Name: Tahsincan
Surname: Köse
ID: 2188423

Homework 3 - Jacobians and Mobile Robots

1

It is straightforward to derive FK. Starting from the origin, applying all transformations in-order until the end effector tip would suffice:

${}^{O}T_E = T_x(d_1) \times R_z(\theta_2) \times T_x(d_2)$, this is the compact form. First we translate along the prismatic joint with d_1 units, then rotate around z axis of θ_2 degrees and finally apply the constant d_2 translation along the new x axis.

$$
{}^{O}T_E = \begin{bmatrix} 1 & 0 & d_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 \\ sin(\theta_2) & cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{O}T_E = \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & d_1 + d_2cos(\theta_2) \\ sin(\theta_2) & cos(\theta_2) & d_2sin(\theta_2) \\ 0 & 0 & 1 \end{bmatrix}
$$

In order to compute Jacobian, partial derivatives with respect to d_1 and θ_2 must be derived from the ${}^{O}T_E$ found above.

Compute $\frac{d}{d_t}\begin{bmatrix} {}^{O}t_E \\ \theta \end{bmatrix}$, where $\begin{bmatrix} {}^{O}t_E \\ \theta \end{bmatrix} = \begin{bmatrix} d_1 + d_2cos(\theta_2) \\ d_2sin(\theta_2) \\ \theta_2 \end{bmatrix}$, however it is not necessary to compute the derivative of angular part in this specific problem. Through ignoring it, we have the 2×2 Jacoban matrix.

Result is: $\frac{d}{d_t}\begin{bmatrix} {}^{O}t_E \\ \theta \end{bmatrix} = \begin{bmatrix} 1 & -d_2sin(\theta_2) \\ 0 & d_2cos(\theta_2) \end{bmatrix}$

2

In order to construct a 2D robot, I used ETS2 class of the toolbox. It is pretty easy to construct a 2D robot in terms of *Denavit-Hartenberg* parameters:

```
1  E = Tx('d1') * Rz('q2') * Tx(d2);
```

Above code line constructs a PR robot that has 2 joint variables. The rest of the code can be seen in **hw3_script1.m** file. After implementing the function which takes specified values as its inputs, I made a bunch of experiments. Below are the most useful ones which have their particular motion of each joint and end-effector with respect to the given inputs:

You might want to inspect the inputs given. See ***Table 1***

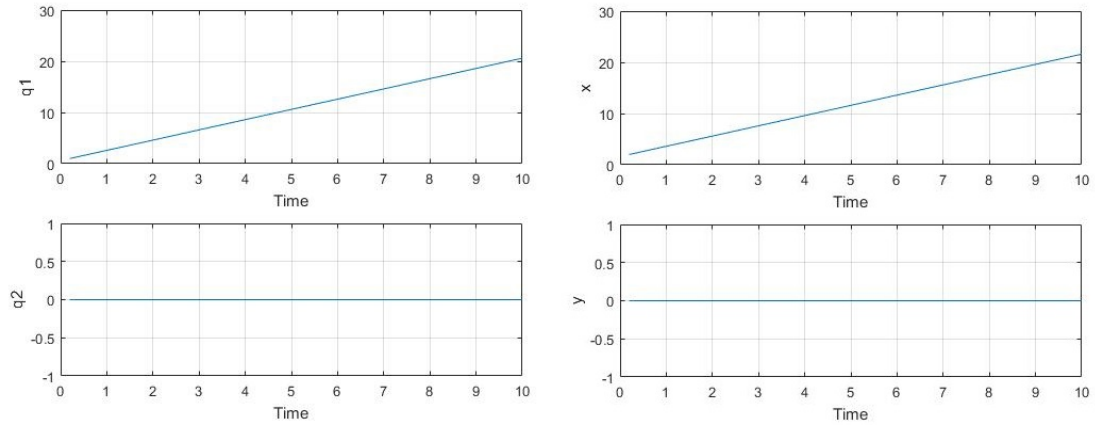


Figure 1: Experiment 1

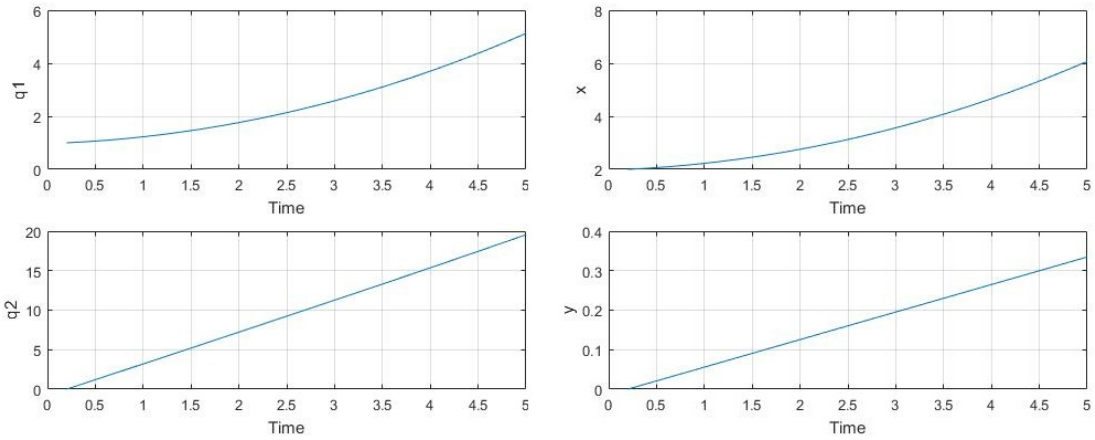


Figure 2: Experiment 2

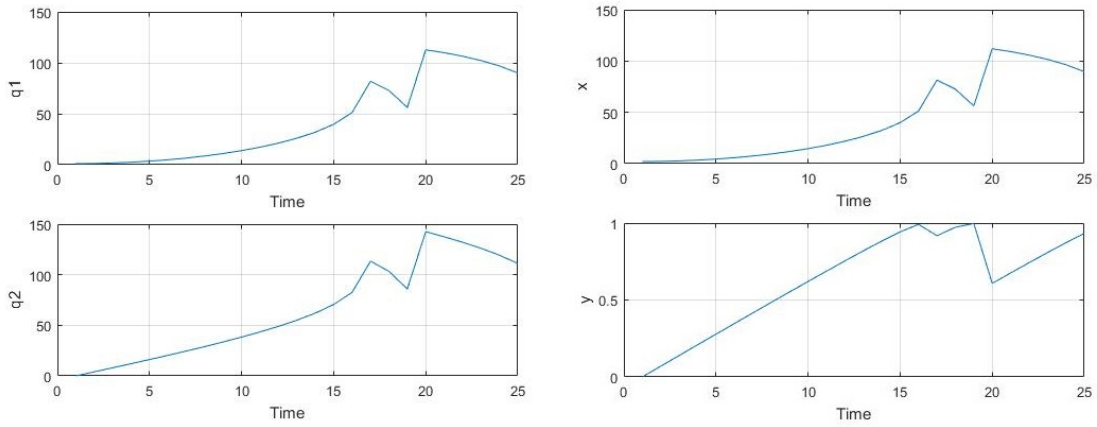


Figure 3: Experiment 3

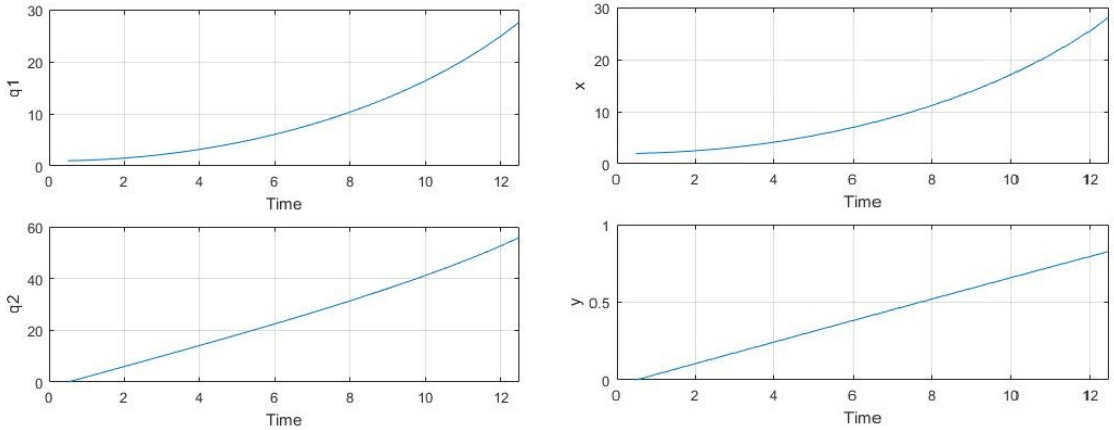


Figure 4: Experiment 4

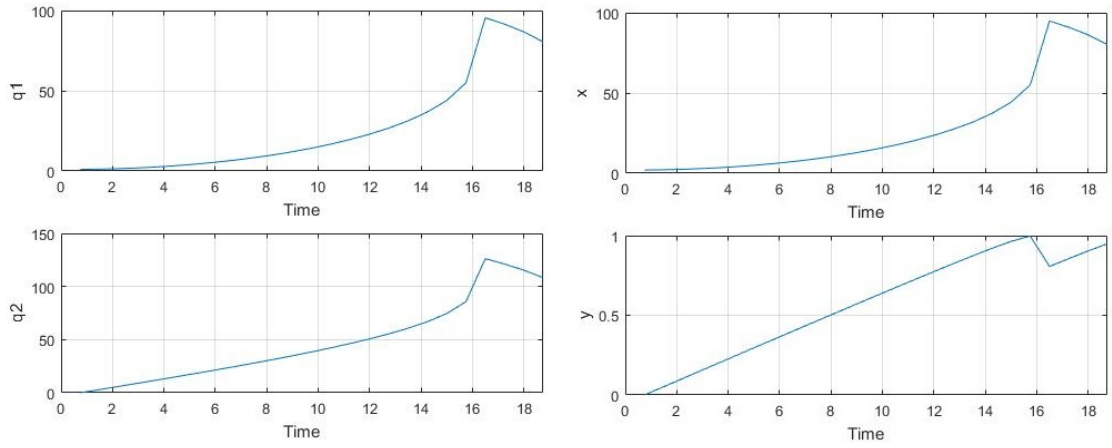


Figure 5: Experiment 5

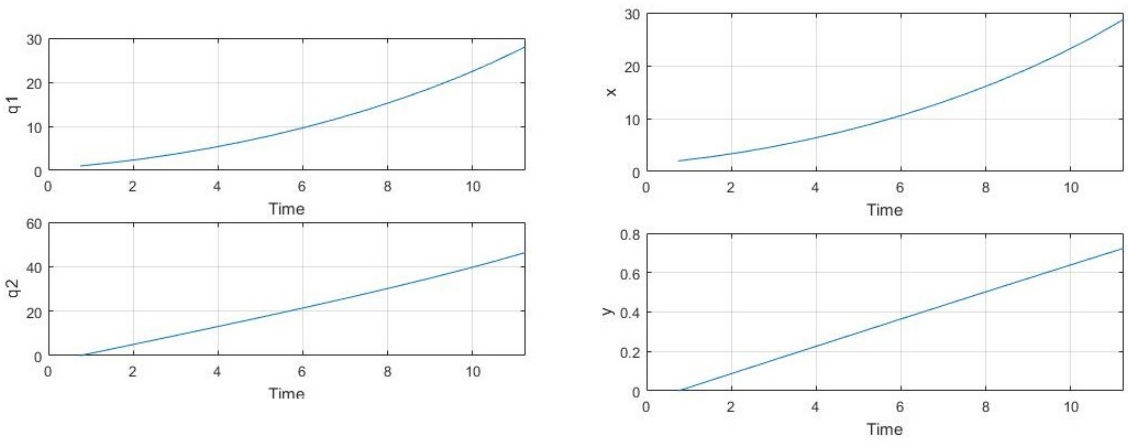


Figure 6: Experiment 6

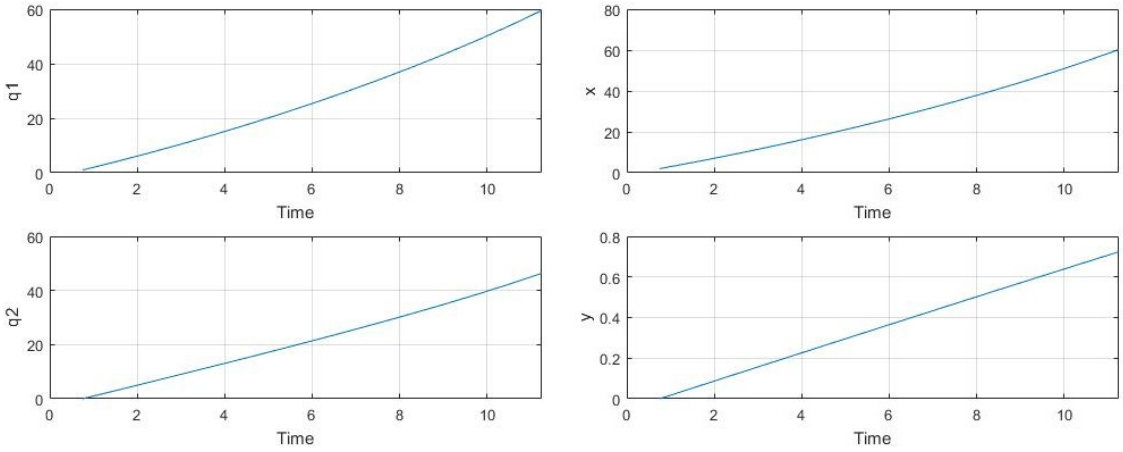


Figure 7: Experiment 7

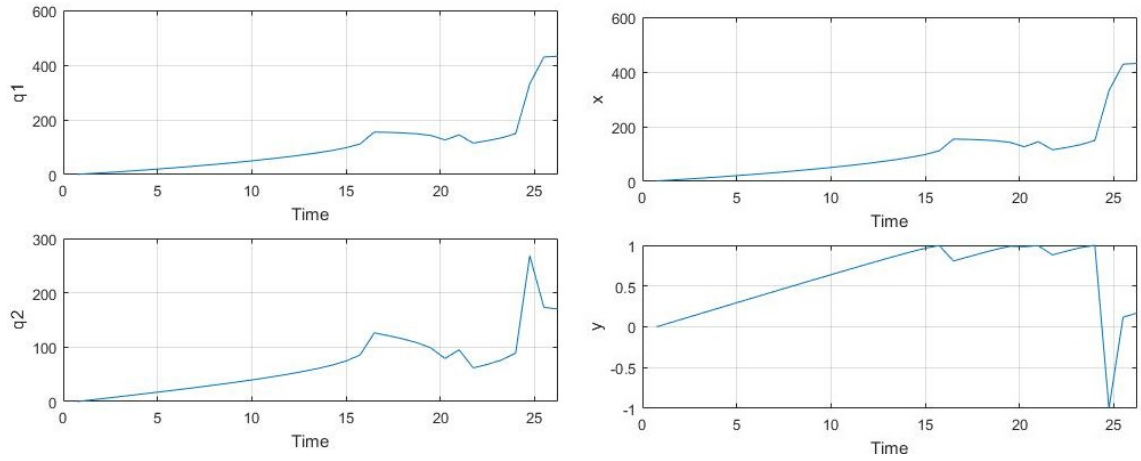


Figure 8: Experiment 8

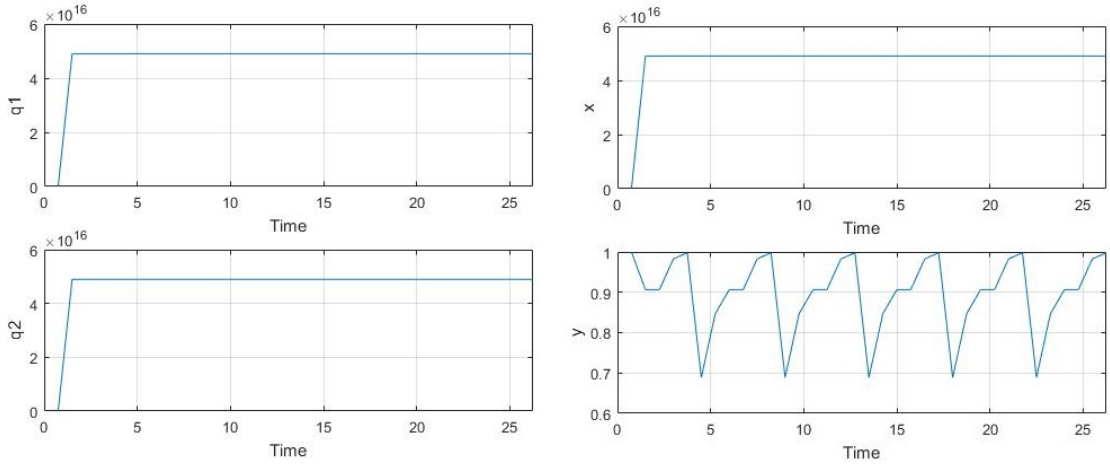


Figure 9: Experiment 9

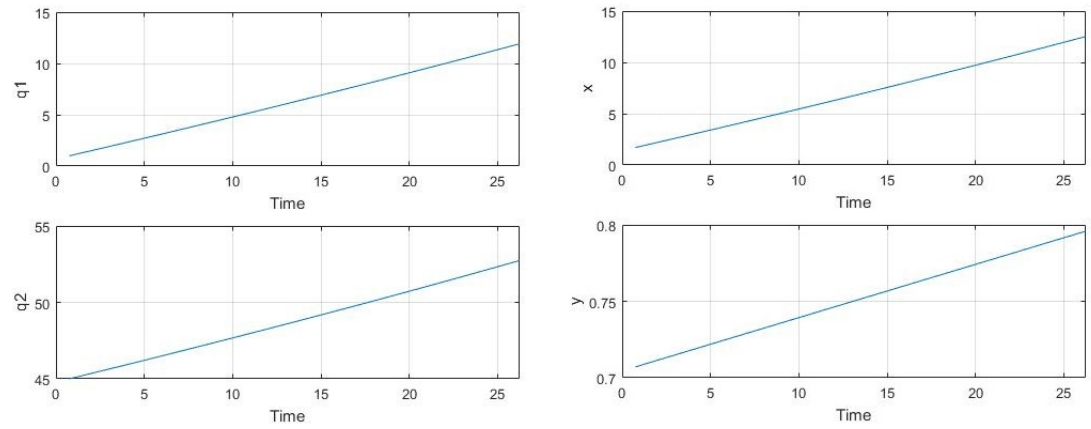


Figure 10: Experiment 10

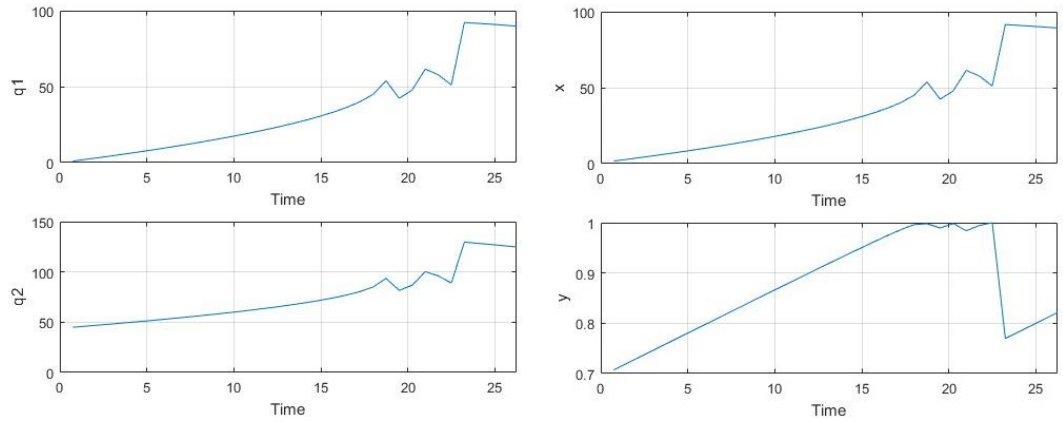


Figure 11: Experiment 11

Experiment No.	$\hat{v}$	q_{init}	n	Δ
1	[2 0]	[1 0]	50	0.2
2	[0.2 4]	[1 0]	25	0.2
3	[0.2 4]	[1 0]	25	1
4	[0.2 4]	[1 0]	25	0.5
5	[0.2 4]	[1 0]	25	0.75
6	[1 4]	[1 0]	15	0.75
7	[4 4]	[1 0]	15	0.75
8	[4 4]	[1 0]	35	0.75
9	[4 4]	[1 90]	35	0.75
10	[0.2 0.2]	[1 45]	35	0.75
11	[0.5 1]	[1 45]	35	0.75

Table 1: Input Configurations - 1

There are many deductions to be made. To start with, let's compare Experiments 2,3,4 and 5. Only different input in those experiments is Δ. As Δ increases, revolute joint becomes closer and closer to 1. It is in fact 90°, which is a singularity point for the robotic arm. In detail, its Jacobian matrice becomes degenerate and looses 1 rank. The ultimate consequence of this phenomenon is that the robot is no longer able to follow the specified vector.

Another comparison that results in a meaningful inference is between the Experiments 1 and 10. If you inspect a little more, you see that the initial value of the revolute joint equals to $atan2(v_y, v_x)$, which implies horizontal motion direction. And in only those experiments, a perfectly straight line motion could be constructed.

We can conclude that all configurations that not satisfy the above condition will deviate to the singularity at some point and will not be able to follow the specified line any longer. Furthermore, in Experiment 9, initial value of revolute joint is directly given as 90°, therefore the robot cannot follow the line from the beginning and applies enormous rates on its joints. Those sudden changes and discontinuities are the direct result of this loss of manipulability.

3

For this question, I don't want to introduce overhead with ETS2 class. Moreover, it has subtle problems in Linux systems. Therefore, I solely calculated the pose and joint configuration data. Subsequent plots will illustrate the findings better.

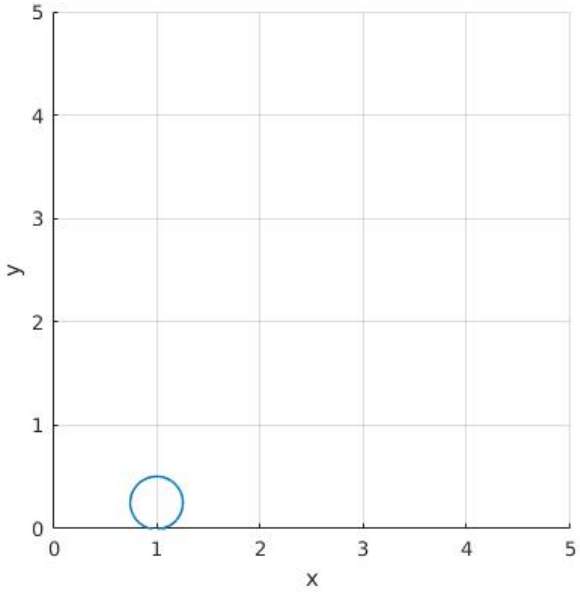
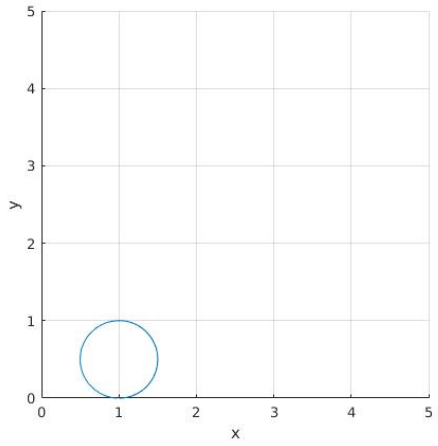
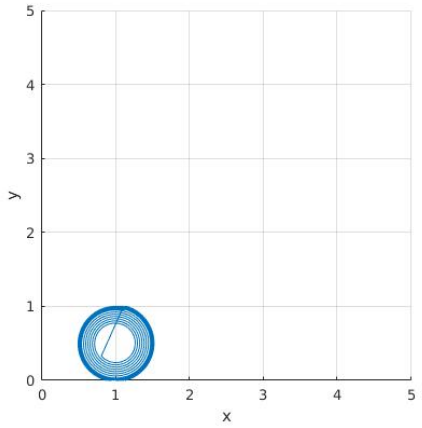


Figure 12: Experiment 12



(a) Experiment 13.a



(b) Experiment 13.b

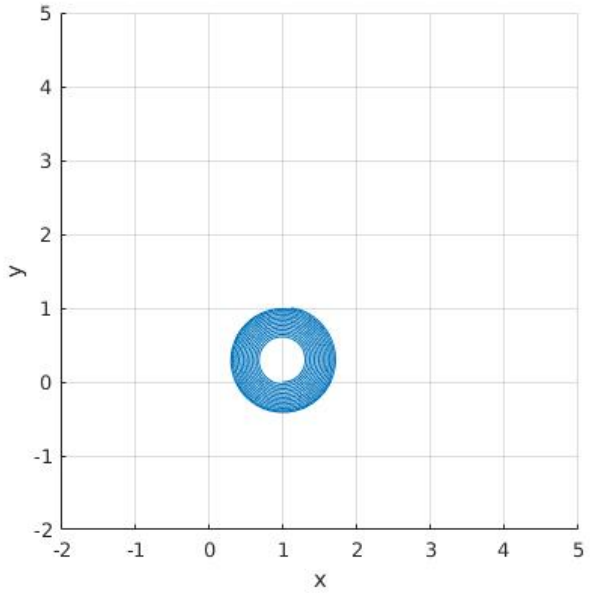


Figure 14: Experiment 14

Experiment No.	r	c	n	Δ
12	[1 0.25]	0.2	5000	0.001
13a	[1 0.5]	0.5	5000	0.001
13b	[1 0.5]	0.5	5000	0.01
14	[1 0.3]	0.3	5000	0.001

Table 2: Input Configurations - 2

Let's start with the succesful one, which is Experiment 12. You can see the input configurations from the table above. The circle is deliberately chosen to be entirely valid for the robot. Consequently, the robot followed a path that is a very good approximation of the circle specified. In fact, this approximation is the direct result of the Jacobian manipulation. Since we are dealing with velocities, but not Poses, restricts us in such circular paths. In order to successfully follow the path, one must sample time as much as possible - that is, Δ should be small. One can reach that conclusion directly through inspecting Experiments 13a and 13b. Their only difference is the value of Δ. Since it is bigger in the latter experiment, the robot becomes unable to follow the circular path within a reasonable amount of error rate and its trajectory becomes more like a spiral. In the former, that particular phenomenon does not occur just because of the smaller Δ value. Another implication of the experiments up until now that as long as we close to the continuity, we become more skillful and able in such continuous paths. Experiment 14 is just another example to that implication.

In conclusion, one can end up with the idea that this type of controller would not be suitable for such particular motions. Even though we avoid the Inverse Kinematics computations, we are overheaded or more like restricted with the characteristics of vectors. Vectors might overshoot the precise trajectory. In an open-loop system like this, that error rate will accumulate dramatically and end results will be much more drastic than the Kinematics controller. Since Kinematics controller compute each sampled Pose independently, even an error is introduced for a particular Pose, the next computation will not care about that error and operate independently all along. This is the trade-off between choice of Jacoiban and Kinematics in terms of Controllers.

4

This question requires nothing more than the simple geometry.
After inserting x_F and y_F onto the vehicle the angle between the rolling direction of A (rd_A from now on) and y_f becomes 30°. This observation directly leads us to the angle between rd_A and x_F, which is 60°. Accounting θ as well, we can directly compute p_A, p_B and p_C. I will only calculate p_A in detail. The rest can be easily computed through the change of sign and angle substitution.

Let's consider θ for now. If it is 30°, then wheel A is directly on top of the center and its position $p_A = [x, y + D]$. We should generalize this into a formula. Through a little inspection, one can immediately figure out the formula for p_A:

$$p_A = \begin{bmatrix} x + D * cos(60 + \theta) \\ y + D * sin(60 + \theta) \end{bmatrix}$$

Other wheels also have similar formulas:

$$p_B = \begin{bmatrix} x - D * cos(\theta) \\ y - D * sin(\theta) \end{bmatrix}$$

$$p_C = \begin{bmatrix} x + D * cos(60 + \theta) \\ y - D * sin(60 + \theta) \end{bmatrix}$$

5

To solve this problem, I made a research on the *Omnidirectional Vehicles*. Among many papers, one PhD dissertation[1] was particularly useful on the problem modelling.

Before starting to formulise the question in terms of corresponding wheel and rolling vectors, it is necessary to figure out α, which is the angle between rolling axis and wheel axis. $\alpha = \pi/2$ for all wheels in our case, since those are of type simple omnidirectional. Let's setup the kinematic model for wheels:

To be comfortable later on, two more shortcut notations are necessary. Rolling direction of a particular wheel will be denoted as rd_i and rolling axis will be denoted as ra_i.

$v_i =^F v +^F w\hat{z_F} \times p_i$ (1) , where p_i denotes the coordinate vectors.

$v_i = \beta\hat{rd_i} + \gamma\hat{x_F}$ (2), where β and γ are simple constants that will be replaced with meaningful values.

Multiplying both sides of (2) with $\hat{y_F}^T$ gives:

$\hat{y_F}^T v_i = \beta(\hat{y_F}^T \hat{rd_i}) + \beta(\hat{y_F}^T \hat{x_F}^T)$ (3)

Second term at right-hand side simply produces 0, because $\hat{y_F}^T$ and $\hat{x_F}^T$ are perpendicular to each other.

Multiplying again both sides of (2) with $\hat{ra_i}^T$ gives:

$\hat{ra_i}^T v_i = \gamma(\hat{ra_i^T x_F}) + \gamma(\hat{ra_i}^T \hat{rd_i})$ (4)

Note that same thing occurs for (4) as well. Second term at right-hand side is 0.

Now we can write β and γ in terms of our primitive direction vectors. Substituting those two with corresponding values from (3) and (4), in (2) gives a larger but much more useful equation:

$$v_i = \frac{\hat{y_F}^T v_i}{\hat{y_F}^T \hat{rd_i}} \hat{rd_i} + \frac{\hat{ra_i}^T v_i}{\hat{ra_i}^T \hat{x_F}} \hat{x_F} \quad (5)$$

There are some subtle observations that should be made here. The first term represents *passive rotation*. On the other hand, second term directly arises from the main

[1] **Kalman, V. (2013)**. *On modeling and control of omnidirectional wheels.* Retrieved from https://repozitorium.omikk.bme.hu/bitstream/handle/10890/1242/ertekezes.pdf

wheel rotation. In a perfect rolling situation (which is default for our vehicle), the angular velocity w_i is directly proportional to 2^{nd} term. That is:

$$\frac{r\hat{a_i}^T}{\hat{x_F}^T r a_i} v_i = w_i R \ (6)$$

One significant observation which will ease our job is the equality of $\hat{x_F}^T r a_i = -sin(\alpha)$. Using this equality, we can rewrite (6) directly as:

$$w_i R = -\frac{r\hat{a_i}^T}{sin(\alpha)} v_i$$

Finally, we can substitute v_i terms in (1) and get:

$$w_i R = -\frac{1}{sin(\alpha)} r\hat{a_i}^T \hat{v_F} + \frac{1}{sin(\alpha)} \underbrace{r\hat{a_i}^T p_i \times \hat{z_F}}\, w_F \ (7)$$

The underbraced part can be further simplified and substituted with $-p_i^T r d_i$. The most concise version of the final equation becomes:

$$w_i R sin(\alpha) = -(r\hat{a_i}^T \hat{v_F} + p_i^T r\hat{d}_i w_F) \ (8)$$

This is the general formula for any wheel. Applying it for 3 wheels at hand, produces a matrix. Also, let's get rid of funky $\hat{v_F}$ and w_F terms. They are already defined in the question part.

Having $\hat{v_F} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$ and $w_F = \dot{\theta}$, the matrix is:

$$-\begin{bmatrix} ra_{Ax} & ra_{Ay} & p_A^T r\hat{d}_A \\ ra_{Bx} & ra_{By} & p_B^T r\hat{d}_B \\ ra_{Cx} & ra_{Cy} & p_C^T r\hat{d}_C \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} w_A \\ w_B \\ w_C \end{bmatrix} * R * \underbrace{sin(\alpha)}, \text{ since } \alpha = \pi/2,$$

underbraced term is simply 1. And also, $v_i = \dfrac{-w_i * R * sin(\alpha)}{r\hat{a}_i}$ (9).

Now, whole problem is reduced to finding *position vectors, rolling direction vectors and roller axis vectors.* In fact p_A, p_B and p_C are already derived at 4^{th} question. Moreover, rolling direction vectors and roller axis vectors can be easily derived through those position vectors. All are listed below:

$$r\hat{d}_A = D * \begin{bmatrix} cos(\theta + 60) \\ sin(\theta + 60) \end{bmatrix}, r\hat{a}_A = D * \begin{bmatrix} -sin(\theta + 60) \\ cos(\theta + 60) \end{bmatrix}$$
$$r\hat{d}_B = D * \begin{bmatrix} -cos(\theta) \\ -sin(\theta) \end{bmatrix}, r\hat{a}_B = D * \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix}$$
$$r\hat{d}_C = D * \begin{bmatrix} cos(60 - \theta) \\ -sin(60 - \theta) \end{bmatrix}, r\hat{a}_C = D * \begin{bmatrix} sin(60 - \theta) \\ cos(60 - \theta) \end{bmatrix}$$

Substituting all indirect terms with their equivalent values, the final equation is below. It is indeed a linear function of $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ to w_A, w_B and w_C.

$$\begin{bmatrix} w_A \\ w_B \\ w_C \end{bmatrix} = \frac{D}{R} * \begin{bmatrix} sin(60+\theta) & -cos(60+\theta) & -(x*cos(\theta+60)+y*sin(\theta+60)+D) \\ -sin(\theta) & cos(\theta) & (x*cos(\theta)+y*sin(\theta)-D) \\ -sin(60-\theta) & -cos(60-\theta) & y*sin(60-\theta)-x*cos(60-\theta)-D \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

Finally, to compute $\dot{p_A}, \dot{p_B}$ and $\dot{p_C}$, recall (9). Regulating terms, we get each translational velocity:

$$\dot{p_A} = -\frac{1}{w_A * R} * D * \begin{bmatrix} -sin(\theta+60) \\ cos(\theta+60) \end{bmatrix}$$
$$\dot{p_B} = -\frac{1}{w_B * R} * D * \begin{bmatrix} sin(\theta) \\ -cos(\theta) \end{bmatrix}$$
$$\dot{p_C} = -\frac{1}{w_C * R} * D * \begin{bmatrix} sin(60-\theta) \\ cos(60-\theta) \end{bmatrix}$$

7

As suggested in the question description, I first experimented with the same rotational velocity case.

Input Configuration is as follows: $w = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, initial_{pose} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, t_{span} = [0\ 10]$

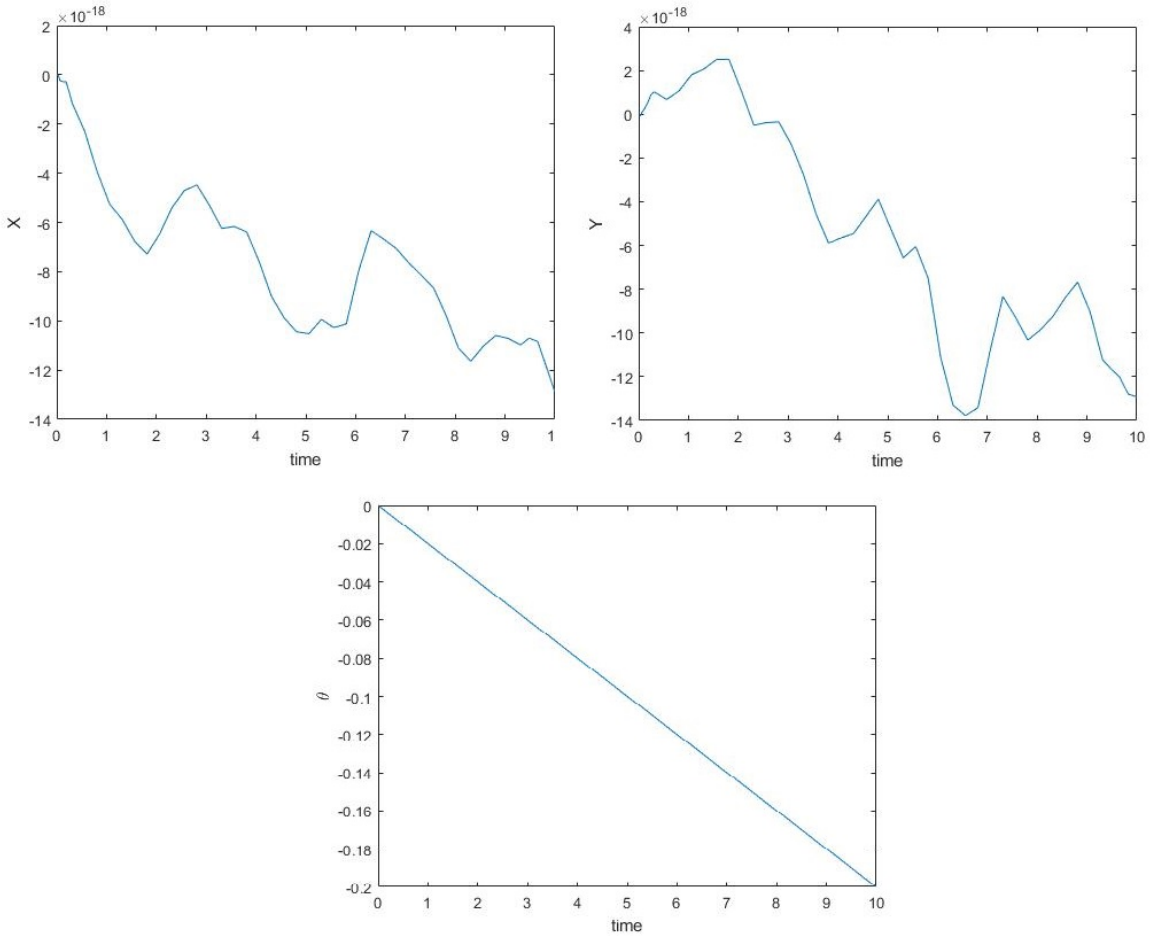


Figure 15: Same rotational velocity - Experiment 1

One direct inference is that the vehicle does not move according to X and Y frames. The plots illustrate the noisy outputs with 10^{-18} multiplier, which is probably a drawback of floating point arithmetic. In reality, those are unnoticeably minor movements and effectively possess no problem. Thus, we can say that vehicle stay stills. The more important part is the rotational motion of vehicle. It rotates constantly in the clock-wise direction. In conclusion, when all wheels have the same velocity, vehicle does not make any translational movement, but only does rotational movement. One final inference would be the rotation direction of the vehicle compared to the rotation direction of wheels.

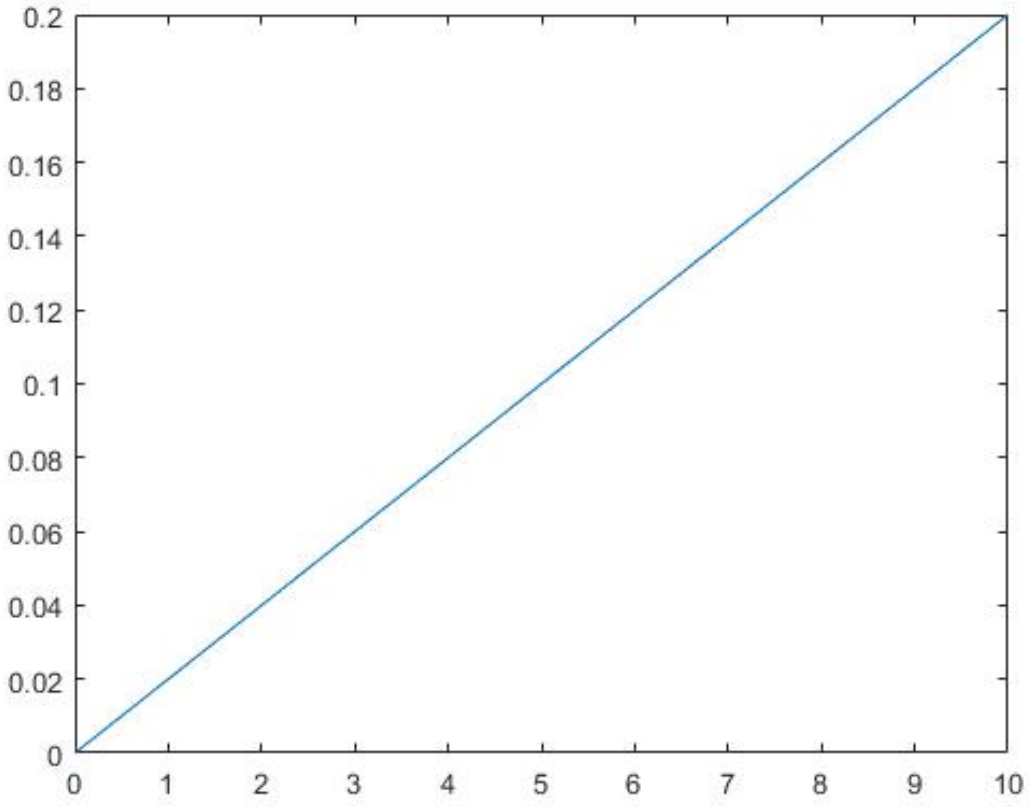


Figure 16: Same rotational velocity - Experiment 2

For the sake of clarity, only θ displacement is shown above. Only difference in the input configuration is that $w = \begin{bmatrix} -0.1 \\ -0.1 \\ -0.1 \end{bmatrix}$. Consequently, when both cases are compared with respect to each other and to their respective input configurations, vehicle does a rotational movement in the opposite direction of wheel rotations.

To continue with, I consider the case when only one wheel rotates and others stay idle.

Input Configuration is as follows: $w = \begin{bmatrix} 1.5708 \\ 0 \\ 0 \end{bmatrix}, initial_{pose} = \begin{bmatrix} 0 \\ 0 \\ 30 \end{bmatrix}, t_{span} = [0\ 50]$. θ is set as 30° in order to align wheel A with the x coordinate frame.

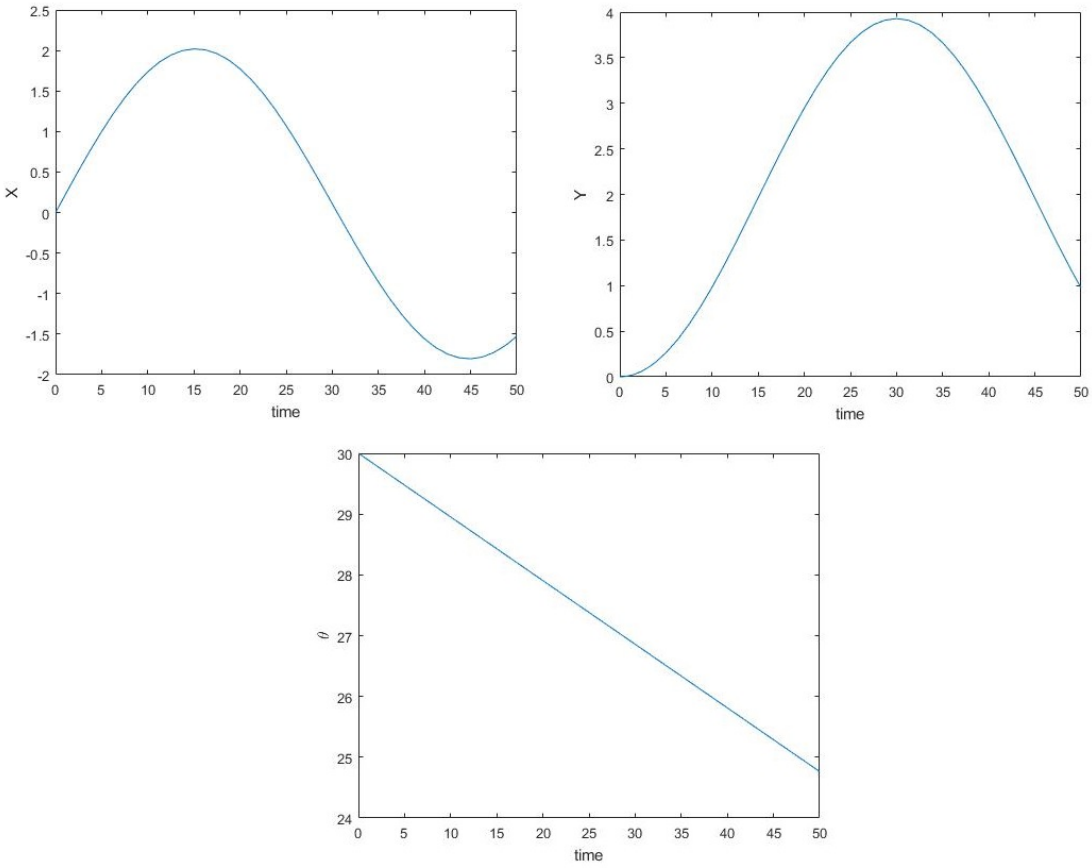


Figure 17: Wheel A Rotation Only

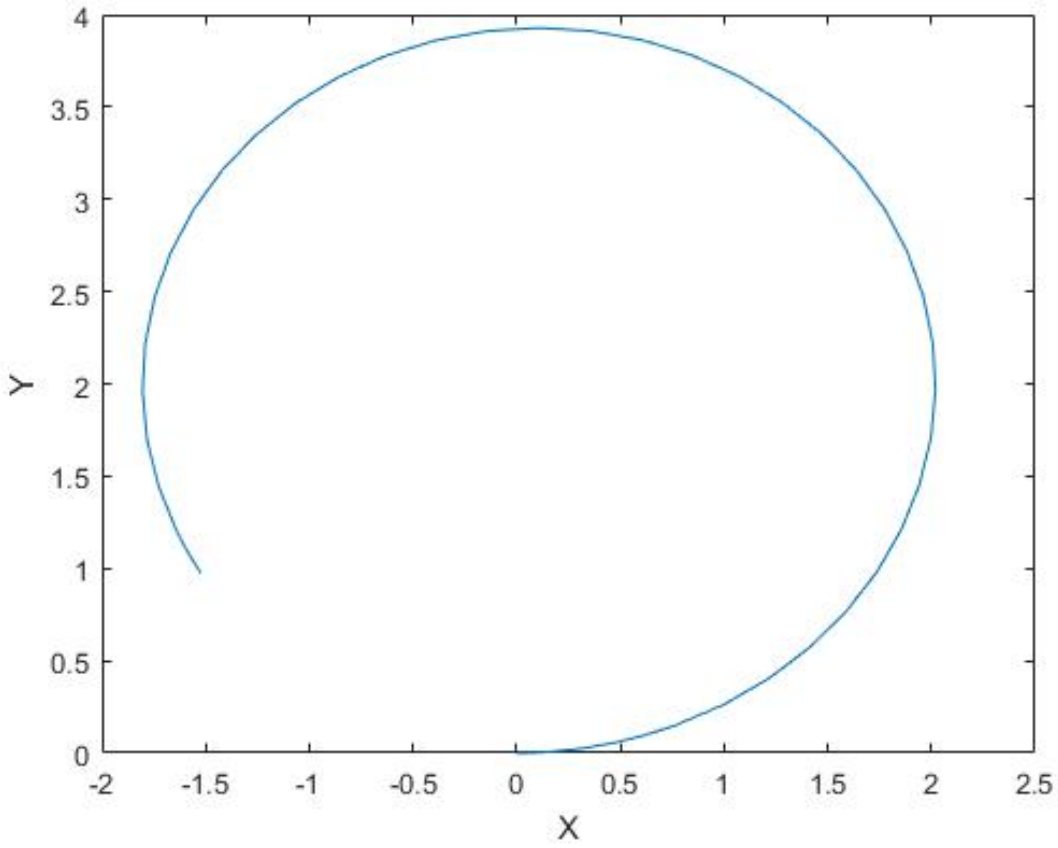


Figure 18: Wheel A Rotation Only - XY

If we look only to Figure 17 at first sight, movement along x and y frames seem somewhat oscillating in an inconsistent manner. However, when y is plotted with respect to x, we will get a perfect circle as in the Figure 18. The reason why it is incomplete is related with the t_{span} specified in the *Input Configuration*. After all, one can conclude that one-wheel rotation results in a circular path for the vehicle. We can generalize this for all other wheels as well. I think, these 2 experiment illustrates the edge cases for our omnidirectional vehicle. Other input configuration combinations would produce motions somewhat between these two type of motions.

In order to replicate experiments of this question, run the command line:

```
[t,y] = ode45(@(t,y) odeq7(t,y,w),tspan, initial_condition);
```

after specifying each parameter as given in the corresponding ***Input Configurations****. This will produce a* $[x\ y\ \theta]$ *array, which each corresponds to a time-sequence. Plotting commands are below:*

```
plot(t,y(:,1));
plot(t,y(:,2));
plot(t,y(:,3));
plot(y(:,1),y(:,2));
```