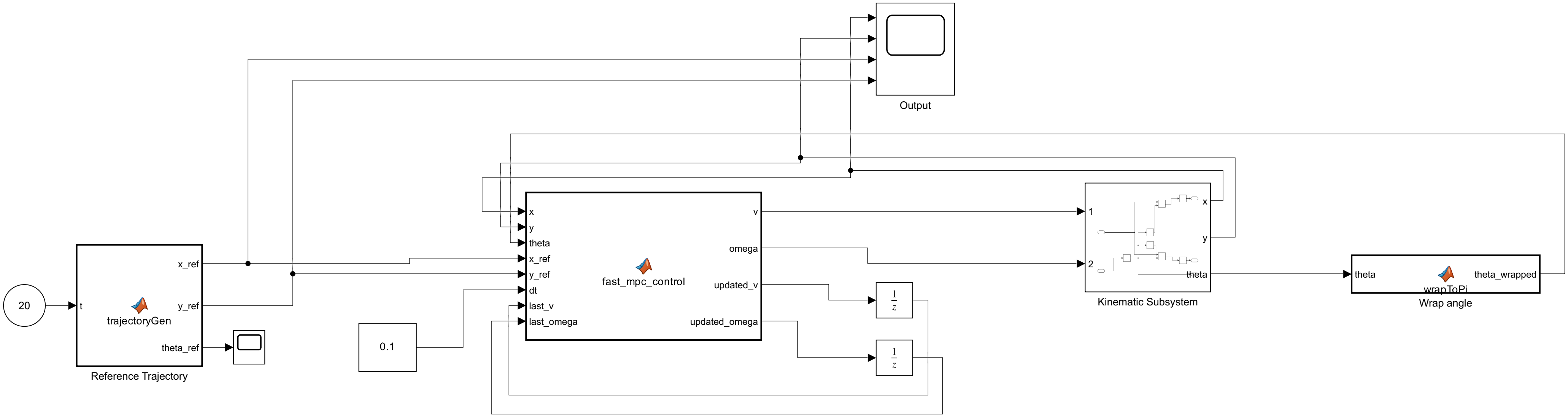
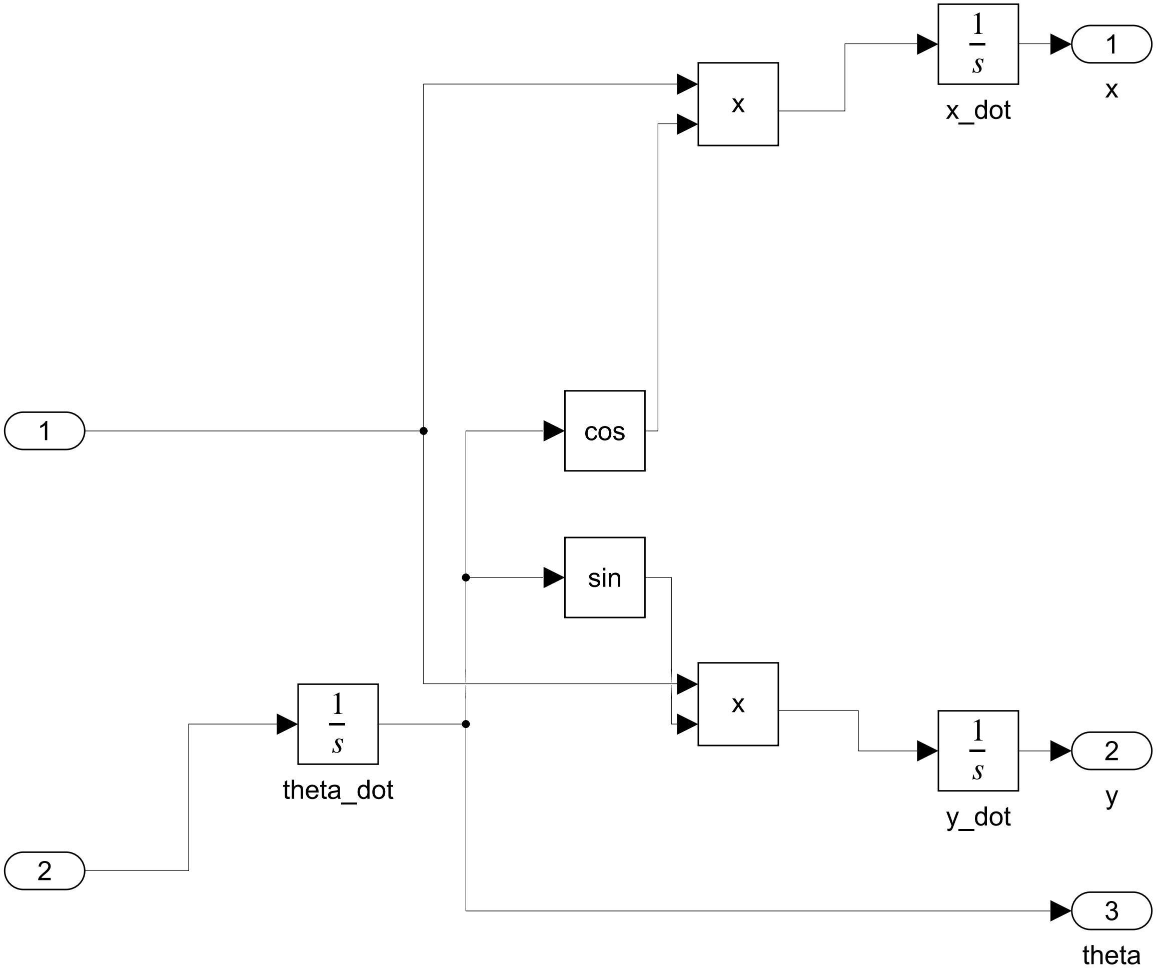

Output
20
t
trajectoryGen
x_ref
y_ref
theta_ref
Reference Trajectory
0.1
x
y
theta
x_ref
y_ref
dt
last_v
last_omega
fast_mpc_control
v
omega
updated_v
updated_omega
1/z
1/z
1
2
x
y
theta
Kinematic Subsystem
theta
wrapToPi
theta_wrapped
Wrap angle


1
2
1/s
theta_dot
cos
sin
x
x
1/s
x_dot
1/s
y_dot
1
x
2
y
3
theta

```
function [v, omega, updated_v, updated_omega] = fast_mpc_control(x, y, theta, x_ref, y_ref, dt, last_v, last_omega)
%#codegen
robotState = [x; y; theta];
targetPoint = [x_ref; y_ref];
[v, omega, updated_v, updated_omega] = FastMPCController(robotState, targetPoint, dt, last_v, last_omega);

function [best_v, best_omega, last_v, last_omega] = FastMPCController(robotState, targetPoint, dt, last_v, last_omega)
    % Unpack current state
    x = robotState(1);
    y = robotState(2);
    theta = robotState(3);

    x_target = targetPoint(1);
    y_target = targetPoint(2);

    % MPC parameters
    N = 20;  % Prediction horizon

    % Cost weights (speed-prioritized)
    w_pos = 0.5;
    w_theta = 1.0;
    w_v = 0.05;
    w_omega = 0.1;
    velocity_bias = 0.3;  % Reward for high velocity

    % Control limits
    v_min = 0.0;
    v_max = 5.0;
    omega_min = -pi;
    omega_max = pi;

    % Control search ranges (favoring acceleration)
    v_range_min = max(v_min, last_v - 0.3);
    v_range_max = min(v_max, last_v + 0.7);
    v_options = linspace(v_range_min, v_range_max, 5);
    omega_options = linspace(max(omega_min, last_omega - 0.5), min(omega_max, last_omega + 0.5), 7);

    % Initialize best control
    best_cost = inf;
    best_v = last_v;
    best_omega = last_omega;

    for vi = 1:length(v_options)
        for wi = 1:length(omega_options)
            v = v_options(vi);
            omega = omega_options(wi);

            pred_x = x;
            pred_y = y;
            pred_theta = theta;
            total_cost = 0;

            for i = 0:N-1
                % Predict next state
                pred_x = pred_x + v * cos(pred_theta) * dt;
                pred_y = pred_y + v * sin(pred_theta) * dt;
                pred_theta = pred_theta + omega * dt;
                % Errors
                pos_error = sqrt((x_target - pred_x)^2 + (y_target - pred_y)^2);
                desired_theta = atan2(y_target - pred_y, x_target - pred_x);
                theta_error = atan2(sin(desired_theta - pred_theta), cos(desired_theta - pred_theta));

                % Smoothness
                v_change = abs(v - last_v);
                omega_change = abs(omega - last_omega);

                % Reward for speed
                velocity_reward = -velocity_bias * v / v_max;

                % Step cost
                step_cost = w_pos * pos_error + w_theta * abs(theta_error) + w_v * v_change + w_omega * omega_change + velocity_re
                discount = 0.9 ^ i;
                total_cost = total_cost + discount * step_cost;
            end

            % Update best control
            if total_cost < best_cost
                best_cost = total_cost;
                best_v = v;
                best_omega = omega;
            end
        end
    end

    % Update memory for next call
    last_v = best_v;
    last_omega = best_omega;
end
end
```

```
function [x_ref, y_ref, theta_ref] = trajectoryGen(t)
    radius = 5;
    omega_ref = 0.2;
    x_ref = radius * cos(omega_ref * t);
    y_ref = radius * sin(omega_ref * t);
    theta_ref = omega_ref * t + pi/2;
end
```

```
function theta_wrapped = wrapToPi(theta)
    theta_wrapped = mod(theta + pi, 2*pi) - pi;
end
```