Matt Ritter, Jay Gorasia, Leslie Velez
Dynamics
Application Problem 2
November 21, 2008

Roomba Robot Simulation

Introduction

The goal of this application problem is to model a Roomba robot as a two dimensional rigid body with spatial extent, verify the model we develop, and apply inverse kinematics to get the Roomba to follow a specified trajectory. We use a free body diagram of the Roomba, while applying what we have learned about coordinate frame transforms, to develop a set of governing equations. From there we use the governing equations, and other necessary relationships, to create a simulation in Python. The model is verified through this simulation by looking at the displacement and trajectory of the Roomba. We conclude with inverse kinematics where we control the Roomba's motion from the end path we want the Roomba to follow.

The Model

In order to accurately abstract the Roomba and create a valuable model, we began by developing an understanding of the kinematics of the Roomba. We did this by drawing a free body diagram and identifying our coordinate frames (Figure 1). The motion of the Roomba required that we make use of two reference frames. Both are in rectangular coordinate systems. The first is the stationary frame X-Y, with Z coming out of the page (as per the right hand rule). The second is the t-n frame, indicating the tangential and normal directions, which captures the rotation of the Roomba relative to the stationary frame. The origin of X-Y is fixed and set at an arbitrary location, while the origin of t-n is fixed to the center of the Roomba (center of mass) and moves relative to the X-Y coordinate frame. This frame is stationary relative to the wheels. Since the Roomba can only move forward and rotate, the combination of these two frames is satisfactory in capturing the overall motion.

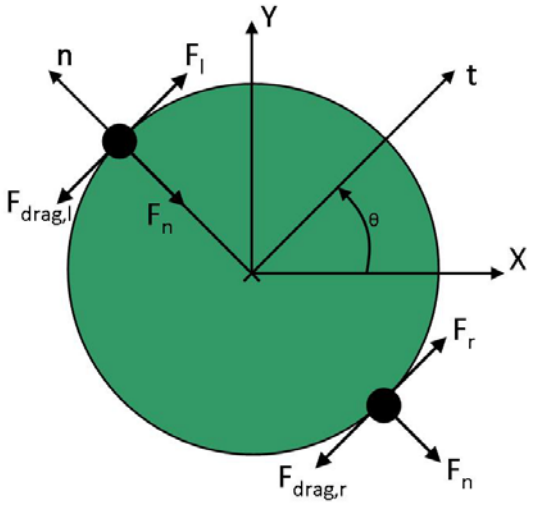


Figure 1: This figure shows the free body diagram of the Roomba along with the necessary reference frames. There are two rectangular reference frames used to capture the Roomba's motion: X-Y and t-n. X-Y is stationary and t-n rotates and translates as the Roomba moves. The angle theta is used to control the rotation of the

reference frame t-n, and with that, the forward direction of the Roomba. For the direction of forces, we are taking velocity to be positive in the positive X and Y-directions.

Governing Equations

Using the free body diagram and the selected reference frames (Figure 1), we were able to determine the kinetics of the Roomba, which are presented in the form of the equations of motion. Since we are modeling the Roomba in two dimensions, the Roomba can be thought of as a thin disk, with center of mass at the origin of t-n. Applying the kinetics by summing the forces and moments, leads to three governing equations: forces in the X-direction (1), forces in the Y-direction (2), and moments about the center of the Roomba (3). We initially derived the equations in the t-n frame, and then used coordinate frame transforms to express them in the X-Y frame. This allows us to look at the Roomba from the stationary frame and understand both its translation and rotation.

Sum of Forces:

$$m\ddot{x} = \left(F_l + F_r + F_{drag,r} + F_{drag,l}\right)\cos\theta - 2F_n \sin\theta \qquad (1)$$

$$m\ddot{y} = \left(F_l + F_r + F_{drag,r} + F_{drag,l}\right)\sin\theta + 2F_n \cos\theta \qquad (2)$$

Sum of Moments:

$$I\ddot{\theta} = (F_r - F_l - F_{drag,r} + F_{drag,l})r \qquad (3)$$

It is important to note that the sign of the forces due to drag on each of the wheels, as well as the normal force, is positive in the equations listed above but these forces do act as indicated in the free body diagram. The free body diagram was used as an aid to determine the governing equations and gain a qualitative understanding of how the Roomba moves. However, the equations are used in our simulation so, in an effort to keep the equations consistent and let the equations generate the appropriate signs, we have included the direction of the drag forces as negatives in our constitutive relationships (Equations (7)-(10), to follow).

The factor of 2 in the last term of Equations (1) and (2) comes from the inclusion of the normal force from each wheel of the Roomba. The forces are the same on each wheel in terms of both magnitude and direction.

Additional Equations & Constitutive Relationships

In order to use the governing equations in a simulation, we must put them in a form where we know the value of each term. The force on each wheel is known, as we are prescribed a value for the maximum force. Including our constitutive relationships is the last step in abstraction and allows us to write the equations of motion in known terms.

For use in our simulation, we have determined additional equations that make use of known values and allow us to define unknown values in our equations of motion. The velocity of the Roomba in the n-direction, as well as the velocity of each wheel, is presented in Equations (4), (5), and (6). Each of these equations has been developed using coordinate frame transforms. This enables us to express the velocities with respect to the stationary X-Y frame.

$$V_n = -\dot{x}\sin\theta + \dot{y}\cos\theta \quad (4)$$

$$V_l = \dot{x}\cos\theta + \dot{y}\sin\theta - \dot{\theta}r \quad (5)$$

$$V_r = \dot{x}\cos\theta + \dot{y}\sin\theta + \dot{\theta}r \quad (6)$$

The first constitutive relationship is for the mass moment of inertia. This equation (7) is for a circular disk or plate rotating about its z-axis, which runs through the center of mass of the disc. It assumes that the disk is of constant density and, thus, also that the center of mass is at the center of the disk.

The constitutive relationships for force of drag on the wheels, and also the normal force on the wheels, involve the velocity of the wheel in a specific direction, along with a prescribed or determined coefficient of drag. These relationships stem from a quadratic drag model for viscous drag. Equation (8) shows the relationship for the normal force. This is necessary to ensure minimal translation in the n-direction. This coefficient is arbitrarily set at a value high enough to render this translation insignificant. Equations (9) and (10) show the relationship for the tangential drag force on each wheel. The value of the drag coefficients were prescribed as $k_n = 1000\frac{Ns^2}{m}$ and $k_{drag} = 3.4\frac{Ns^2}{m}$. The absolute value of one velocity in each of the constitutive relationships is in place to ensure that the force is acting in the direction opposite the motion of the Roomba.

Constitutive Relationships:

$$I = \frac{1}{2}mr^2 \quad (7)$$

$$F_n = -k_n V_n |V_n| \quad (8)$$

$$F_{drag,r} = -k_{drag} V_r |V_r| \quad (9)$$

$$F_{drag,l} = -k_{drag} V_l |V_l| \quad (10)$$

We determined the final governing equations by inserting the constitutive relationships. Equation (11) governs the angular acceleration, while Equations (12) and (13) specify the linear acceleration in the X-direction and Y-direction, respectively.

Final Equations of Motion:

$$\ddot{\theta} = \frac{2\left[F_r - F_l - \left(k_{drag}\left(V_r|V_r| + V_l|V_l|\right)\right)\right]}{mr} \quad (11)$$

$$\ddot{x} = \frac{\left[F_l + F_r - \left(k_{drag}\left(V_r|V_r| + V_l|V_l|\right)\right)\right]\cos\theta - 2\left(-k_n V_n|V_n|\right)\sin\theta}{m} \quad (12)$$

$$\ddot{y} = \frac{\left[F_l + F_r - \left(k_{drag}\left(V_r|V_r| + V_l|V_l|\right)\right)\right]\sin\theta + 2\left(-k_n V_n|V_n|\right)\cos\theta}{m} \quad (13)$$

The final governing equations can be expressed in the state-vector form, which allows for easy manipulation within the computer simulation. This form is presented in Equation (14). The variable we have selected to represent our vector is X. Vector X contains information about the position (x , y , θ) and velocity ($\dot{x}$, $\dot{y}$, $\dot{\theta}$) of the Roomba.

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} X[1] \\ \ddot{x} = \left[\left[F_l + F_r - \left(k_{drag}\left(V_r|V_r| + V_l|V_l|\right)\right)\right]\cos\theta - 2\left(-k_n V_n|V_n|\right)\sin\theta\right]/m \\ X[3] \\ \ddot{y} = \left[\left[F_l + F_r - \left(k_{drag}\left(V_r|V_r| + V_l|V_l|\right)\right)\right]\sin\theta + 2\left(-k_n V_n|V_n|\right)\cos\theta\right]/m \\ X[5] \\ \ddot{\theta} = \left[2\left[F_r - F_l - \left(k_{drag}\left(V_r|V_r| + V_l|V_l|\right)\right)\right]\right]/mr \end{bmatrix} \quad (14)$$

Model Verification

In order to verify our model, we wrote a computer simulation using Python. We did this within the PyLab environment, which allowed us to use several modules valuable modules to run our simulation. Our simulation uses an ODE solver with a slope function. The slope function takes inputs from our state space and returns the first derivative (Jacobian matrix) of our variables. The simulation includes logic statements which use time to vary forces. We send it a time and it applies the proper forces to the left and right wheels of the Roomba model. Our code commented code is provided in the Appendix.

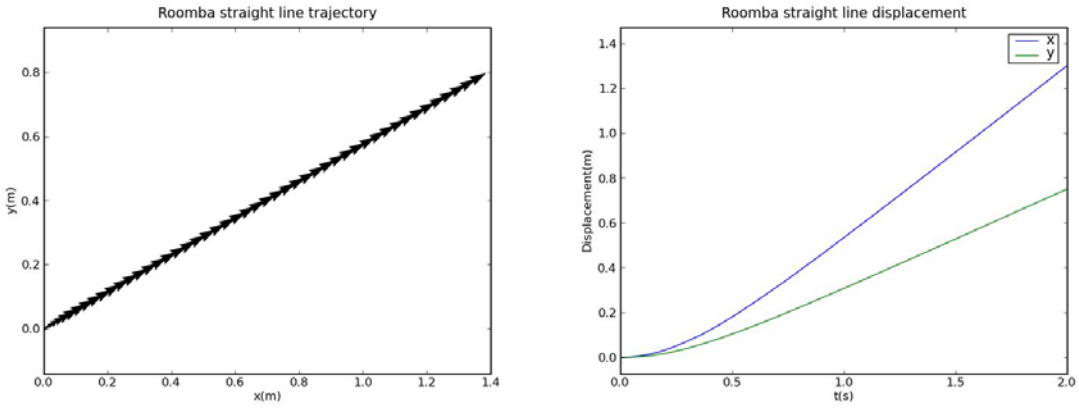
To verify that our model was working appropriately, we simulated two simple trajectories. Simple simulations allow us to compare results with our intuition to find bugs in the code and errors in our equations of motion.

Straight Line Trajectory

The first simple trajectory is a straight line motion. The Roomba follows a straight line for 2 seconds as can be seen in Figure 2a. The Roomba starts at the origin of the X-Y reference frame (0m, 0m), and moves to (1.4m, 0.8m). The arrows indicate are proportional to the magnitude of the velocity of the Roomba at different locations within its trajectory. As expected, the arrows close to (0m, 0m) are small and get larger quickly as the Roomba reaches its cruising velocity. During the initial movement, the Roomba must overcome friction. The input generating this movement is the constant maximum force ($F_{\max} = 2.7N$) applied to each wheel.

When the simulation stops running (after 2 seconds), the Roomba's velocity is still the same as it was the moment immediately before. The Roomba has not been slowed at all, so the straight line trajectory concludes with an arrow of the same size as the moment before.

Figure 2b illustrates the displacement of the Roomba in the X and Y directions over time. The movement in both directions shows slight curvature initially which corresponds to the increase in velocity of the Roomba.



(a) Trajectory (COM) (b) Displacement (COM)

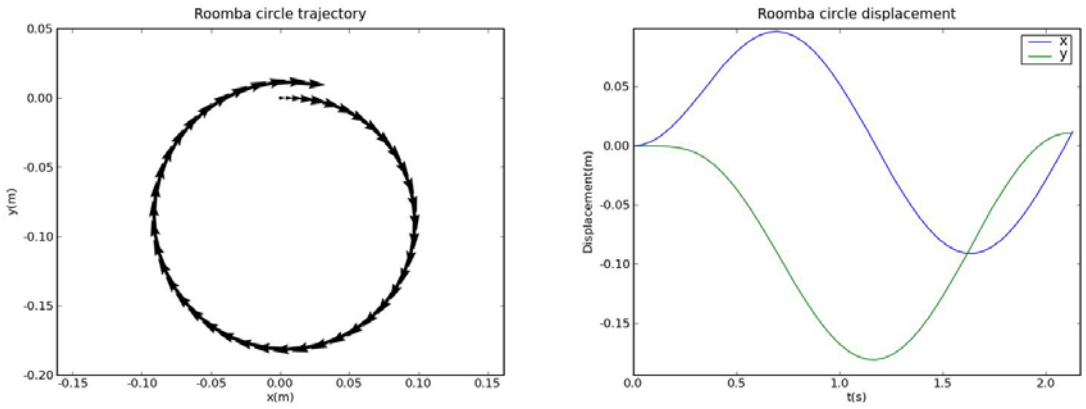
Figure 2: This figure shows the straight line trajectory within the X-Y reference frame (a) and the displacement over time (b) of the Roomba. The Roomba started off from rest with an initial velocity of zero and a 30° heading. A constant force of 2.7 N was applied at both wheels. The arrows in (a) indicate the magnitude and direction of the velocity. In (b), the blue line shows the displacement in X and the green line shows the displacement in Y.

Circular Trajectory

Another simple trajectory used to verify the accuracy and functionality of our simulation is that of a circle. To do this, we input a force of 2.7 N to just the left wheel, while the right wheel received no input force. This example tests the behavior of the model and helps us to verify that the model is obeying kinematic constraints. The trajectory and displacement of the center of mass of the Roomba as it moves in a circle is shown in Figure 3.

The arrows in (a) are proportional to the magnitude and direction of the velocity. The Roomba starts off with a velocity of 0 m/s, and increases to a constant velocity as it completes the trajectory. The arrows also show that the heading of the Roomba is tangent to the circle at all times. This is a meaningful way for us to verify that the kinematic constraints within the equations of motion are included correctly and that our model is functioning appropriately.

Plot (b) shows the displacement of the center of mass in both the X-direction and the Y-direction over time. This is helpful in verifying that the model behaved as expected and did in fact follow the trajectory as indicated in (a). When approximately 0.6 seconds have elapsed, the Roomba is at about (0.9m, -0.9m) as seen in (a). This is the maximum displacement from the origin in the X-direction, which means that it should be the maximum peak point in plot (b). Looking at plot (b), this is confirmed. The blue line (displacement of X) peaks at about 0.9m, while the green line (displacement of Y) is at about -0.9 m. It takes the Roomba just over 2 seconds to complete the circular trajectory.



(a) Trajectory (COM) (b) Displacement (COM)

Figure 3: This figure shows the circular trajectory within the X-Y reference frame (a) and the displacement over time (b) as the Roomba moves in circle. The Roomba started off from rest with an initial velocity of zero. A constant force of 2.7 N was applied at the left wheel, while the right wheel received no input force. The arrows in (a) indicate the magnitude and direction of the velocity. In (b), the blue line shows the displacement in X and the green line shows the displacement in Y.

One additional way to verify that the heading is always tangent to the trajectory is to plot the angle (θ) against time. As can be seen in Figure 4, the Roomba starts off at 0 radians and progresses to 2π radians over time. The angle is negative because the Roomba is turning right and we originally defined theta positive according to the right hand rule (Figure 1). The curve at the start of the motion is due to the Roomba starting from a resting position. The data set soon becomes linear, indicating constant velocity. The illustration of the Roomba reaching a constant velocity coincides with the size of the arrows shown on the trajectory in Figure 3a.

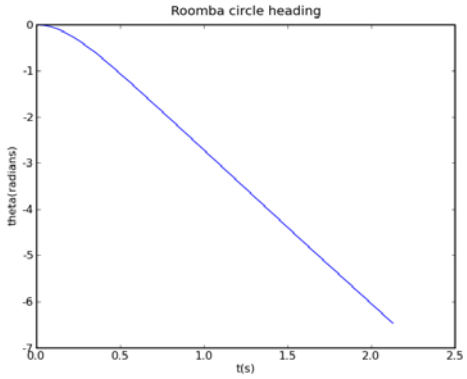


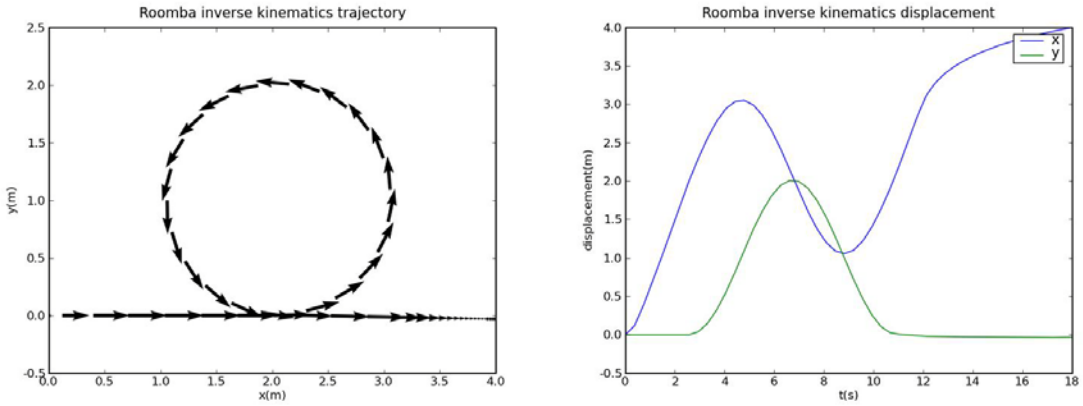
Figure 4: This plot shows the Roomba's heading over time as it moves in a circle. Moving in a complete circle means that the Roomba will move from 0 radians to -2π radians. The Roomba starts from rest, so there is a slight curve initially. However, the Roomba soon reached a constant velocity as expected.

By modeling the straight line and circular trajectories, we were able to verify that our equations of motion are correct and that our simulation behaves as it should. This is an important step in the process of creating a working simulation because it allows us to trust the simulation and its results in more complicated trials.

Inverse Kinematics

With our developed model, we can now work backwards through kinematics to get the Roomba to follow a desired trajectory. Figure 5 shows the trajectory the Roomba followed. Plot (a) shows the motion, with arrows indicating magnitude and direction of the velocity. The arrows are always tangent to the circle showing the heading of the Roomba. The Roomba moves in a straight line for 2m, then complete a 1m radius circle, and then returns to where it started the circle to continue moving forward for 2 additional meters.

Plot (b) of Figure 5 shows the displacement of the Roomba in the X and Y directions over time. The blue line indicates the motion in the X-direction, while the green line indicates the motion in the Y-direction. Initially and finally, the Roomba is just moving in the X-direction and so the Y-direction displacement is zero meters. At 2.4 seconds, the Roomba begins to move in a circle. It completes the circle at t=10.7 seconds, and continues forward again in the X-direction. After about 11.8 seconds the input force on the wheels is zero and the Roomba coasts to a stop, due to the dynamic friction. All the times were determined empirically.



(a) Trajectory (COM) (b) Displacement (COM)

Figure 5: These plots show the trajectory and displacement over time of the center of mass of the Roomba. The arrows in (a) indicate the direction and magnitude of the velocity of the Roomba through the course of its trajectory. The Roomba coasts to a stop as can be seen by the arrows getting smaller at the end of the trajectory. Plot (b) shows the displacement over time. As expected, the displacement in the Y-direction is at zero meters before and after the Roomba completed the circle.

This trajectory was input by varying the forces to the wheels at different times during the Roomba's motion. The specific piece of code that controls the motion is shown in Figure 6. The first if statement and the last two show the Roomba moving in a straight line; both wheels receive the same force. The second statement shows the circular movement. The Roomba moves counter clockwise due to a greater force input to its right wheel.

```
if t<2.4: #Move in a straight line
    Fl = 2.7
    Fr = 2.7
  elif t<10.6665: #Move in a circle
    Fl = 1.4
    Fr = 2.7
  elif t<11.84589: #Move in a straight line
    Fl = 2.7
    Fr = 2.7
  else: #drift to stop
    Fl = 0
    Fr = 0
```

Figure 6: Part of the code used to control the Roomba and satisfy the inverse kinematics problem is presented above. The forces are varied at different times within the complete trajectory so that the Roomba will respond as desired.

We were successful in using inverse kinematics to control the trajectory of the Roomba. The Roomba was able to complete this trajectory in approximately 18 seconds. We slowed the Roomba down (stopped inputting force) after about 11.8 seconds. This had an effect on the time it takes the Roomba to complete the entire trajectory. Were we to allow the Roomba to continue moving in a straight line in the X-direction after reaching 4 meters, and thus keep the constant maximum force input to the wheels, the Roomba could have completed the trajectory in about 14.3 seconds.

Conclusion

We were able to successfully create a model of a Roomba by developing the equations of motion of the Roomba in a global reference frame. We created a simulation of our model in Python and used test cases to verify that our model was accurate and that our simulation was controlling the behavior of the model appropriately. Finally, we were able to use inverse kinematics to specify the trajectory of the Roomba.

Reflection and Learning

We learned a lot from completing this application problem. We were able to apply kinetics, kinematics, and constitutive relationships to determine the equations of motion. The application problem is a fairly general use of rigid body dynamics, so it gave us widely applicable experience in modeling simple systems.

We started with a free body diagram and clearly indicated our reference frames. This allowed us to discuss all the forces on the Roomba and translate the diagram into governing equations. We used coordinate frame transfers to express the motion of the Roomba in the global reference frame, and then put the resulting equations into a simulation. The inevitable debugging process began at this point, and we iterated back through the free body diagram and governing equations until the test cases of the code worked as expected.

One of the most significant and beneficial things we learned was the importance of test cases. We were able to see how they can be helpful in verifying that the equations of motion are correct and also that our code functions properly. This is really important in verifying that the model obeys the kinematic constraints.

Doing the simulation in Python worked out well. There was a learning curve that had to be overcome. However, we were able to learn how to use Python in the PyLab environment and gain experience in working with this language. We feel that this will be a valuable skill for us in our future endeavors.

We feel that we have also improved in the area of technical writing. We worked to clearly illustrate our process to the problem, conclusions, and learning. We used equations, figures, and plots to supplement our explanations. This application problem was a great way for us to use tools we have acquired in the Dynamics Course and become more confident in using them with complicated problems.

Appendix

Simulation Code

```
# Application problem 2
# Roomba simulation
# November 20 2008
# by J. Gorasia, M. Ritter, L. Velez

from numpy import *
from pylab import *
from scipy.integrate import odeint
import math

# Initial Conditions
# The "state" of the system is {x, \dot(x),y, \dot{y},theta,\dot{theta}}^T
x0 = zeros( 6 )
#x0[4] =   math.radians(30)  # heading in radians

# Time Bounds
t0 = 0.0                #Initial time [s]
tF = 18                 #Final time [s]
tt = linspace(t0,tF,50)     #Time array for use in ode solver

# Roomba information
d = 0.33                #Diameter [m]
r = d/2.0               #Radius [m]
m = 2.7                 #Mass [kg]
I  = (0.5*m)*(r**2)         #Inertia about the center, assuming it is like a disc [kgm^2]
f_max = 2.7              #maximum output force of the wheels [N]
kdrag = 3.4             #drag tangential to motion of the wheels [N/(m/s)^2]
kn = 1000               #drag normal to motion of the wheels [N/(m/s)^2]

def sign(x):
   """Returns the sign of the input value"""
   if x < 0: return -1
   elif x > 0: return 1
   else: return 0

def dX_dt(X, t=0):
   """
   Return the Jacobian (first-derivative) of the state.
   Both the input and output need to be arrays
   """
   #Vary input force to vary trajectory
   if t<2.4:            #Move in a straight line
      Fl = 2.7
```

```
        Fr = 2.7
    elif t<10.6665:         #Move in a circle
        Fl = 1.4
        Fr = 2.7
    elif t<11.84589:        #Move in a straight line
        Fl = 2.7
        Fr = 2.7
    else:                   #drift to stop
        Fl = 0
        Fr = 0

    vl = X[1]*cos(X[4])+X[3]*sin(X[4])-X[5]*r         #velocity at the left wheel
    vr = X[1]*cos(X[4])+X[3]*sin(X[4])+X[5]*r         #velocity at the right wheel
    vn = -X[1]*sin(X[4])+X[3]*cos(X[4])               #normal velocity of the Roomba

    Fdrag = kdrag*(vr*abs(vr)+vl*abs(vl))             #tangential drag force for both wheels
    Fnorm = kn*vn*abs(vn)                             #force normal to the Roomba
    Ftangent = Fl + Fr - Fdrag                        #force tangential to the Roomba

    xdot = X[1]                                       #velocity in x direction
    xddot = (Ftangent*cos(X[4])+2*sin(X[4])*Fnorm) / m #acceleration in x direction
    ydot = X[3]                                       #velocity in y direction
    yddot = (Ftangent*sin(X[4])-2*cos(X[4])*Fnorm) / m #acceleration in y direction
    thetadot = X[5]                                   #angular velocity
    thetaddot = (Fr - Fl - kdrag*vr*abs(vr) + kdrag*vl*abs(vl))/(0.5*m*r) #net torque

    dX = [xdot,xddot,ydot,yddot,thetadot,thetaddot]    #form the Jacobian

    dXa = array(dX)
    dXa = dXa.transpose()
    return dXa

def plot_data(XX,type=1):
    """Plot the results from the solution of the ODE
    type=1 : Trajectory
    type=2 : Displacement
    type=3 : Heading
    type=4 : Information about the movement of the Roomba"""
    if type == 1:
        quiver(XX[:,0],XX[:,2],XX[:,1],XX[:,3])
        xlabel('x(m)')
        ylabel('y(m)')
        title('Trajectory of the Roomba')
        axis('equal')
        show()
    elif type == 2:
```

```
        plot(tt,XX[:,0],label='x')
        hold(True)
        plot(tt,XX[:,2],label='y')
        legend()
        xlabel('t(s)')
        ylabel('displacement(m)')
        title('Displacement of the Roomba')
        show()
    elif type == 3:
        plot(tt,XX[:,4],label='theta')
        xlabel('t(s)')
        ylabel('theta(radians)')
        title('Heading of the Roomba')
        show()
    elif type == 4:
        width =  max(XX[:,0])-min(XX[:,0])
        center = (max(XX[:,0])+min(XX[:,0]))/2.0
        height = max(XX[:,2])-min(XX[:,2])
        print "Width: %f, Height: %f, Center: %f " %(width, height, center)

XX,infodict = odeint(dX_dt, x0, tt, full_output=True)     #Solve the ODE
plot_data(XX,1)                                  #plot the data
```