Table of Contents

Format

```
clc
clear all
close all
format compact
```

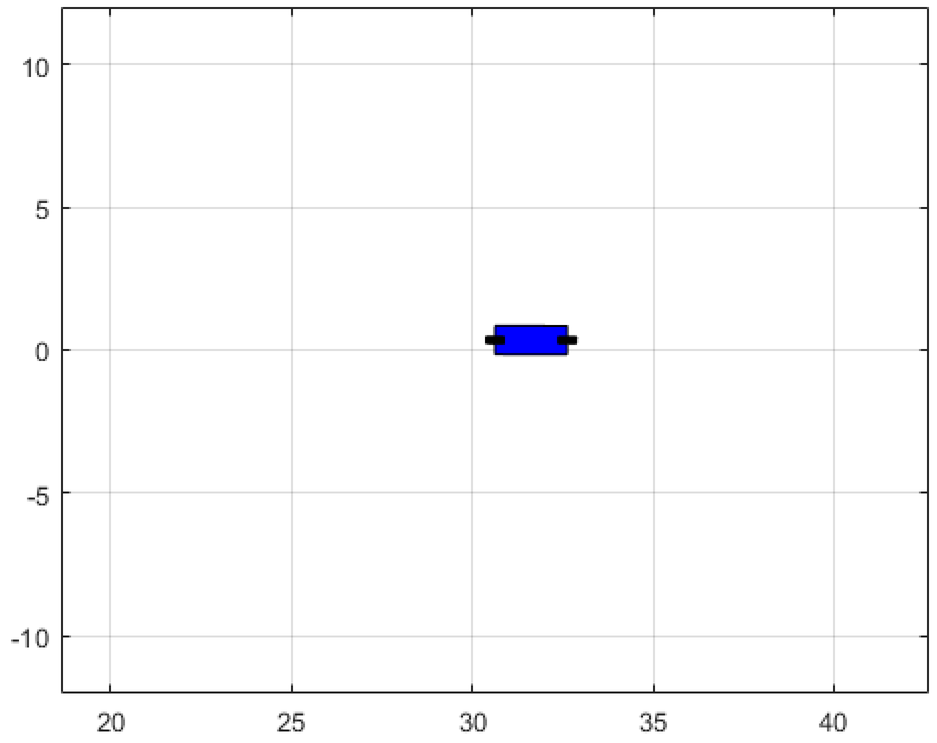
Physical/ Initial Characteristics

```
%Robot characteristics
robot.X = 0;
robot.Y = 10;
robot.Phi = 0;
robot.radius = 1;
robot.width = 2;
robot.length = 1;
robot.Vel = 1;
robot.angVel = 0;

% Wheel Characteristics
Wheel.radius = .25;
Wheel.wheel_width = 0.125;
Wheel.gamma = 0;
```

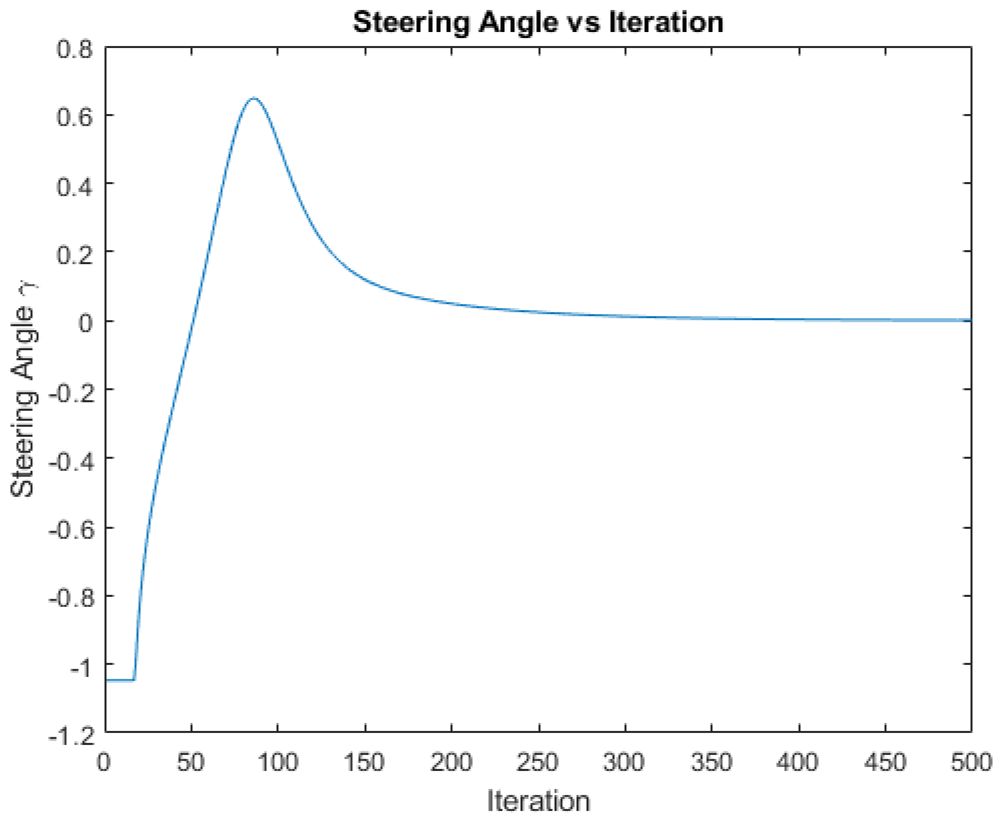
Control Implementation

```
% Desired y value
des.Y = 0;

% simulated dt
dt = .075;

% initial error in y
old_error = des.Y - robot.Y;

% tracks total error
error_sum = 0;

% noise term
drift = 0.1;

% hand picked gains
gains = [.3 2 .05];
```

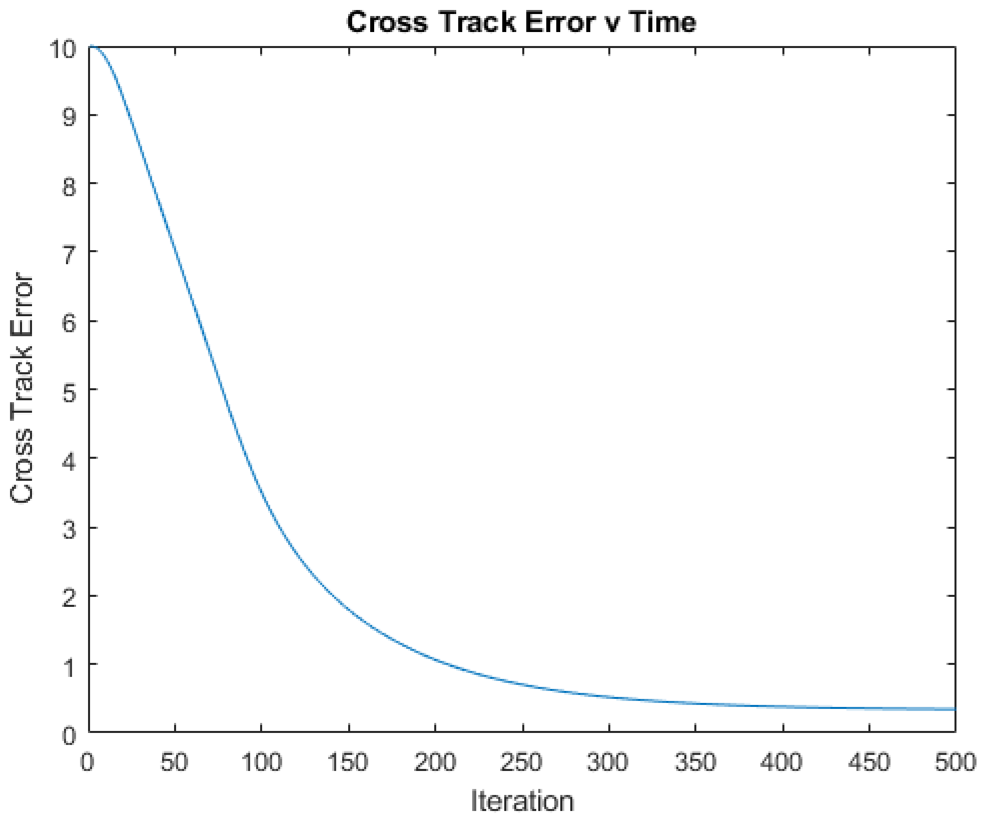
```matlab
y_values = [];
% Simulates and draws robot positions with controller
for i = 1:500
    y_front_wheel = drawRobot_Ackerman(robot, Wheel,i);
    pause(0)
    width = robot.width;
    robot = fwdSim(robot, dt);
    [omega, gamma, error, error_sum] = my_controller(robot, des, old_error,
error_sum, dt, gains, width, drift);

    Wheel.gamma = gamma;
    robot.angVel = omega;

    old_error = error;

    steering_angles(i) = gamma;

    y_values(i) = robot.Y;


end

figure()
    plot(steering_angles)
    xlabel("Iteration");
    ylabel("Steering Angle \gamma");
    title("Steering Angle vs Iteration")
figure()
    plot(y_values)
    xlabel("Iteration");
    ylabel("Cross Track Error");
    title("Cross Track Error v Time")
```



10
5
0
-5
-10
20
25
30
35
40



Steering Angle vs Iteration
Steering Angle γ
Iteration
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
0
50
100
150
200
250
300
350
400
450
500



```
function [y_front_wheel] = drawRobot_Ackerman(robot, Wheel, i)


%% Main body
length = robot.length; %y-direction
width = robot.width; % x-direction

% Body vertices
y_box = [-length/2 -length/2 length/2 length/2 -length/2];
x_box = [width 0 0 width width];

% Robot initial conditions
x = robot.X;
y = robot.Y;
phi = robot.Phi;

% Rotating and translating robot body
rot_matrix = [cos(phi), -sin(phi); sin(phi), cos(phi)];
box_rotated = rot_matrix * [x_box; y_box];

box_translated_rotated = [box_rotated(1,:) + x; box_rotated(2,:) + y];


%% Wheels
radius = Wheel.radius;
```

```
wheel_width = Wheel.wheel_width; %y direction


%Back Wheel

x_back_wheel = [(-radius), ( -radius), (radius), (radius), (-radius)];
y_back_wheel = [wheel_width, -wheel_width, -wheel_width, wheel_width,
wheel_width];

back_wheel_rotated = rot_matrix * [x_back_wheel; y_back_wheel];

back_wheel_translated_rotated = [back_wheel_rotated(1,:) + x;
back_wheel_rotated(2,:) + y];


%Front Wheel
gamma = Wheel.gamma;

% Initial position
x_front_wheel = [(-radius), ( -radius), (radius), (radius), (-radius)];
y_front_wheel = [wheel_width, -wheel_width, -wheel_width, wheel_width,
wheel_width];

% Rotate wheel by steering angle
front_rot_matrix = [cos(gamma), -sin(gamma); sin(gamma), cos(gamma)];
front_wheel_steered = front_rot_matrix * [x_front_wheel; y_front_wheel];

% Put wheel at front of the body
front_wheel_translated = [front_wheel_steered(1,:) + width;
front_wheel_steered(2,:)];

% Rotate wheel with body
front_wheel_rotated = rot_matrix * front_wheel_translated;

% Combine rotation and translation
front_wheel_translated_rotated = [front_wheel_rotated(1,:) + x;
front_wheel_rotated(2,:) + y];


%% Plotting

if (mod(i-1, 10) == 0)
cla
fill(box_translated_rotated(1,:), box_translated_rotated(2,:), 'b')
hold on
grid on
fill(back_wheel_translated_rotated(1,:), back_wheel_translated_rotated(2,:),
'k');
hold on
fill(front_wheel_translated_rotated(1,:),
front_wheel_translated_rotated(2,:), 'k')
axis([(-12+ x) (12+x) -12 12])
drawnow;
end
```

```
function [omega, gamma, error, error_sum] = my_controller(robot, des,
old_error,error_sum, dt, gains, width, drift)

  kp = gains(1);
  kd = gains(2);
  ki = gains(3);

  error = des.Y - robot.Y;%(robot.Y + (width*sin(robot.Phi)));


  % if abs(error) < .5
  %     error_sum = error_sum + error * dt;
  %     gamma = kp*error + kd*(error-old_error)/dt + ki*error_sum;
  % else
  %     gamma = kp*error + kd*(error-old_error)/dt;
  % end
  %


  gamma = kp*error + kd*(error-old_error)/dt;

  gamma = gamma + drift;

%gamma = atan2(sin(gamma), cos(gamma));

  if (gamma>=pi/3)
      gamma = pi/3;
  elseif (gamma<=-pi/3)
          gamma=-pi/3;
  end


  omega = (robot.Vel/robot.width) * tan(gamma);


end



function [robot] = fwdSim(robot,dt)

%Current
X = robot.X;
Y = robot.Y;
Phi = robot.Phi;
Vel = robot.Vel;
angVel = robot.angVel;

%Future
X = X + Vel*cos(Phi)*dt;
```

```
Y = Y + Vel*sin(Phi)*dt;
Phi = Phi + angVel*dt;

Phi = atan2(sin(Phi), cos(Phi));

%Passing values
robot.X = X;
robot.Y = Y;
robot.Phi = Phi;


end
```

Published with MATLAB® R2024a