        for i = 1:N+1
            pred_x(i) = pred_states{i}(1);
            pred_y(i) = pred_states{i}(2);
        end
        set(h_pred, 'XData', pred_x, 'YData', pred_y);
    end

    % Update dynamic obstacles
    for i = 1:size(dynamic_obstacles, 1)
        dynamic_obstacles(i,1:2) = dynamic_obstacles(i,1:2) + ...
                                   dynamic_obstacles(i,3:4) * dt;

        % Optional: boundary wrapping or bouncing could be added here
        set(h_dyn_obs(i), 'Position', ...
            [dynamic_obstacles(i,1)-dynamic_obstacles(i,5), ...
             dynamic_obstacles(i,2)-dynamic_obstacles(i,5), ...
             2*dynamic_obstacles(i,5), 2*dynamic_obstacles(i,5)]);
    end

    drawnow;
    pause(0.05);

    % Check for target reach
    if norm(x(1:2) - target) < target_tolerance
        reached_target = true;
    end
end

fprintf('Simulation complete.\n');
if reached_target
```

```
    fprintf('Target reached successfully!\n');
else
    fprintf('Failed to reach target within simulation time.\n');
end
```

--- Functions ---

```
function [u_opt, pred_states] = mpc_control(x0, target, obstacles, N, Q, R,
dt, ...
        max_accel, max_decel, max_steering, max_vel)

    nvars = 2*N;
    u0 = zeros(nvars, 1);
    lb = repmat([max_decel; -max_steering], N, 1);
    ub = repmat([max_accel;  max_steering], N, 1);

    options = optimoptions('fmincon','Display','off','Algorithm','sqp');
    [u_opt_vec, ~] = fmincon(@(u) objective_function(u, x0, target,
obstacles, N, Q, R, dt, max_vel), ...
        u0, [], [], [], [], lb, ub, [], options);

    u_opt = reshape(u_opt_vec, 2, N);

    if nargout > 1
        pred_states = cell(N+1, 1);
        pred_states{1} = x0;
        x_pred = x0;
        for i = 1:N
            u_i = u_opt(:,i);
            x_pred = rover_dynamics(x_pred, u_i, dt);
            pred_states{i+1} = x_pred;
        end
    end
end

function J = objective_function(u_vec, x0, target, obstacles, N, Q, R, dt,
max_vel)
    u = reshape(u_vec, 2, N);
    J = 0;
    x = x0;
    for i = 1:N
        a = u(1,i);
        delta = u(2,i);
        x = rover_dynamics(x, [a; delta], dt);
        pos_error = x(1:2) - target;
        J = J + pos_error' * Q * pos_error + u(:,i)' * R * u(:,i);
        for j = 1:size(obstacles, 1)
            obs_pos = obstacles(j, 1:2)';
            obs_rad = obstacles(j, 3);
            dist = norm(x(1:2) - obs_pos) - obs_rad;
            if dist < 1.5
                J = J + 50 * max(0, 1.5 - dist)^2;
            end
        end
        if x(4) > max_vel
            J = J + 50 * (x(4) - max_vel)^2;
        end
    end
end

function x_next = rover_dynamics(x, u, dt)
    x_pos = x(1); y_pos = x(2); theta = x(3); v = x(4);
    a = u(1); delta = u(2); L = 0.5;
    v_next = max(0, v + a * dt);
    theta_next = theta + (v / L) * tan(delta) * dt;
    x_pos_next = x_pos + v * cos(theta) * dt;
    y_pos_next = y_pos + v * sin(theta) * dt;
    x_next = [x_pos_next; y_pos_next; theta_next; v_next];
end
```

Published with MATLAB® R2024b