Model Predictive Control - Assignment 1
Group 2: Ali Ghasemi, aghasemi@kth.se
Sofia Simões, ssimoes@kth.se

Problem: Q1.

Using the expressions provided in the handout, the continuous-time system matrices

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ v_x \cdot \frac{L_r}{L_f + L_r} \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad \text{and} \qquad D = 0$$

were incorporated into the given function `_compute_system_matrices` as shown below:

```
    def _compute_system_matrices(self,v_ref : float = 20.0):
        """
        Compute the system matrices A, B, C, D for the linearized
           continuous-time vehicle model.

        :param v_ref: Reference velocity for the vehicle (m/s)
        :type v_ref: float
        """

        k = self.Lr / self.L

        # todo: set the system matrices A, B, C, D and remove error (
           vehicle properties are available under self.Lf, self.Lr,
           self.mass, self.max_acc, self.v_ref)

        A = np.array([[0.0, 1.0],
                      [0.0, 0.0]], dtype=float)

        B = np.array([[0.0],
                      [self.v_ref * k]], dtype=float)

        C = np.array([[1.0, 0.0]], dtype=float)

        D = np.array([[0.0]], dtype=float)

        return A, B, C, D
```

Problem: Q2.

The continuous-time system is:

$$\dot{x}(t) = Ax(t) + Bu(t),$$

with

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad \text{and} \qquad B = \begin{bmatrix} 0 \\ b \end{bmatrix},$$

where

$$b = v_x \cdot \frac{L_r}{L_f + L_r}.$$

The discretized system with sampling time T under zero-order hold (ZOH) is

$$x_{k+1} = A_d x_k + B_d u_k,$$

with

$$A_d = e^{AT}, \qquad B_d = \int_0^T e^{A\tau} B \, d\tau.$$

We can compute e^{At} using inverse Laplace transform:

$$\mathcal{L}\left\{e^{AT}\right\} = (sI - A)^{-1} = \begin{bmatrix} s & -1 \\ 0 & s \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s} & \frac{1}{s^2} \\ 0 & \frac{1}{s} \end{bmatrix}.$$

Taking inverse Laplace term by term gives:

$$e^{At} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}.$$

Thus,

$$A_d = e^{AT} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}.$$

For input matrix:

$$B_d = \int_0^T e^{A\tau} B \, d\tau = \int_0^T \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ b \end{bmatrix} d\tau = \int_0^T \begin{bmatrix} b\tau \\ b \end{bmatrix} d\tau = \begin{bmatrix} \frac{1}{2} bT^2 \\ bT \end{bmatrix}.$$

The final discrete-time system state-space model is:

$$A_d = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}, \qquad B_d = \begin{bmatrix} \frac{1}{2} bT^2 \\ bT \end{bmatrix}, \qquad C_d = C, \qquad D_d = D.$$

For the given parameters of the assignment:

$$L_f = 1.55 \text{ m}, \quad L_r = 1.35 \text{ m}, \quad v_x = 20 \text{ m/s}, \quad T = 0.1 \text{ s},$$

we obtain

$$b = v_x \cdot \frac{L_r}{L_f + L_r} = 20 \cdot \frac{1.35}{2.90} = 9.3103.$$

Therefore,

$$A_d = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix} \qquad \text{and} \qquad B_d = \begin{bmatrix} 0.0466 \\ 0.9310 \end{bmatrix}.$$

To compare the values with the ones from `c2d` function, we can run the code below:

```
# Get the system discrete-time dynamics
vehicle.c2d()
Ad, Bd, Cd, Dd = vehicle.get_discrete_dynamics()
print("Q2 - Discrete Time Matrices obtained with c2d()")
print("Ad =", Ad)
print("Bd =", Bd)
```

The numerical results agree with the `c2d` function output, consistent within the limits of numerical precision.

Problem: Q3.

For the continuous-time model with

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ b \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad \text{and} \qquad D = 0,$$

the transfer function from the control input u to the output y is given by:

$$G(s) = C(sI - A)^{-1}B + D = \frac{b}{s^2}, \qquad \text{with} \qquad b = v_x \cdot \frac{L_r}{L_f + L_r}.$$

Thus, the continuous-time system has two poles at $s = 0$ (double integrator) and no finite zeros, which can be verified in Figure 1, obtained using the given Python function `poles_zeros`.

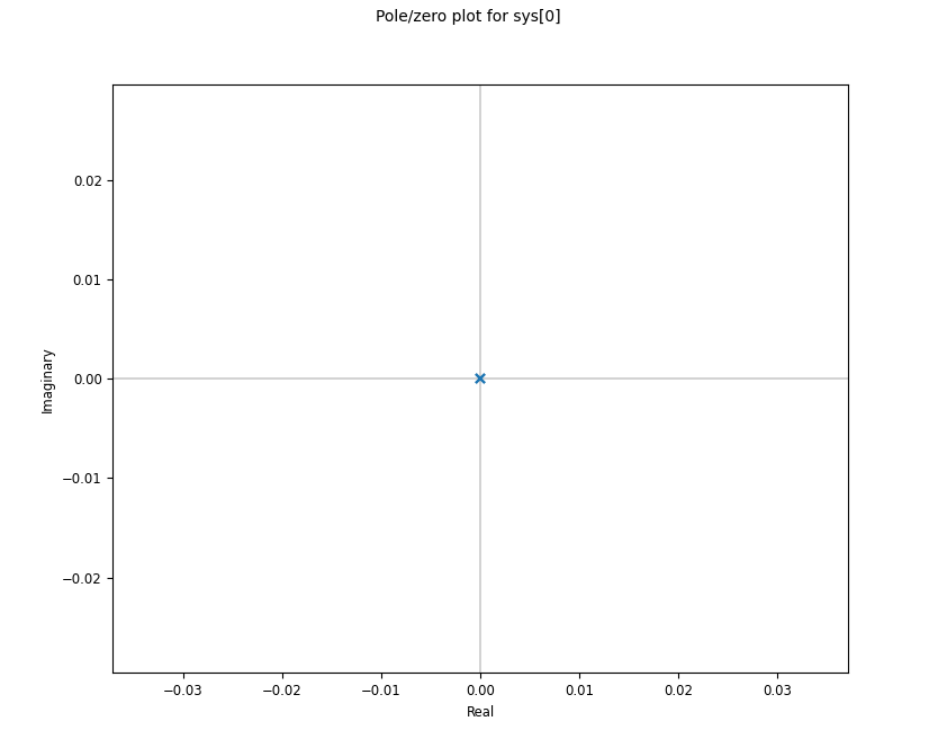

Figure 1: Pole and zero plot for the continuous-time system.

For the discrete-time model, the two continuous-time poles at $s = 0$ map through $z = e^{sT}$ to a double pole at $z = 1$, so we expect two poles located at $z = 1$. Unlike poles, zeros do not map directly under discretization but depend on the method used. With zero-order-hold discretization of a double integrator, the resulting transfer function is

$$G(z) = b\,\frac{T^2}{2}\,\frac{z+1}{(z-1)^2},$$

which is the same obtained using the formula $G(z) = C_d(zI - A_d)^{-1}B_d + D_d$, with

$$A_d = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}, \quad B_d = \begin{bmatrix} \frac{1}{2}bT^2 \\ bT \end{bmatrix}, \quad C_d = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad \text{and} \qquad D_d = 0.$$

The previous results show one finite zero at $z = -1$ in addition to the double pole at $z = 1$. Therefore, the discrete-time system is expected to have a double pole at $z = 1$ and a single zero at $z = -1$, which matches the computed result in Figure 2, obtained using the given Python function `poles_zeros`.

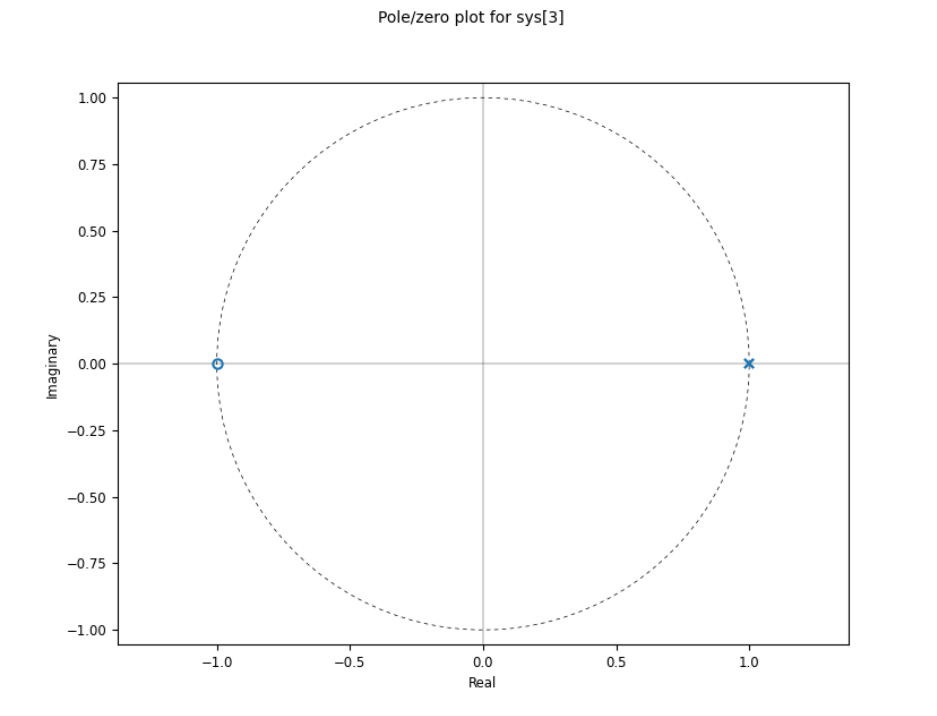

Figure 2: Pole and zero plot for the discrete-time system.

Problem: Q4.

The state-feedback controller of the form $u_k = -L(x_k - x_{\text{ref}})$ was implemented in Python by completing the provided function `control_law`. The gain matrix L was obtained by placing the poles of the closed-loop poles at the desired location using the function `place` in the function `get_closed_loop_gain`. The implementation of the control law is shown below:

```
    def control_law(self, x):
        """
        Nonlinear control law.

        :param x: state
        :type x: np.ndarray
        :param ref: cart reference position, defaults to 10
        :type ref: float, optional
        """

        if self.use_integral is True:
            self.update_integral(x)

        # todo: complete control law and remove error
```

```
        u = - self.L @ (x - self.ref)
        u = float(u)
        u = np.clip(u, -0.25, 0.25)

        return np.array([[u]])
```

We simulated the system starting from $x_0 = [-1.0\ 0.0]^\top$ and aiming for $x_{\text{ref}} = [1.0\ 0.0]^\top$. The resulting plots in Figures 3 and 4 demonstrate that all requirements are satisfied: the lateral position reaches at least 75% of the reference before 10 s, the steering signal remains within $|u| < 0.25$, the lateral velocity remains within $|v_y| < 1$ m/s, and the lane-change trajectory does not overshoot more than 1 m. These results confirm that our controller achieves a safe and smooth lane change.

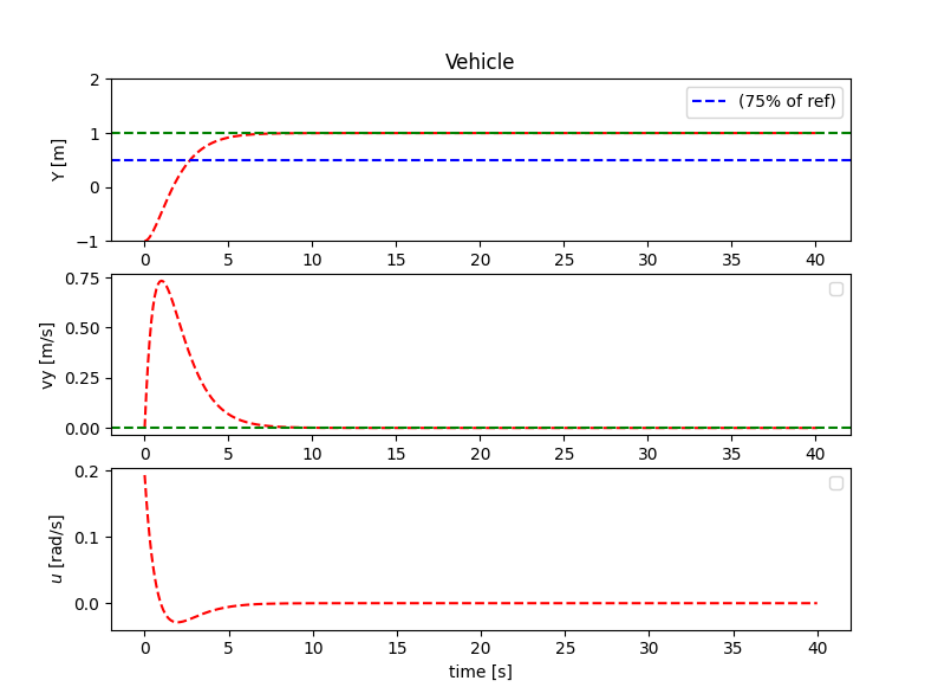

Figure 3: Simulation results for lateral position $y(t)$, lateral velocity $v_y(t)$, and steering rate $u(t)$.

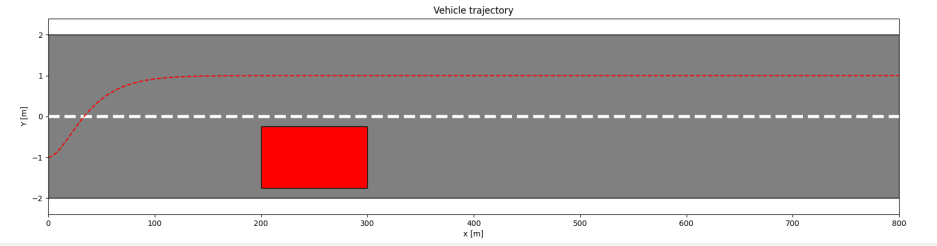

Figure 4: Vehicle trajectory.

Problem: Q5.

In this part, we considered the effect of a constant disturbance acting on the system. As expected, the pure state-feedback controller of Q4 is unable to completely reject the disturbance, leading to a steady-state offset in the lateral position (Figure 5).

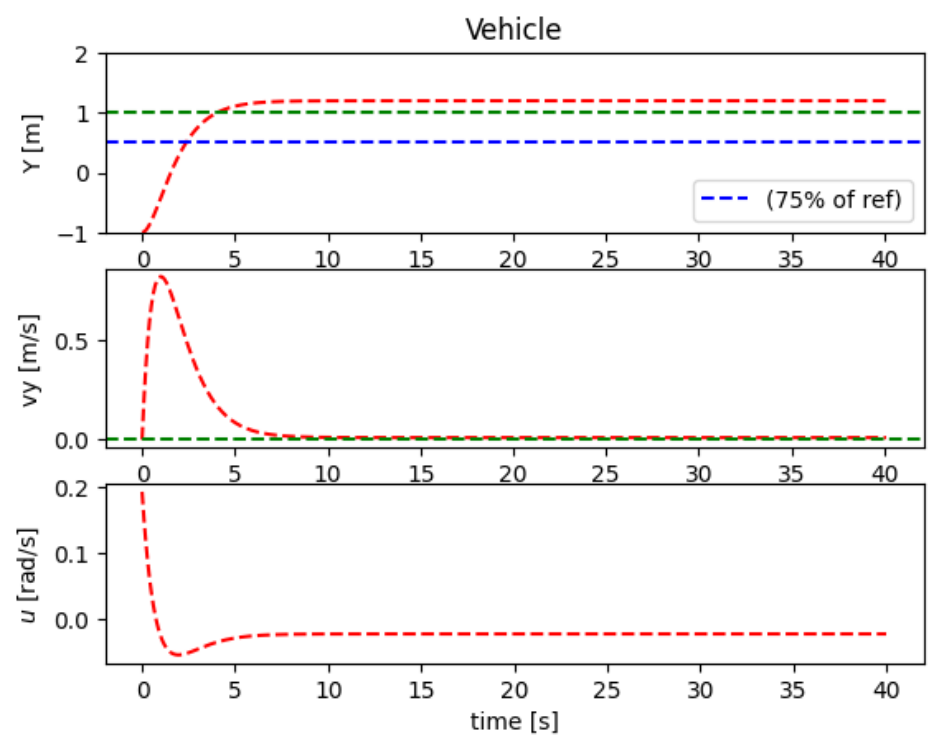

Figure 5: Response in the presence of disturbance.

To overcome this, we extended the control law with an integral term,

$$u_k = -L(x_k - x_{\text{ref}}) - K_i i_k, \qquad i_{k+1} = i_k + h\,(Y_k - Y_{\text{ref}}),$$

which drives the accumulated error to zero.
With $K_i = 0.02$, the integral action successfully compensates the disturbance while keeping all safety constraints satisfied. As shown in Figure 6, the lateral position converges smoothly to the reference without residual error. The steering input remains bounded within $|u| \leq 0.25$, and the lateral velocity satisfies $|v_y| < 1$ m/s throughout. The trajectory plot in Figure 6 further confirms that the vehicle reaches and maintains the desired lane center despite the constant disturbance.

```
    def control_law(self, x):
        """
        Nonlinear control law.

        :param x: state
        :type x: np.ndarray
        :param ref: cart reference position, defaults to 10
```

```
        :type ref: float, optional
        """

        if self.use_integral is True:
            self.update_integral(x)
        # todo: complete control law and remove error

        u = - self.L @ (x - self.ref)

        if self.use_integral:
            u -= self.Ki * self.i_term

        u = float(u)
        u = np.clip(u, -0.25, 0.25)

        return np.array([[u]])
```

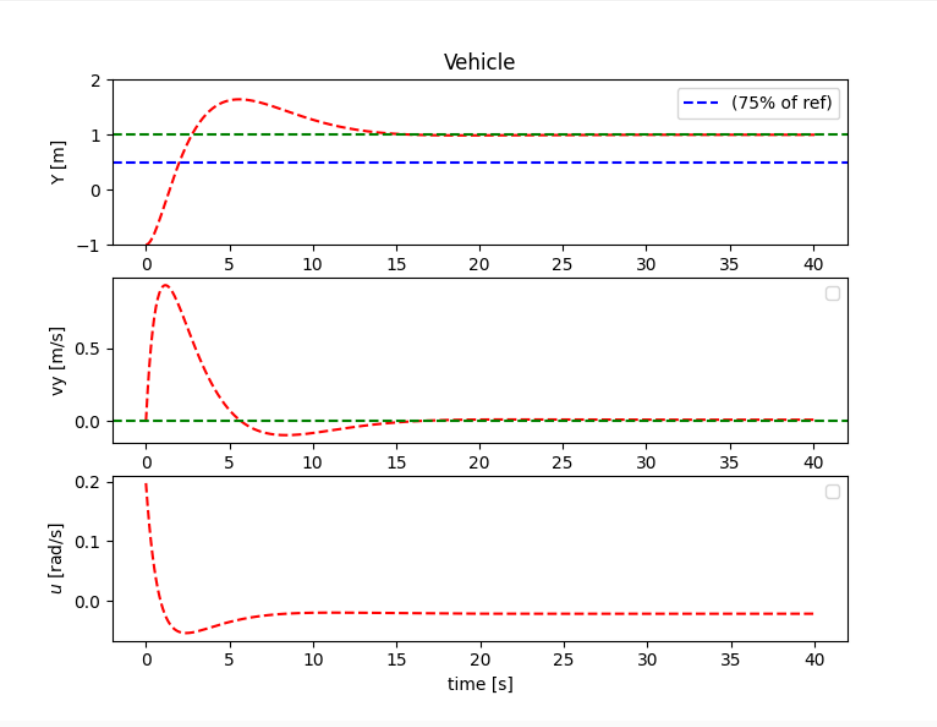

Figure 6: Simulation results for lateral position $y(t)$, lateral velocity $v_y(t)$, and steering rate $u(t)$ with constant disturbance.

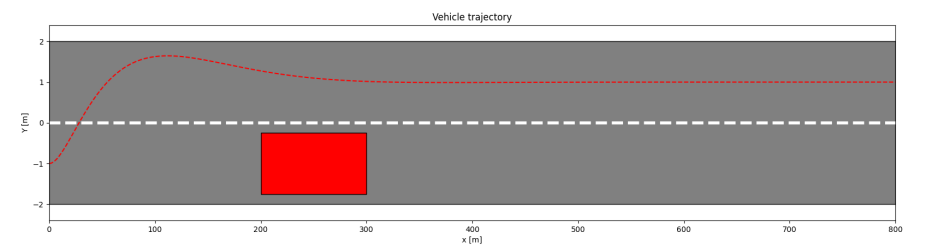

Figure 7: Vehicle trajectory with constant disturbance.