Nonholonomic Differential Drive Robot Point-to-Point Motion Control System

Raisal P Wardana - 13319072

1. Nonholonomic Robot Model

A nonholonomic robot is a type of robot that is subject to nonholonomic constraints, which are constraints on the system's motion that cannot be represented as a set of independent and redundant velocities. Nonholonomic robots are typically characterized by the fact that their degrees of freedom (DOFs) are not independent, meaning that the motion of one DOF can influence the motion of another.

One common example of a nonholonomic robot is a car-like robot that is able to move forward, backward, turn left, and right. While the robot has two DOFs (forward/backward motion and turning), it is not possible to independently control these DOFs. For example, if the robot is moving forward, it cannot turn in place; it must first come to a stop and then turn. This constraint is an example of a nonholonomic constraint, as it cannot be represented as a set of independent and redundant velocities.

A nonholonomic robot can be mathematically modelled as such[1],

$$\begin{bmatrix}\dot{\theta}\\ \dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}0 & 1\\ \cos\theta & 0\\ \sin\theta & 0\end{bmatrix}\begin{bmatrix}v\\ w\end{bmatrix}$$

In the case of a two-wheeled differential drive nonholonomic robot, usually motor speed is the controllable variable. Therefore, if we want to have motor speed as the input, the mathematical model needs to be modified as such,

$$\begin{bmatrix}\dot{\theta}\\ \dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}-r/2d & r/2d\\ \frac{\mathrm{r}}{2}\cos\theta & \frac{r}{2}\cos\theta\\ \frac{\mathrm{r}}{2}\sin\theta & \frac{\mathrm{r}}{2}\sin\theta\end{bmatrix}\begin{bmatrix}\omega_L\\ \omega_R\end{bmatrix}$$

Where ω_R and ω_L are the angular speed of the motor, r is the wheel's radius, and d is the distance from robot's centre coordinate to the wheel. The robot is tasked with a point-to-point motion, with a designated coordinate and angle as the goal. The system can be modelled as such,

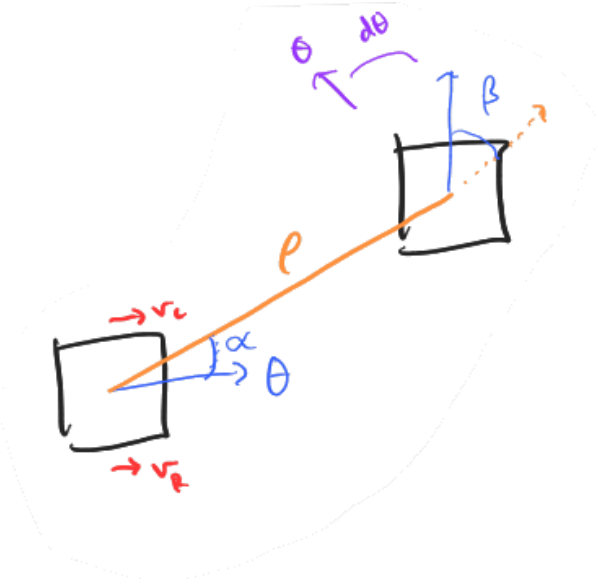


Figure 1-1 Differential Drive Robot System Model

Deviation of the robot from the goal is represented with $\Delta x, \Delta y$, and $\Delta\theta$. The distance (ρ) from current coordinate to goal coordinate is defined by,

$$\vec{\rho} = \overrightarrow{P_i P_g} = (\Delta x, \Delta y)$$

$$|\rho| = \sqrt{\Delta x^2 + \Delta y^2}$$

α and β as the angle between distance vector to the robot angle (θ) and goal angle correspondingly. Mathematically, both can be calculated using,

$$\alpha = arctan2(\Delta y, \Delta x) - \theta_i$$

$$\beta = arctan2(\Delta y, \Delta x) - \theta_g$$

The position of each wheel can be inferred from the robot coordinate using the equation.

$$x_L = x + d\sin(-\theta)$$

$$y_L = y + d\cos(-\theta)$$

$$x_R = x - d\sin(-\theta)$$

$$y_R = y - d\cos(-\theta)$$

2. Motion Control System

One of the advantages of a differential drive robot is its ability to rotate in place. This ability can be exploited in scenarios where the difference between current angle and destination angle is extreme. A forward only robot will need to traverse a longer path while a differential drive robot can prioritize arriving at destination then adjusting its angle. A differential drive robot can also move backwards, which result in significantly faster traversal when the destination coordinate is located behind the robot.

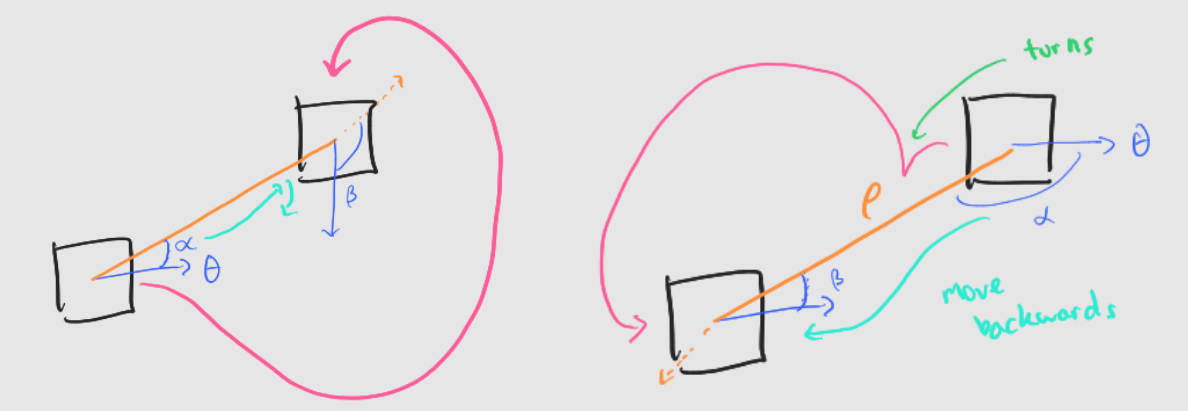


Figure 2-1 Previous System Path (pink) and Proposed System Path (cyan)

The proposed system will check whether the destination is in front or behind the robot, and the angle difference between current angle and destination angle of the robot. This will decide whether the robot will move forwards or backwards, and how the path will be generated. The generated path will prioritize robot coordinate over its orientation, as the robot can just rotate in place after arriving at the goal coordinate.

To reach the goal coordinate in the shortest period, a heavy priority is placed on coordinate deviation. The shortest path from current robot coordinate to goal coordinate is the Euclidean distance between the two points. As such, the shortest travel distance can be achieved by employing a simple path following algorithm by using α as the path angle deviation.

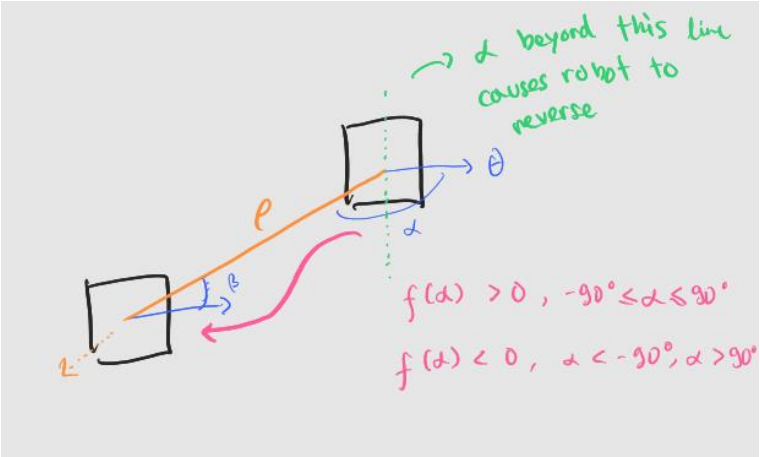


Figure 2-2 Movement Direction Rule

If the angle path angle deviation α is above 90°, moving backward will significantly shorten the duration. Therefore, a rule is established so that the direction of movement is,

$$direction = f(\alpha) = \begin{cases} f(\alpha) = 1 : -\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2} \\ f(\alpha) = -1 : \alpha < -\frac{\pi}{2}, \alpha > \frac{\pi}{2} \end{cases}$$

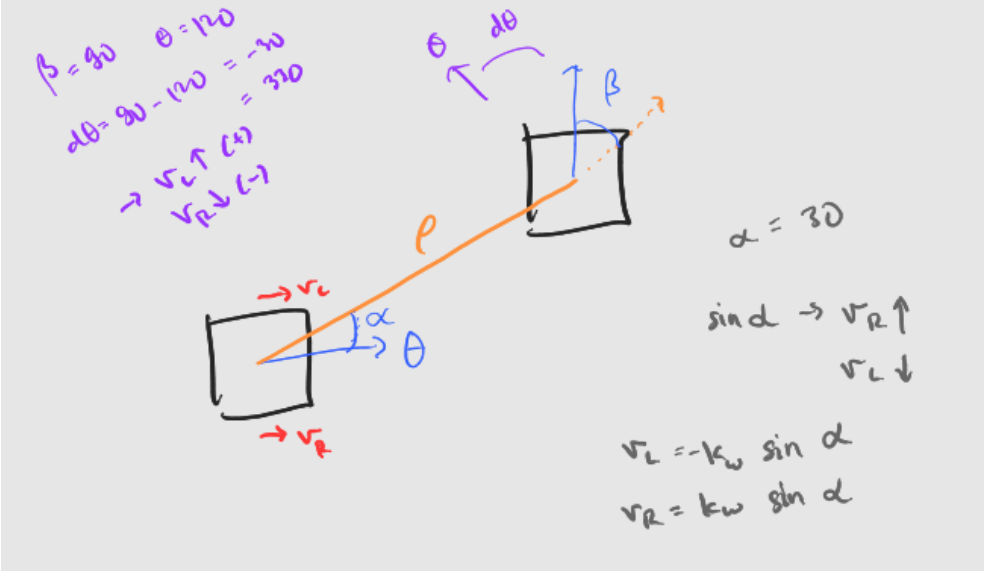


Figure 2-3 Coordinate Algorithm Notes

There are three scenario on how the robot can reach the goal.

1. Robot approaches the goal with the same angle as the designated angle
2. One of robot's wheel reaches the designated position for said wheel
3. Robot arrives at the goal coordinate with angle deviation

For scenario one, there's no required angle adjustment.

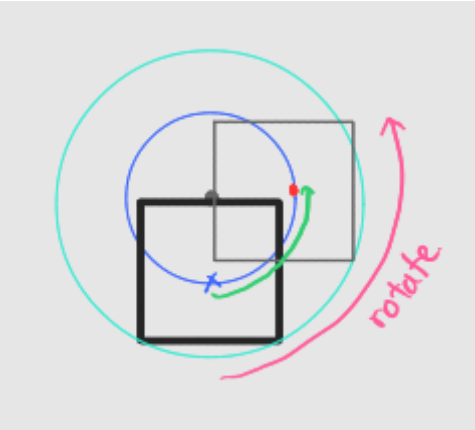


Figure 2-4 Rotational path of the robot with one stationary wheel

For scenario two, the goal coordinate of the wheel can be calculated using previously mentioned formula. If the wheel reaches its designated coordinate, there's no need for said wheel to move as the robot centre coordinate coincides with the goal coordinate in the same circular path with the stationary wheel as the rotational axis.

For scenario three, the robot centre coordinate arrived at the goal coordinate. However, there's an angle deviation. All the robot required to do is to rotate in place to fulfill the goal angle.

After considering each scenario, the control formula for each wheel is as such,

$$v_L = v_{max} * \frac{2}{\pi} * \arctan\left[\frac{\pi}{2}\left(direction(k_v\rho - k_\omega \sin\alpha)\rho_L + \frac{k_\omega \sin(\Delta\theta)}{\rho_L + 1}\right)\right]$$

$$v_L = v_{max} * \frac{2}{\pi} * \arctan\left[\frac{\pi}{2}\left(direction(k_v\rho + k_\omega \sin\alpha)\rho_L - \frac{k_\omega \sin(\Delta\theta)}{\rho_L + 1}\right)\right]$$

The function $f(x) = \frac{2}{\pi}\arctan\left[\frac{\pi}{2}kx\right]$ is employed to normalized the proportional formula for v_{max}. The directional and path following portion of the formula is colored orange and the rotational portion is colored in green.

The directional portion of each wheel is multiplied by ρ_L, so that the wheel will not move when the wheel coordinate deviation approaches zero. While the rotational potion is divided by $\rho_L + 1$, this is done to give it less priority over the path following algorithm as discussed previously.

3. Testing and Result

eight test scenarios were devised to test the algorithm provided in class and designed algorithm. Tests includes lateral movements with extreme angle, movement in the same axis, and scenario where backward movement is advantageous.

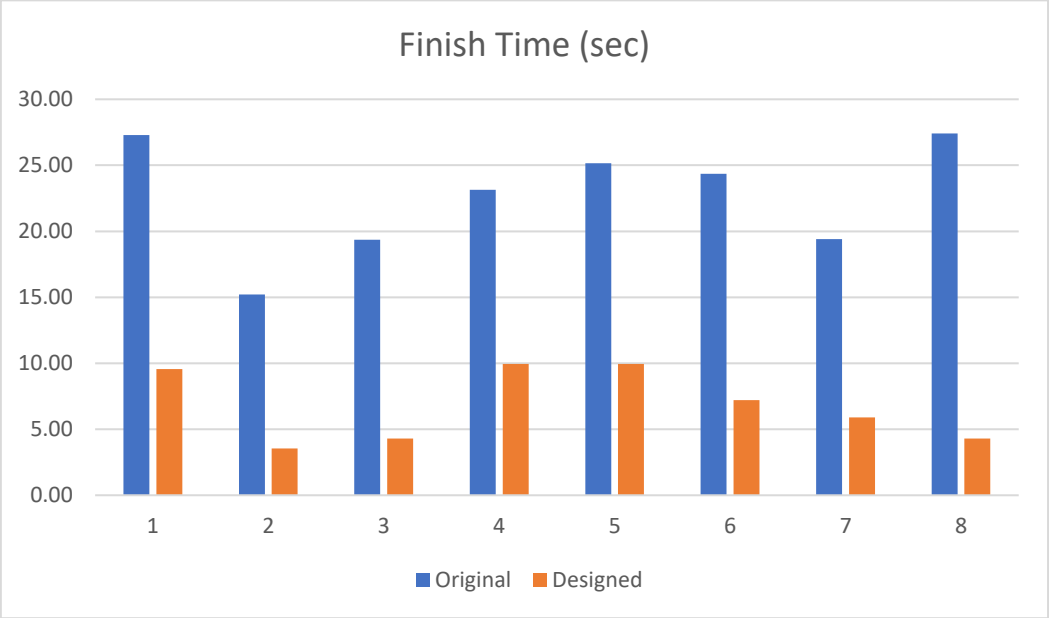


Figure 3-1 Finish time comparison

The designed system manages to reduce finish time by 15.83 seconds in average a reduction of 69.73% in travel duration, or 231.44% faster travel speed than class-provided algorithm.

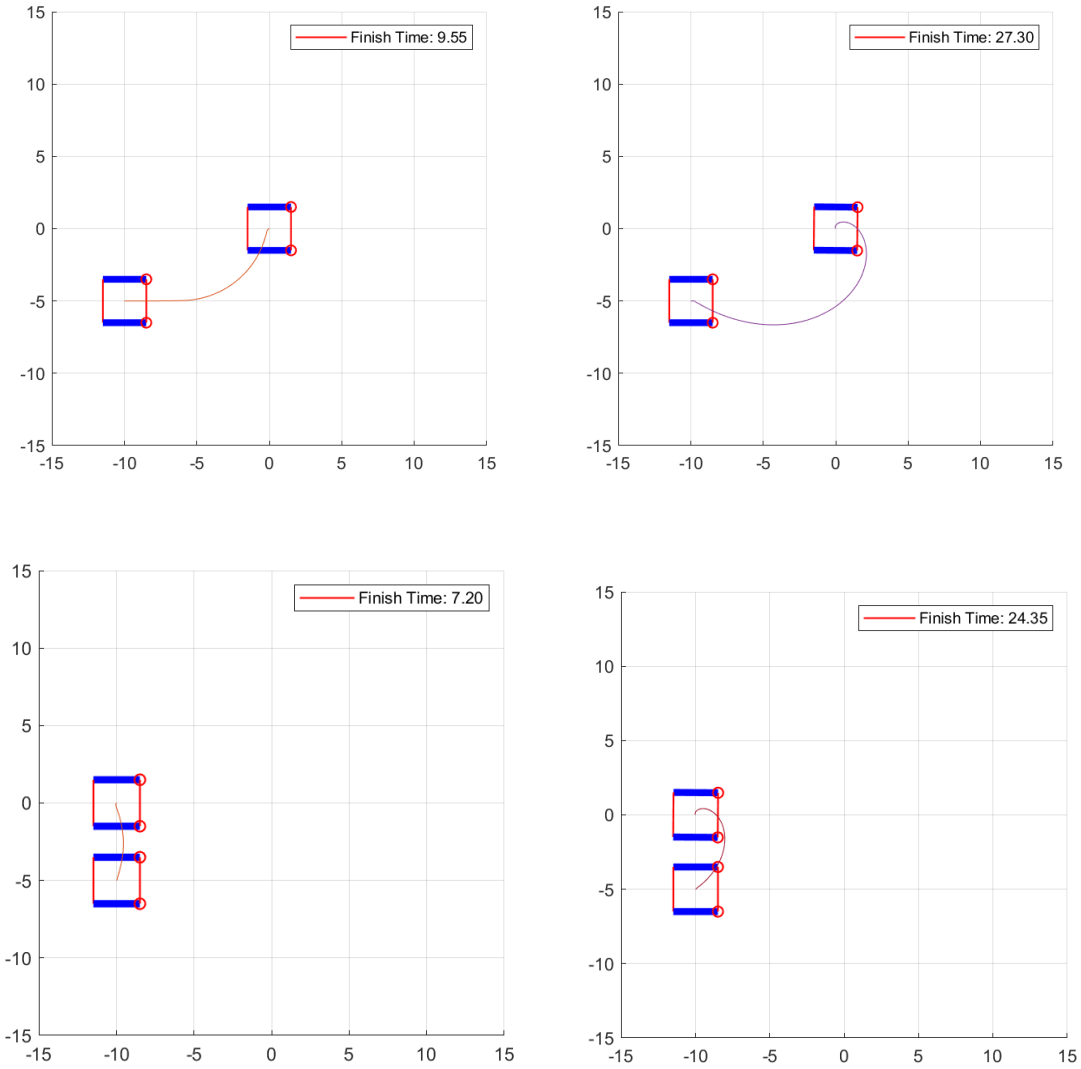


Figure 3-2 Example test results of designed system (left) and class provided system (right)

Shorter path planning is definitely one of the factors of improvement as illustrated by **Figure 3-2**. The designed system generates significantly shorter paths by prioritizing goal coordinate over goal angle. Another improvement also can be seen from the velocity data of the tests.

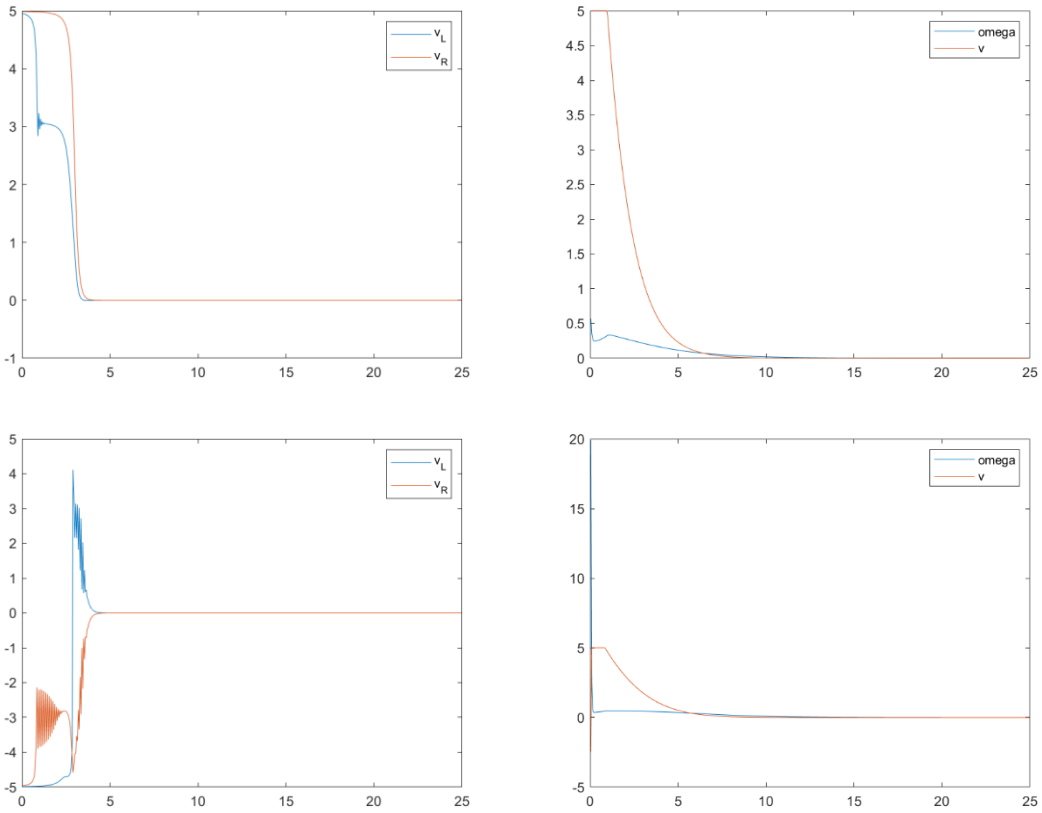


Figure 3-3 Test 2 and test 3 velocity data of designed system (left) and class provided system (right)

Designed system makes maximal use of both motors capability. In the designed system, both motors reaches maximum speed more often than provided system. This is the benefit of normalizing the proportional formula and scaling it to the maximum velocity.

4. Conclusion

By implementing backward motion and stationary rotation, the designed motion control system shortened the robot travel duration by 69.73% over class-provided algorithm with the exact same condition. Inverse kinematics was used to infer wheel goal coordinate, providing a significant improvement in path planning. By normalizing the proportional formula and scaling it to the motor maximum speed, the designed system increased motors usage efficiency significantly. Resulting in more effective use of the motor capability. In general, the designed algorithm generated shorter paths and is more effective at maximizing the motors capability.

Appendices

All source code and figures can be accessed at,

https://github.com/IEatCodeDaily/WheeledRobotMotionControlSim

4.1. Source Code – Designed System

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

%======Test Conditions List======
testConditions = [-10 -5 0 0 0 0;
    -10 -5 0 0 0 pi/2;
    -10 -5 0 0 0 pi;
    -10 -5 0 0 0 3/2*pi;
    -10 -5 0 0 0 -pi/4;
    -10 -5 0 -10 0 0;
    0 0 0 -10 -5 0;
    0 0 0 -10 -5 pi]; %x_i,y_i,theta_i,x_g,y_g,theta_g

%======Choose which test to do=====
%test = 8

for test=1:1:8
%======Code starts here========
% Constants
kv = 0.8;   %Velocity constant
kw = 10;     %Angular Constant

% robot vars
robot_distToCentre = 1.5; %Distance from center to wheel
max_v = 5;

% input initial coordinate & orientation here
x(1) = testConditions(test,1);
y(1) = testConditions(test,2);
theta(1) = testConditions(test,3);%0=hadap kanan, pi/2=hadap atas,2pi=hadap
kiri,3pi/2=hadap bawah,

% initialize variables
x_L(1) = x(1) + robot_distToCentre * sin(-theta(1));
y_L(1) = x(1) + robot_distToCentre * cos(-theta(1));
x_R(1) = x(1) - robot_distToCentre * sin(-theta(1));
y_R(1) = x(1) - robot_distToCentre * cos(-theta(1));
v_L(1) = 0;
v_R(1) = 0;

t = 0;
dt = 0.05;

xd = testConditions(test,4);
yd = testConditions(test,5);
thetad = testConditions(test,6);
xd_L = xd + robot_distToCentre * sin(-thetad);
yd_L = xd + robot_distToCentre * cos(-thetad);
xd_R = xd - robot_distToCentre * sin(-thetad);
yd_R = xd - robot_distToCentre * cos(-thetad);
```

```
dx = x(1)-xd;
dy = y(1)-yd;
dtheta = theta(1)-thetad;
finish = 0;

i = 1;
while (abs(dx) > 0.01 || abs(dy) > 0.01 || abs(dtheta/thetad) > 0.01 || i < 500)
&& i < 1000 %limit iteration to 1000 just in case
    % deviation from goal
    dx = xd - x(i);
    dy = yd - y(i);
    dtheta = mod(thetad - theta(i), 2*pi);
    dx_L = dx + robot_distToCentre * sin(-thetad);
    dy_L = dy + robot_distToCentre * cos(-thetad);
    dx_R = dx - robot_distToCentre * sin(-thetad);
    dy_R = dy - robot_distToCentre * cos(-thetad);

    % Euclidean Distance
    rho(i) = sqrt(dy^2 + dx^2);
    rho_L(i) = sqrt(dy_L^2 + dx_L^2);
    rho_R(i) = sqrt(dy_R^2 + dx_R^2);

    % Angle between deviation vector and robot
    alpha(i) = mod(atan2(dy, dx), 2*pi) - theta(i);
    alpha(i) = mod(alpha(i),2*pi);

    %  - UNUSED - Angle between deviation vector and destination angle
    beta(i) = mod(atan2(dy, dx), 2*pi) - thetad;
    beta(i) = mod(beta(i),2*pi);

    %Robot direction
    dir = sign(sin(alpha(i)+pi/2));

    %Wheel velocity

    v_L(i) = max_v * 2/pi * atan(pi/2* ( dir*(kv*rho(i) - kw*sin(alpha(i)))*
rho_L(i) + kw*sin(dtheta)/(rho_L(i)+1) ) );
    v_R(i) = max_v * 2/pi * atan(pi/2* ( dir*(kv*rho(i) + kw*sin(alpha(i)))*
rho_R(i) - kw*sin(dtheta)/(rho_R(i)+1) ) );

    %Velocity check
    if abs(v_L(i)) > max_v
        v_L(i) = 0.99 * v_L(i);
    end

    if abs(v_R(i)) > max_v
        v_R(i) = 0.99* v_R(i);
    end

    %Position update
    x(i+1) = x(i) + ( v_L(i) + v_R(i) ) * cos(theta(i)) / 2 * dt;
    y(i+1) = y(i) + ( v_L(i) + v_R(i) ) * sin(theta(i)) / 2 * dt;
    theta(i+1) = theta(i) + ( -v_L(i) + v_R(i) ) / (2*robot_distToCentre) * dt;

    % position of wheel
    x_L(i+1) = x(i+1) + robot_distToCentre * sin(-theta(i+1));
    y_L(i+1) = x(i+1) + robot_distToCentre * cos(-theta(i+1));
```

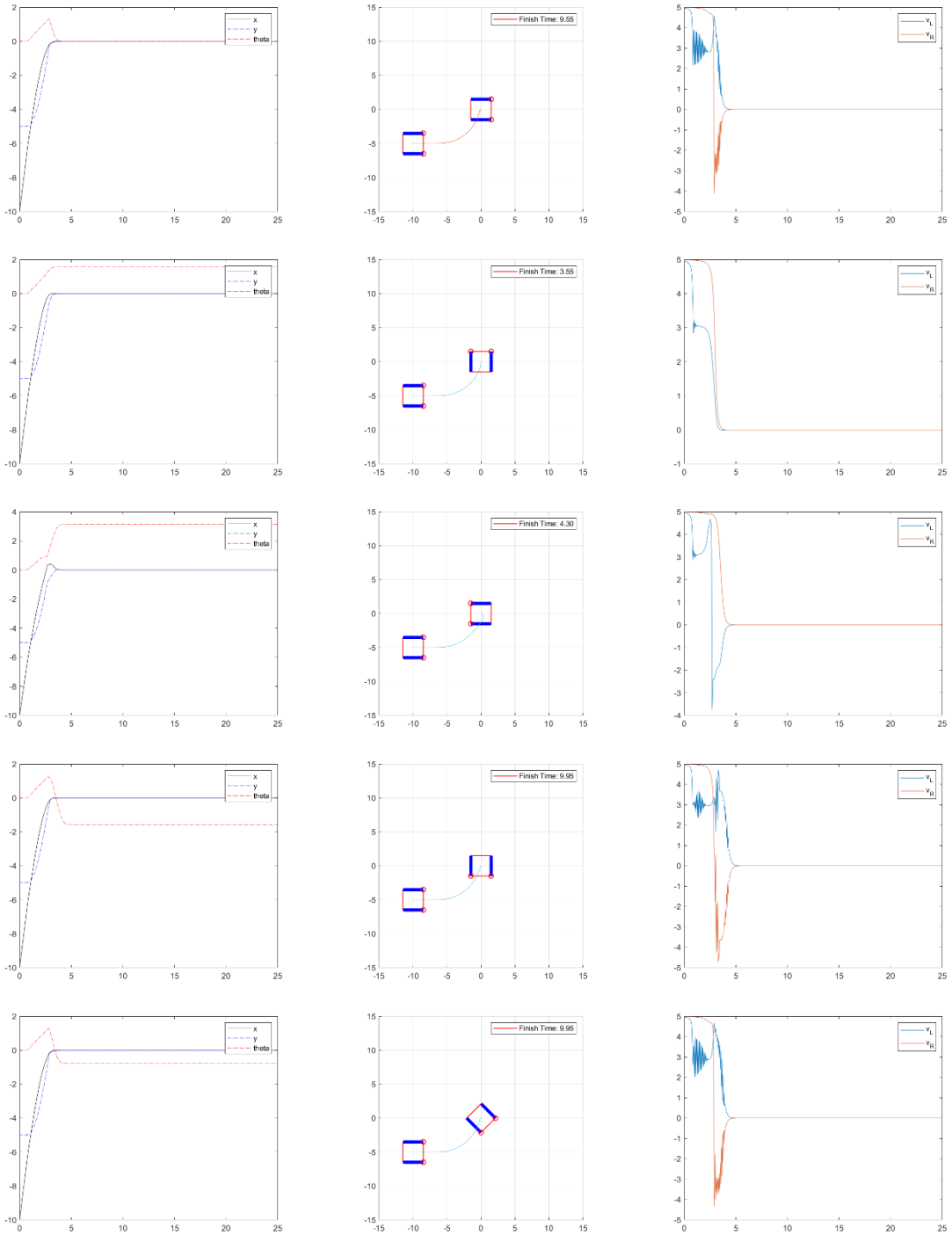
```
    x_R(i+1) = x(i+1) - robot_distToCentre * sin(-theta(i+1));
    y_R(i+1) = x(i+1) - robot_distToCentre * cos(-theta(i+1));

    t(i+1) = t(i) + dt;
    i = i + 1;


    if abs(dx) < 0.01 && abs(dy) < 0.01 && abs(dtheta) < 0.01
        if finish == 0
        finish = t(i)
        end
    end


end
%end

x = x'; y = y'; theta = theta'; t = t';

%plot(t, x, 'k-', t, y, 'b-.', t, theta, 'r-.');

pathLength = length(x);
Xc = x(1);
Yc = y(1);
fig1 = figure("Name","Simulation");
hold on;

plot (Xc,Yc,'r-','LineWidth',1);
% if (y(1)<=8) && (y(1)>=2) && (x(1)<42) && (x(1)>=1)
% plot(yd1,yd2,'y.','LineWidth',1);
% hold on;
% elseif (y(1)<=8) && (y(1)>=2) && (x(1)<=90) && (x(1)>48)
% plot(yd1,yd2,'y.','LineWidth',1);
% hold on;
% elseif (y(1)<16) && (y(1)>2) && (x(1)>=42) && (x(1)<=48)
% plot(xd,yd,'y.','LineWidth',1);
hold on;
%end




%f=[50:90];
%g=[0:40];
%h=[10:20];
%plot (g,10,'go',40,h,'go',50,h,'go',f,10,'go');
%hold on

axis equal
axis([-15 15 -15 15])
grid on
Xr_c  = 5;
Yr_c  = 0;
Xr_c1 = 5/1.4142;
Yr_c1 = 0;
gama  = pi/4;

path = size(t);
```

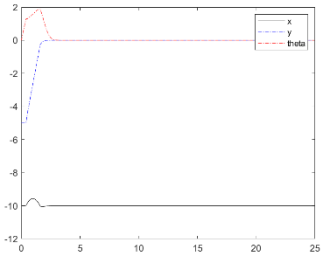
```
Na    = path(1,1);

for j = 1:1:length(t)  % change to M for sinewave, N for circle
R1    = 5/1.414;
R2    = 5/1.414;
Xc    = x(j);
Yc    = y(j);
fai   = theta(j);
Xr    = Xc+Xr_c*cos(fai)-Yr_c*sin(fai);
Yr    = Yc+Xr_c*sin(fai)+Yr_c*cos(fai);
Xr1   = Xc+Xr_c1*cos(fai+gama)-Yr_c1*sin(fai+gama);
Yr1   = Yc+Xr_c1*sin(fai+gama)+Yr_c1*cos(fai+gama);
%theta_r_dot = q(j,6);
%theta_l_dot = q(j,7);
sinfai(j) = sin(fai);
cosfai(j) = cos(fai);
%linearvelocity(j) = (theta_l_dot+theta_r_dot)*0.75/2;
%theta_r_dotre(j)  = theta_r_dot;
%theta_l_dotre(j)  = theta_l_dot;
%theta_r(j) = q(j,4);
%theta_l(j) = q(j,5);
%Xcre(j)  = q(j,1);
%Ycre(j)  = q(j,2);
faire(j) = fai;

[Robot] = Robotplot(Xc, Yc, fai, robot_distToCentre);

%********************* Create the 4 lines that draw the box of the mobile robot
************************************

Robot1x = [Robot(1,1) Robot(1,3)];
Robot1y = [Robot(1,2) Robot(1,4)];

Robot2x = [Robot(1,3) Robot(1,5)];
Robot2y = [Robot(1,4) Robot(1,6)];

Robot3x = [Robot(1,5) Robot(1,7)];
Robot3y = [Robot(1,6) Robot(1,8)];

Robot4x = [Robot(1,7) Robot(1,1)];
Robot4y = [Robot(1,8) Robot(1,2)];

Robotlinkx  = [Xr Xc];
Robotlinky  = [Yr Yc];

Robotlinkx1 = [Xr1 Xc];
Robotlinky1 = [Yr1 Yc];

Robotlinkx2 = [Xr1 Xr];
Robotlinky2 = [Yr1 Yr];


RobotBox(1,:) = [Robot1x Robot2x Robot3x Robot4x];
RobotBox(2,:) = [Robot1y Robot2y Robot3y Robot4y];

plot (Xc,Yc);

XrYr         = [Xr Yr];
XcYc         = [Xc Yc];
```

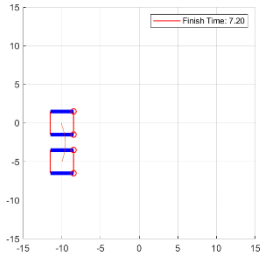
```
Xr1Yr1       = [Xr1 Yr1];
XrYrtoXcYc   = pdist([XrYr;XcYc]);
Xr1Yr1toXcYc = pdist([Xr1Yr1;XcYc]);
XrYrtoXr1Yr1 = pdist([XrYr;Xr1Yr1]);

MMcos = (-(XrYrtoXcYc*XrYrtoXcYc-Xr1Yr1toXcYc*Xr1Yr1toXcYc-
XrYrtoXr1Yr1*XrYrtoXr1Yr1)/(2*XrYrtoXr1Yr1*Xr1Yr1toXcYc));
MM(j) = sqrt(1 - MMcos^2);

if j==1

    h1 =plot(Robot1x, Robot1y,'b-','LineWidth',4);
    h2 =plot(Robot2x, Robot2y,'r-','LineWidth',1);
    h3 =plot(Robot3x, Robot3y,'b-','LineWidth',4);
    h4 =plot(Robot4x, Robot4y,'or-','LineWidth',1);
   % h5 =plot(Robotlinkx, Robotlinky,'ro','LineWidth',1);
    %h6 =plot(Robotlinkx1, Robotlinky1,'r-','LineWidth',1);
    %h7 =plot(Robotlinkx2, Robotlinky2,'r-','LineWidth',1);

%    set(h1,'EraseMode','xor');
%    set(h2,'EraseMode','xor');
%    set(h3,'EraseMode','xor');
%    set(h4,'EraseMode','xor');
    %set(h5,'EraseMode','xor');

elseif j==2
    h1 =plot(Robot1x, Robot1y,'b-','LineWidth',4);
    h2 =plot(Robot2x, Robot2y,'r-','LineWidth',1);
    h3 =plot(Robot3x, Robot3y,'b-','LineWidth',4);
    h4 =plot(Robot4x, Robot4y,'or-','LineWidth',1);
   %
else

    set(h1,'XData', Robot1x,'YData',Robot1y)
    set(h2,'XData', Robot2x,'YData',Robot2y)
    set(h3,'XData', Robot3x,'YData',Robot3y)
    set(h4,'XData', Robot4x,'YData',Robot4y)
    %set(h5,'XData', Robotlinkx,'YData',Robotlinky)
    %set(h6,'XData', Robotlinkx1,'YData',Robotlinky1)
    %set(h7,'XData', Robotlinkx2,'YData',Robotlinky2)

    end
    pause(0.001)

end

plot(x,y);legend(sprintf('Finish Time: %.2f',finish)); %plot trails

%==========Metrics==========
fig2 = figure("Name","Coordinate"); plot(t(1:500), x(1:500), 'k-', t(1:500),
y(1:500), 'b-.', t(1:500), theta(1:500), 'r-.'); legend('x','y','theta');
fig3 = figure("Name", "Velocity");
plot(t(2:500),v_L(2:500),t(2:500),v_R(2:500)); legend('v_L','v_R');
%figure("Name","Wheel Coord"); plot(t, x_L, t, y_L, t, x_R, t, y_R);
legend('x_L','y_L','x_R','y_R');
%figure("Name","Angle"); plot(t, theta, t(2:end), alpha, t(2:end), beta);
legend('theta','alpha','beta');
saveas(fig1,'Result/Designed_'+ string(test)+'_Simulation.png');
saveas(fig2,'Result/Designed_'+ string(test)+'_Coordinate.png');
```

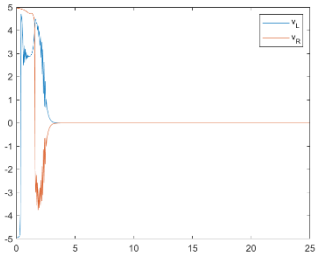
```
saveas(fig3,'Result/Designed_'+ string(test)+'_Velocity.png');
hold on;
end
```

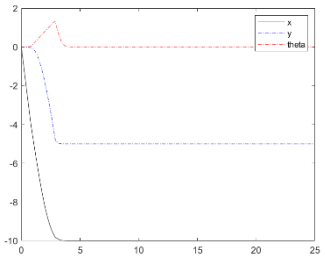
4.2. Test and Result Figures

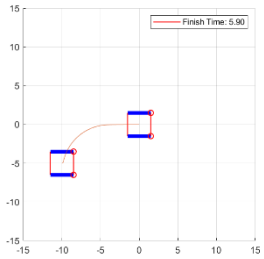


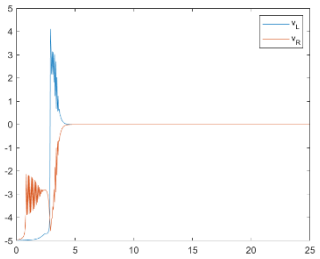

x
y
theta

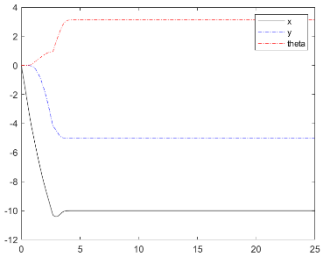

Finish Time: 7.20

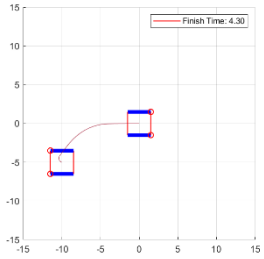

v_L
v_R

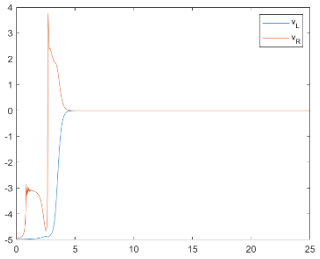

x
y
theta

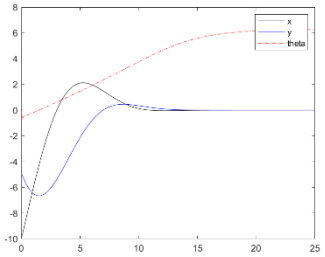

Finish Time: 5.90

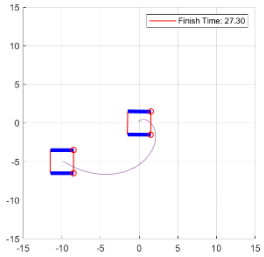

v_L
v_R

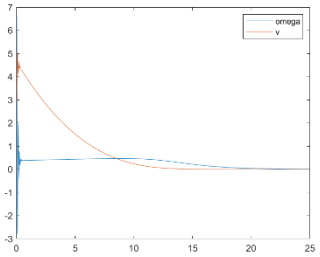

x
y
theta

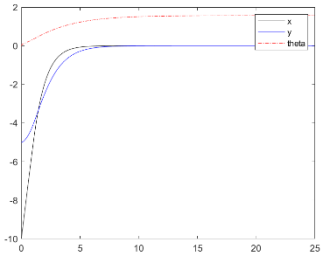

Finish Time: 4.30

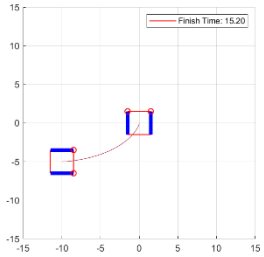

v_L
v_R

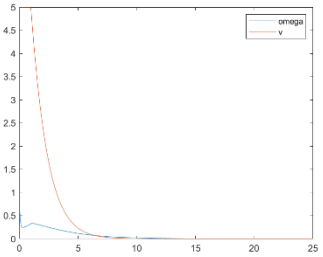

x
y
theta

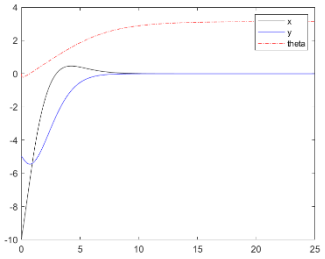

Finish Time: 27.30

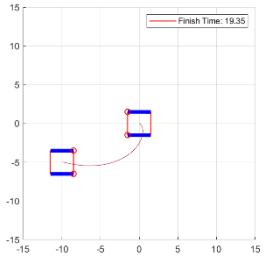

omega
v

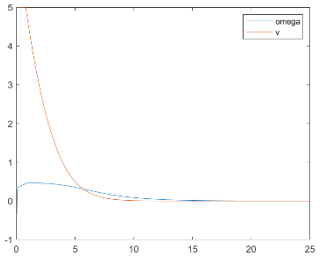

x
y
theta

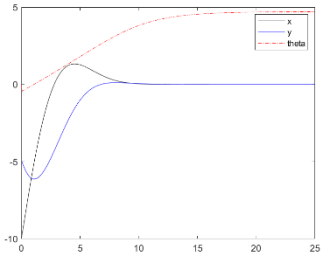

Finish Time: 15.20

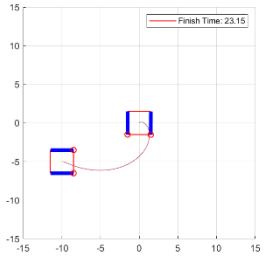

omega
v

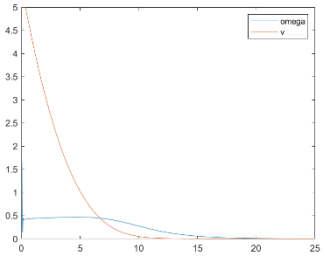

x
y
theta

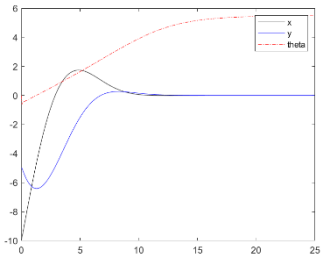

Finish Time: 19.35

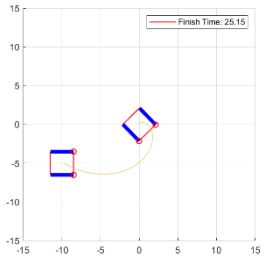

omega
v

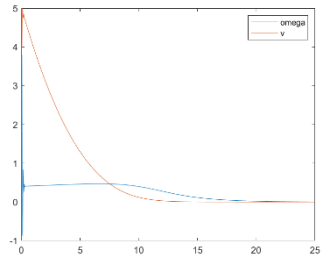

x
y
theta

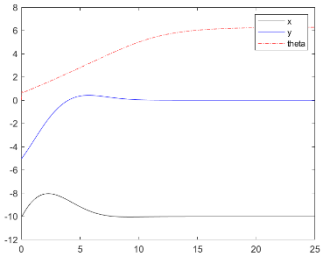

Finish Time: 23.15

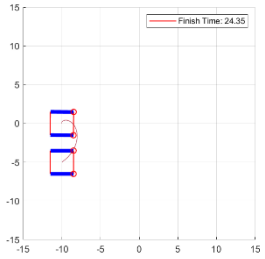

omega
v

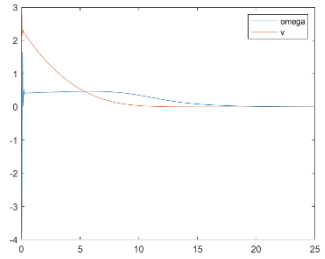

x
y
theta

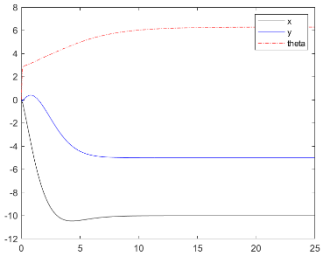

Finish Time: 25.15

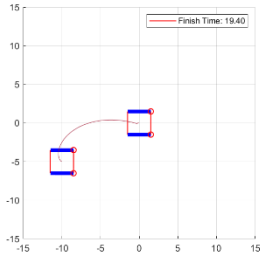

omega
v

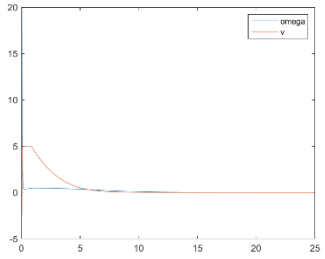

x
y
theta

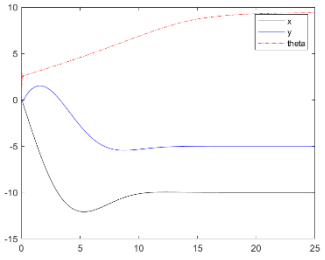

Finish Time: 24.35

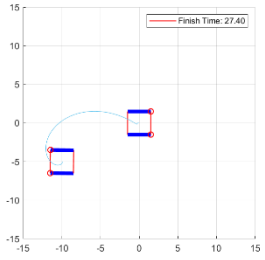

omega
v

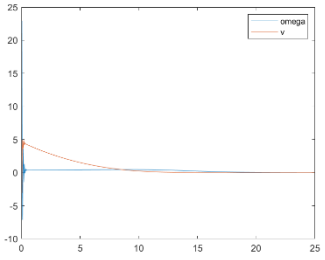

x
y
theta


Finish Time: 19.40


omega
v


x
y
theta


Finish Time: 27.40


omega
v

References

[1] K. Lynch, F. Park, and Northwestern University, “13.3.1. Modeling of Nonholonomic Wheeled Mobile Robots.” https://modernrobotics.northwestern.edu/nu-gm-book-resource/13-3-1-modeling-of-nonholonomic-wheeled-mobile-robots/ (accessed Dec. 31, 2022).