Group 2: Ali Ghasemi, aghasemi@kth.se
Sofia Simões, ssimoes@kth.se

Problem: Q1.

In this assignment, the function compute `compute_racing_line` was implemented in the Python file `racing_line.py`, as described in the handout, to compute the racing line with minimum acceleration while staying within the track boundaries. The racetrack was discretized into a given number of points, and the vehicle positions were expressed as offsets from the centerline along the normal direction. Using convex optimization, the displacements were constrained to remain within the track width, and the objective was set to minimize the total acceleration along the path. The problem was formulated in `CVXPY` and solved with `MOSEK`, yielding the optimal racing line.
The final code for this question is shown below:

```
def compute_racing_line(racetrack: RaceTrack, num_points: int = 1000):
    """
    Computes a minimum-acceleration racing line within track
       boundaries using
    convex optimization. Returns the vehicle positions.
    """
    racetrack  = racetrack
    s_values   = np.linspace(0, racetrack.length, num_points)
    s_delta    = racetrack.length / (num_points - 1)
    center     = np.array([racetrack.position(t) for t in s_values])
    num_points = num_points


    # TODO: Define cvxpy variables for vehicle positions (num_points x
        2) and
    # the centerline offsets (vector of length num_points)
    center  = np.vstack([racetrack.position(s)       for s in s_values
       ])
    normals = np.vstack([racetrack.normal_vector(s)  for s in s_values
       ])

    positions = cp.Variable((num_points, 2))
    n         = cp.Variable(num_points)

    # TODO: Add constraint on centerline offset
    constraints = [cp.abs(n) <= racetrack.track_width / 2]

    # TODO: Relate positions to centerline offset (Eq. (1))
    constraints += [positions == center + cp.multiply(n[:, None],
       normals)]

    # TODO: Define acceleration objective (Eq. (4))
    N = num_points
    e = np.ones(N)
    D2 = (np.diag(e[:-2], k=0) - 2*np.diag(e[1:-1], k=0) + np.diag(e
       [2:], k=0))
    #difference matrix
```

```
    D2 = np.zeros((N-2, N))
    for i in range(1, N-1):
        D2[i-1, i-1] = 1.0
        D2[i-1, i  ] = -2.0
        D2[i-1, i+1] = 1.0


    acc = (D2 @ positions) / (s_delta**2)

    # Objective
    objective = cp.sum(cp.norm(acc, axis=1)) * s_delta

    # TODO: Define cvxpy problem and solve it using MOSEK
    min_acc_problem = cp.Problem(cp.Minimize(objective), constraints)
    min_acc_problem.solve(solver=cp.MOSEK)

    return positions.value
```

Problem: Q2.

In the second part of the assignment, the function `compute_lap_time` was implemented to evaluate the performance of the racing line. Using the formula provided in the handout, the curvature of the trajectory was computed from the discrete velocity and acceleration vectors. Based on this curvature, the maximum admissible velocity at each point along the racing line was determined according to the tire friction model. The lap time was then obtained by summing the time intervals required to travel between successive discretization points at the corresponding maximum velocity. This implementation enables the comparison of performance between different trajectories, such as the optimized racing line and the centerline.

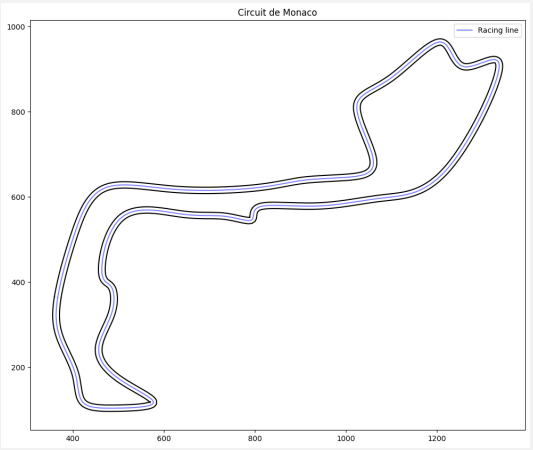
The final code for the second question is shown below:

```
def compute_lap_time(positions, g, mu, s_delta):
    v = np.diff(positions, n=1, axis=0)[:-1] / s_delta
    a = np.diff(positions, n=2, axis=0) / s_delta ** 2

    # TODO: Compute the curvature at each index i (Eq. (5))
    curvature = np.abs(a[:,1]*v[:,0] - a[:,0]*v[:,1]) / (v[:,0]**2 + v
        [:,1]**2)**(3/2)

    # TODO: Compute the absolute velocity at each index i
    V = np.sqrt(mu * g / curvature)

    # TODO: Compute the lap time
    time_deltas = s_delta / V
    lap_time = np.sum(time_deltas)

    return lap_time
```

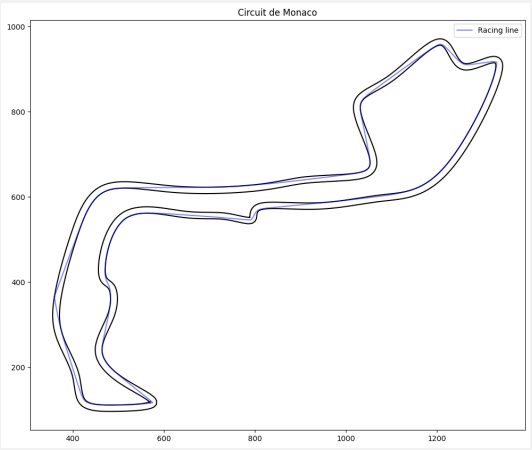
Problem: Q3.

The optimized racing line proved to be significantly faster than simply following the centerline. For the Circuit de Monaco, the optimized trajectory yielded a lap time of 51.894 s, while the centerline lap time was 69.415 s. This corresponds to an improvement of 17.521 s, or about 25.24 %.

When plotting both trajectories, the optimized racing line displays the expected racing behavior: the car approaches wide on corner entry, clips the apex, and then tracks out on exit. By increasing the effective turn radius in this way, the optimized path reduces curvature compared to the centerline, which remains tighter through turns. Since the maximum speed scales inversely with the square root of the curvature, $V_{\max} \propto \kappa^{-1/2}$, the reduced curvature of the optimized trajectory allows higher speeds through corners and thus explains the reduction in lap time.



(a) Centerline trajectory.



(b) Optimal trajectory.

Figure 1: Comparison between two different trajectories.