Version 2024.1

Assignment 1

Mathematics for Robotics

C. Gummaluru, A. Shoelling *University of Toronto*

This assignment is part of the *Fall 2024* offering of *ROB310: Mathematics for Robotics* at the University of Toronto. Please review the information below carefully. Submissions that do not adhere to the guidelines below may lose marks.

Release Date: ..... September 16, 2024 - 00:00
Due Date: ..... September 22, 2024 - 23:59
Weight: ..... 3.75% or 0%
Late Penalties: ..... -10% per day and -100% after 3 days

Submission Information:

- Download this document as a PDF and fill in your information below.
- Answer the questions to the best of your ability.
- Typeset your answers, labelling the questions identically to this document. We recommend using LaTeX, but will accept the use of other typesetting software (such as Microsoft Word or Google Docs).
- Save your typeset answers as a PDF and combine it with the cover-sheet of this document. Do NOT include the question descriptions.
- Submit your PDF on Crowdmark titled `lastname-firstname-rob310f24-a1.pdf`.

Name (First, Last): ..... Stanshall, Jaden

UTORId (e.g., gummalur): ..... stanshal

Acknowledgement:

I hereby affirm that the work I am submitting for this assignment is entirely my own. I have not received any unauthorized assistance, and I understand that plagiarism or any other form of academic dishonesty is a violation of the University's academic integrity policy.

I agree to the above statement..... ✔

Date (dd/mm/yyyy): 21/09/2024

1 Assignment 1

1.1 Part 1.1

1.1.A

Given the model of the system:

$$m\ddot{x}(t) = b\dot{x}(t) + u(t) + mg\sin\theta \tag{1}$$

and defining velocity as:

$$v(t) = \dot{x}(t) \tag{2}$$

We choose our system's internal state to be:

$$x = [\dot{x}] = [v] \tag{3}$$

We choose our system's input to be:

$$u = [u + mgsin(\theta)] \tag{4}$$

We choose our system's output to be:

$$y = [\dot{x}] = [v] \tag{5}$$

This gives the following equivalent model for the system:

$$\dot{x} = \left[\frac{-b}{m}\right] x + \left[\frac{1}{m}\right] y \tag{6}$$

$$y = [1]\, x + [0]\, u \tag{7}$$

So:

$$A = \left[\frac{-b}{m}\right], \qquad B = \left[\frac{1}{m}\right], \qquad C = [1], \qquad D = [0] \tag{8}$$

A discrete time representation of this system would have the form:

$$v_{k+1} = A_D v_k + B_D u_k \tag{9}$$

$$y_k = v_k \tag{10}$$

A_D and B_D are defined as:

$$A_D = e^{AT}, \qquad B_D = B\int_0^T e^{A\tau} d\tau \tag{11}$$

We find A_D analytically:

$$A_D = e^{AT} = e^{\frac{-bT}{m}} \tag{12}$$

We find B_D analytically:

$$B_D = B\int_0^T e^{A\tau} d\tau = \frac{1}{m}\int_0^T e^{\frac{-b\tau}{m}} d\tau = \frac{1}{m}\left[\frac{e^{\frac{-b\tau}{m}}}{\frac{-b}{m}}\right]_0^T = \frac{1 - e^{\frac{-bT}{m}}}{b} \tag{13}$$

Subbing in A_D and B_D, we get the following model for our system:

$$v_{k+1} = \left[e^{\frac{-bT}{m}}\right] v_k + \left[\frac{1 - e^{\frac{-bT}{m}}}{b}\right] u_k \tag{14}$$

1.1.B

Given:

$$m = 1, \quad b = 0.1, \quad T = 0.01, \quad \theta = 0^\circ, \quad g = 10 \tag{15}$$

We sub these values into our equations for A_D and B_D to find:

$$A_D = 0.9990004998 \tag{16}$$

$$B_D = 0.009995001666 \tag{17}$$

1.1.C

We compute A_D and B_D using **expm** within **scipy**:

```
def compute_robotic_system_matrices_expm(A, B, T):
  M = np.concatenate((A,B), axis = 1)
  l = max(M.shape[0], M.shape[1]) # l = 2
  M.resize(l, l)

  # compute the matrix exponential of M*T
  Md = sp.linalg.expm(M * T)

  # extract A_d and B_d
  A_d = Md[:A.shape[0], :A.shape[1]]
  B_d = Md[:A.shape[0], A.shape[1]:]

  return A_d, B_d
```

```
A = np.array([[-0.1]])
B = np.array([[1]])
T = 0.01

A_d, B_d = compute_robotic_system_matrices_expm(A, B, T)
print(A_d, B_d)
```

This returns:

$$A_D = [0.9990005] \tag{18}$$

$$B_D = [0.009995] \tag{19}$$

These computed values for A_D and B_D are the same as the analytical solutions we calculated, only will less precision.

1.1.D

We compute A_D and B_D using **cont2discrete** within **scipy** using a zero-order hold as the discretization method.

```
def compute_robotic_system_matrices_cont2discrete(A, B, T):
  d_system = cont2discrete((A, B, C, D), T, method = 'zoh')
  A_d, B_d = d_system[0], d_system[1]

  return A_d, B_d
```

```
A = np.array([[-0.1]])
B = np.array([[1]])
C = np.array([[1]])
D = np.array([[0]])
T = 0.01

A_d, B_d = compute_robotic_system_matrices_cont2discrete(A, B, T)
print(A_d, B_d)
```

This returns:

$$A_D = [0.9990005] \tag{20}$$

$$B_D = [0.009995] \tag{21}$$

These results are numerically identical to the results computed using **expm** in Part 1.1.C and are the same as the analytical results computed in Part 1.1.B

1.1.E

We now simulate the system using A_D and B_D with:

$$m = 1, \quad b = 0.1, \quad T = 0.01, \quad \theta = 0°, \quad g = 10 \tag{22}$$

$$u_k = 1, \quad k = 1, ..., 10000, \quad v_0 = 0 \tag{23}$$

We will also use the function **compute_robotic_system_matrices_expm()** defined in Part 1.1.C to compute matricies A_D and B_D. First, we define our sumulation function:

```
def simulate(x_0, u_0, A, B, C, D, simulation_time, T, theta):

  simulation_steps = int(simulation_time // T) # the number of simulation steps

  state_history = [x_0]
  u = u_0

  for _ in range(simulation_steps):
    x = np.matmul(A, state_history[-1]) + np.matmul(B, u)
    state_history.append(x)
    y = np.matmul(C, x) + np.matmul(D, u)
    theta = theta # radians
    u = np.array([[1 + 1*10*np.sin(theta)]])

  state_history = np.array(state_history)[:,:,0]

  time = np.linspace(0,simulation_time,num=len(state_history[:,0]))
  car_velocity = state_history[:,0]

  return time, car_velocity
```

And our values for A, B, C, D, T:

```
A = np.array([[-0.1]])
B = np.array([[1]])
C = np.array([[1]])
D = np.array([[0]])
T = 0.01
theta = 0
```

Then compute the system matricies:

```
A_d, B_d = compute_robotic_system_matrices_expm(A, B, T)
```

We define our initial conditions:

```
x_dot0 = 0 # the initial velocity of the car
x_0 = np.array([[x_dot0]])
u_0 = np.array([[1]])
```

Then run our simulation and extract results:

```
time, car_velocity = simulate(x_0 = x_0, u_0 = u_0, A = A_d, B = B_d, C = C, D = D,
    simulation_time = 100, T = T, theta = theta)
```

The following diagram is a plot of the velocity of the car $v = \dot{x}$ over time:

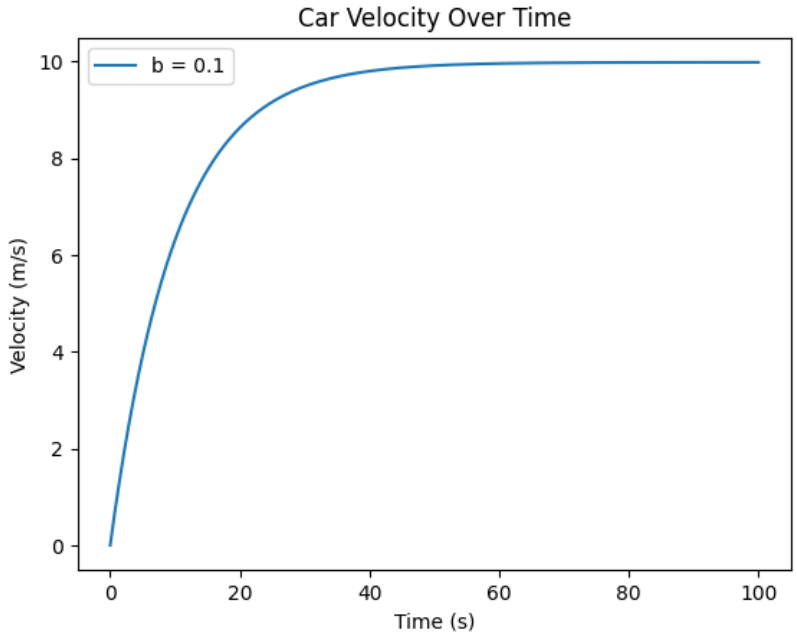

Figure 1:

The car is accelerating for the entire interval of time over which we have simulated and approaches a terminal velocity of $10m/s$. The b term is the coefficient of friction, quantifies the nature of the force of friction the car faces as a result of real-world factors; wind resistance and road roughness are a few examples of such real-world factors.

1.1.F

We will simulate the system again with one minor substitution. We will simulate the system with the following coefficient of friction which is more representative of real-world factors:

$$b = 0.4 \tag{24}$$

This coefficient of friction is higher than what was previously simulated, so we should expect to see the car approach a lower terminal velocity. We redefine matrix A:

```
A_1 = np.array([[-0.4]])
```

And compute a new A_D and B_D:

```
A_d_1, B_d_1 = compute_robotic_system_matrices_expm(A_1, B, T)
```

Then we simulate the system once again:

```
time_1, car_velocity_1 = simulate(x_0 = x_0, u_0 = u_0, A = A_d_1, B = B_d_1, C = C, D =
    D, simulation_time = 100, T = T, theta = theta)
```

And plot the results together with the results of the simulation where $b = 0.1$:

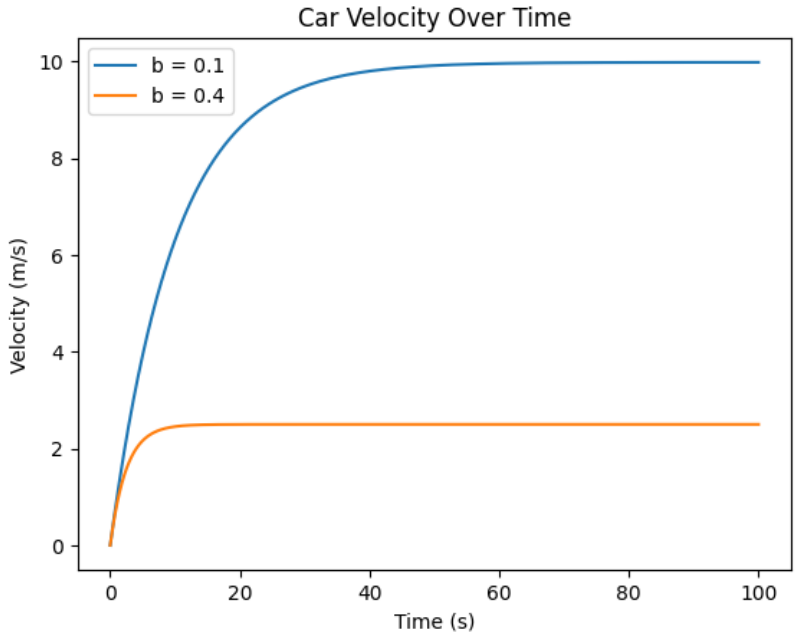

Figure 2:

We can also compute and plot the error between our earlier simulation ($b = 0.1$) and our more accurate simulation ($b = 0.4$):

```
car_velocity_error = []
for i in range(len(car_velocity)):
  error_i = car_velocity[i] - car_velocity_1[i]
  car_velocity_error.append(error_i)
```

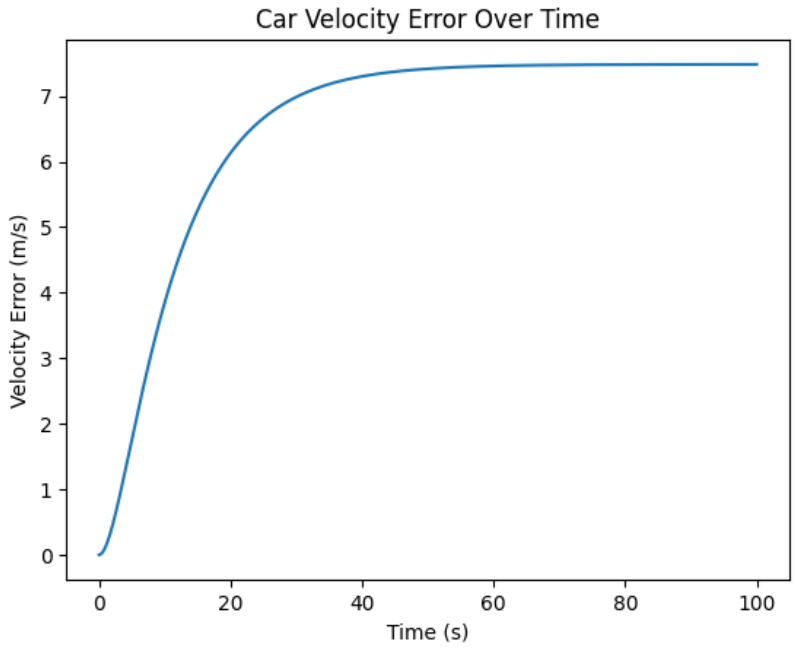

Figure 3:

We can see in **Figure 3** that the difference in terminal velocity is close to $8m/s$ between our earlier simulated model ($b = 0.1$) and our more accurate model ($b = 0.4$).

1.1.G

We will simulate our system once again with all previous parameters, save for a new value for θ:

$$\theta = 30^\circ \tag{25}$$

This theta value indicates that the road is declined by 30°. The following code will simulate the new system and extract results:

```
theta = math.radians(30)
time_2, car_velocity_2 = simulate(x_0 = x_0, u_0 = u_0, A = A_d, B = B_d, C = C, D = D,
    simulation_time = 100, T = T, theta = theta)
```

The following figure is a plot of the car's velocity over time, showing both the $\theta = 0^\circ$ simulation and the $\theta = 30^\circ$ simulation.

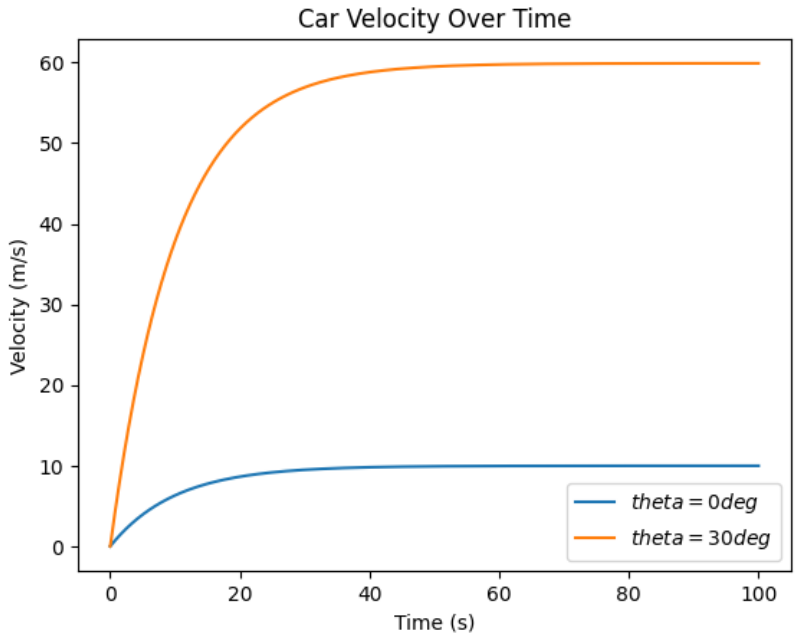

Figure 4:

As seen in **Figure 4**, the car approaches a much higher terminal velocity when it is driving down a slope of 30° as compared to when it was driving on a flat road. This is because there is a new force that must be equalized by the car's friction ($b = 0.1$) before the car can reach it's terminal velocity. When $\theta = 0°$, the term in the model associated with force on the car due to gravity is 0 (as $sin(0) = 0$), but when $\theta = 30°$, the term associated with force due to gravity is greater than zero, and thus the car moves faster.

1.2 Part 1.2

1.2.A

To show that:

$$f(x) = x + acos(x) = 0, \qquad a \in \mathbb{R} \tag{26}$$

has at least one solution, we begin by showing that $f(x)$ is continuous for all $x \in \mathbb{R}$.

1. Both $f(x) = x$ and $f(x) = cos(x)$ are defined for all real numbers
2. $\lim_{x \to a} x$ and $\lim_{x \to a} cos(x)$ both exist.
3. $\lim_{x \to a} x = a$ and $\lim_{x \to a} cos(x) = cos(a)$.

Since both $f(x) = x$ and $f(x) = cos(x)$ are continuous for all $x \in \mathbb{R}$, the sum of these two functions is also continuous. So $f(x) = x + acos(x)$ is continuous.

Now that we know $f(x) = x + acos(x)$ is continuous, we can use the Intermediate Value Theorem to show that $f(x) = 0$ has at least one solution. If we can find values for a and b such that:

$$\lim_{x \to a} f(a) < 0 < \lim_{x \to b} f(b) \tag{27}$$

then we have shown there must be a point at which $f(x)$ crosses the **x-axis**, and thus, there must exist a solution to $f(x) = 0$.

We choose:

$$a = -\infty, \qquad b = \infty \tag{28}$$

Thus,

1. $\lim_{x\to\infty} x + acos(x) = \infty$
2. $\lim_{x\to-\infty} x + acos(x) = -\infty$

And since:

$$-\infty < 0 < \infty, \tag{29}$$

by the Intermediate Value Theorem, there must be at least one solution to $f(x) = x + acos(x) = 0$.

1.2.B

We will now use the Bisection Root-Finding Method to find the root of $f(x) = x + acos(x)$ which lies on the interval $x = [-5, 5]$.

The following function is a Python implementation of the Bisection Method pseudocode discussed in lecture:

```
def find_root_bisection(f, a, b, tol):
  i = 0 # iteration count
  while abs(b - a) > tol:
    i += 1 # increase iteration count
    c = (a + b) / 2
    if f(c) == 0:
      return c
    elif f(a) * f(c) < 0:
      b = c
    else:
      a = c
  return c, i
```

We define our function as follows (with $a = 1$):

```
def f(x):
  a = 1
  return x + a*math.cos(x)
```

Then, we use the Bisection Method to find the root of $f(x)$ with an accuracy of 10^{-4}, and count how many iterations of the Bisection Method are required for such accuracy.

We choose the starting interval of our Bisection Method to be:

$$x = [-5, 5] \tag{30}$$

```
a = -5
b = 5
tol = 0.0001
```

Then we use our Bisection Method function to compute the root with the specified accuracy:

```
root, count = find_root_bisection(f, a, b, tol)
print(root)
print(count)
print(f(root))
```

The above will output:

$$f(root) = f(-0.7390594482421875) = 4.29864359413612^{-05} \tag{31}$$

We can see the result of $f(root)$ is very close to 0, and so the Bisection Method produced a very close approximation of the root of $f(x)$.

We also see, from the count variable within my Python implementation, it takes 17 iterations of the Bisection Method to compute the root of $f(x) = x + acos(x)$ to an accuracy of 10^{-4}.